IBExpert and Firebird Documentation

Version 2008.09.03

copyright IBExpert KG

Getting Started

In order to start working and developing with IBExpert, it is necessary to take the following steps:

1. Download and install Firebird (Open Source database). Alternatively you may also, of course, install InterBase®.
2. Download and install IBExpert (Personal, Trial or Customer edition)
3. Registering a database (the example uses the `EMPLOYEE` database supplied with Firebird and InterBase)
4. Working with a database (based on the `EMPLOYEE` sample database).
5. IBExpert Screen: get acquainted with IBExpert and how it's set up.
6. Where to go from here: if you're just starting out, take the time to read through the documentation sources listed in this section.

Download and install Firebird

Firebird is renowned for its ease of installation and administration. Even an inexperienced user can download and install Firebird using the Installer, with just a number of mouse clicks. If you are totally new to Firebird, please first read the chapter, *Server versions and differences* to help you decide which Firebird version you need.

The current Firebird version can be downloaded free of charge from http://firebirdsql.org subject to Open Source conditions. Alternatively, use the IBExpert Help menu item Download Firebird to directly access the download website.

Simply click the *DOWNLOAD* tab and select *All Released Packages* (Source Forge). The download packages come in a variety of options according to: server type (Classic, SuperServer and Embedded), server version, platform, and incorporating the Installer or as a `ZIP` file.

Scroll down to the latest file releases and click *DOWNLOAD* to the right of the version for your platform, for example Firebird releases for Windows and Linux (most current version in March 2008 is Firebird 2.1 RC1 from January 23rd, 2008). Please refer to Posix Platforms and Windows Platforms for further information for individual platforms with regard to download and installation.

If you are new to Firebird, then go for a version using the Installer. The Zip kit is for manual, custom installs of Classic or Superserver.

A new window appears:

SOURCEFORGE.NET® 2007
COMMUNITY CHOICE AWARDS
WINNER: BEST USER SUPPORT
WINNER: BEST PROJECT FOR THE ENTERPRISE

Latest File Releases

Package	Release	Date	Notes / Monitor	Downloads
firebird	2.1 RC1 (Source)	January 23, 2008	-	Download
firebird-addons	FbConfigManager for v1.5	April 9, 2003	-	Download
firebird-aix-ppc	1.5.3-Release	November 22, 2006	-	Download
firebird-benchmarks	as3ap	February 4, 2003	-	Download
firebird-freebsd-i386	1.5.2-Release	January 4, 2005	-	Download
firebird-hpux	2.0.3-Release	January 7, 2008	-	Download
firebird-jca-jdbc-driver	2.1.3-release-src	February 25, 2008	-	Download
firebird-linux-amd64	2.1 RC1	January 23, 2008	-	Download
firebird-linux-i386	2.1 RC1	January 23, 2008	-	Download
firebird-MacOS-X/darwin	2.1 RC1	February 6, 2008	-	Download
firebird-net-provider	2.1.0	November 26, 2007	-	Download
firebird-ODBC-driver	1.2-Release	August 19, 2004	-	Download
firebird-sinixz	1.5.0-Beta1	January 29, 2003	-	Download
firebird-solaris-sparc	2.0.3-Release	November 20, 2007	-	Download
firebird-solaris-x86	2.0.3-Release	November 20, 2007	-	Download
firebird-win32	2.1 RC1	January 23, 2008	-	Download
firebird-win64	2.1 RC1	January 23, 2008	-	Download

Click on the green *Download* button to the right of the Firebird file you require. Select the file(s) you wish to download:

Package	Release (date)	Filename	Size (bytes)	Downloads	Architecture	Type
firebird-win32						
Latest	2.1 RC1	(2008-01-23 10:51)				
		Firebird-2.1.0.17735-0_Win32_embed_pdb.zip	7457959	488	Any	.zip
		Firebird-2.1.0.17735-0_Win32_embed.zip	4667387	1291	Any	.zip
		Firebird-2.1.0.17735_0_Win32.exe	7132287	9987	i386	.exe (32-bit Windows)
		Firebird-2.1.0.17735_0_Win32_pdb.exe	10807357	569	i386	.exe (32-bit Windows)
		Firebird-2.1.0.17735-0_Win32_pdb.zip	16212789	317	i386	.zip
		Firebird-2.1.0.17735-0_Win32.zip	9800887	1778	i386	.zip
Totals:	1	6	56078666	14430		

If required, select a download server:

Specify drive and path for the download file and save.

Before you proceed with the installation (either using the Firebird Installer or manually from the ZIP file), please ensure first that there is no Firebird server already running on the machine you are about to install onto.

Installation using the Firebird Installer

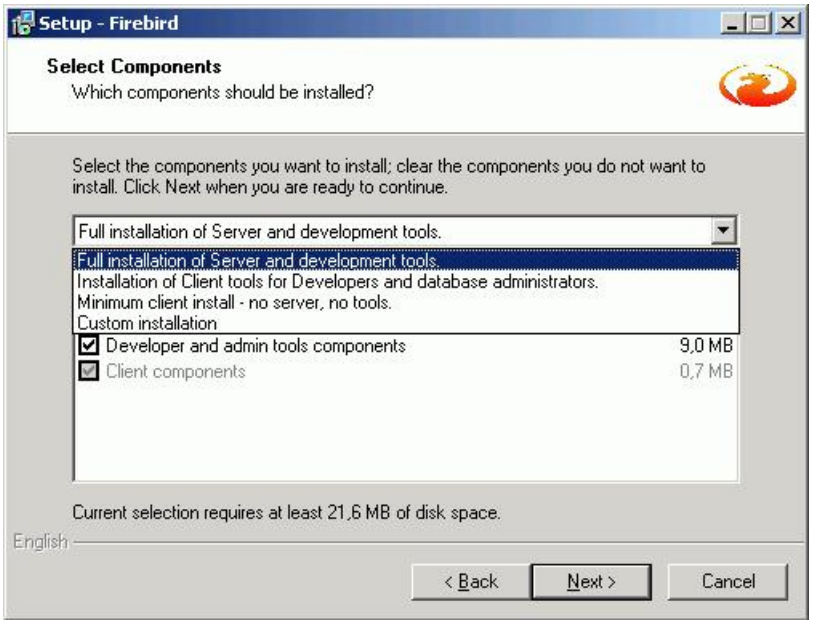
Now double-click the downloaded firebird file to start the installation. Again, please refer to Windows Platforms and Posix Platforms for installation details for the various platforms.

Read and accept the Firebird License Agreement, before proceeding further.

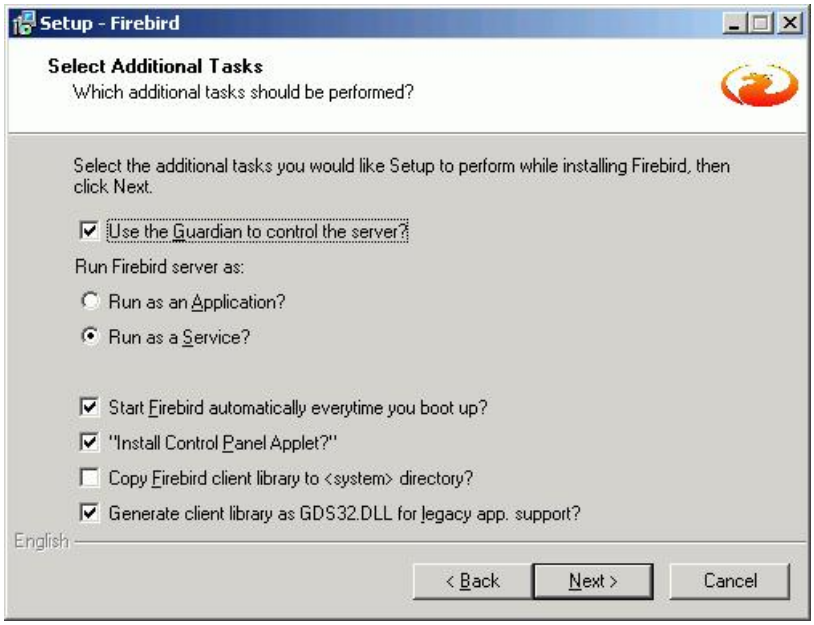
Specify the drive and path where you wish the Firebird server to be installed. Please note that the Firebird server, along with any databases you create or connect to, must reside on a hard drive that is physically connected to the host machine. It is not possible to locate components of the server or database on a mapped drive, a file system share or a network file system.

The Firebird server must be installed on the target computer. In the case of the Embedded Server version the client library is embedded in the server, this combination performing the work of both client and server for a single attached application.

Then select the components you wish to install. If you are still fairly new to Firebird, select the default option, *Full installation of Server and development tools*, checking the *Classic* or *SuperServer* option as wished.



After confirming or altering the *Start Menu folder* name (or checking the *Don't create a Start Menu* folder box), you arrive at the *Check Additional Tasks* dialog:



The Firebird Guardian: The Firebird Guardian is a monitoring utility that does nothing other than check whether the Firebird server is running or not. Nowadays it is not really necessary on modern Windows systems, as it is possible to restart the Firebird service, should it cease to run for any reason, using the operating system. Use the Windows Services (*Restore* page) to specify that every time the Firebird service stops, it should be restarted. When the service is halted, the restart can be viewed in the Windows *Event Log*.

However if the server does go down, it's important to find out what caused it. The logs need checking to trace page corruption and an immediate decision needs to be made right there and then, whether to regress backwards or move forwards. An automatic restart automatically leads to more crashes and more corruption, until the problem is noticed and causes analyzed and repaired. So consider carefully, whether you wish to have the Guardian running in the background on your database server or not.

Further parameter check options include the following:

- **Run the Firebird server as an application or service**.
- **Start Firebird automatically every time you boot up:** recommended.
- **"Install Control Panel Applet":** *Windows Vista CAUTION* If you are installing onto Windows Vista, the installer option to install the Control Panel applet must be *DISABLED* to avoid having it break the Control Panel on your Vista system.
- **Copy Firebird client library to <system> directory:** care needs to be taken here if there is more than one instance of Firebird running on the server. If the `fbclient.dll` is simply overwritten, it can cause problems for the Firebird server that is already installed and running. Instead of copying to the `\system` directory, simply move it to your application directory.
- **Generate client library as GDS32.DLL for legacy app. support:** Many programs, including for example older Delphi versions, rely on a direct access using this file name. This option can be checked to copy the file under the old name.

Should problems be encountered during installation, please refer to the Firebird Information file.

Windows platforms

On Windows server platforms - Windows NT, 2000 and XP, the Firebird service is started upon completion of the installation. It starts automatically every time the server is booted up.

The non-server Windows platforms, Windows 95, 98 and ME, do not support services. The installation starts the Firebird server as an application, protected by another application known as the Guardian. Should the server application terminate abnormally, the Guardian will attempt to restart it.

Posix platforms

As there may be significant variations from release to release of any Posix operating system, especially the open source one, it is important to read the release notes pertaining to the Firebird version to be installed. These can be downloaded from the *Download* page at http://firebird.sourceforge.net.

Please consult the appropriate platform documentation, if you have a Linux distribution supporting rpm installs, for instructions about using the RedHat Package Manager. Most distributions offer the choice of performing the install from a command shell or through a GUI interface.

For Linux distributions that cannot process rpm programs, use the `.tar.gz` kit. Again instructions are included in the release notes (see above link).

Shell scripts have been provided, but in some cases, the release notes may advise modification of the scripts as well as some manual adjustments.

`ZIP` installation

Another way to install Firebird is from a `ZIP` file. This method is more flexible for embedded installations. Download the appropiate `ZIP` file from the Firebird Download site, following the directions at the beginning of this chapter. This `ZIP` file basically contains the complete installation structure.

Name	Size	Packed	Type	Modified	CRC32
..			Ordner		
udf			Ordner	23.04.2008 08:46	
system32			Ordner	23.04.2008 08:47	
misc			Ordner	06.07.2005 00:39	
lib			Ordner	23.04.2008 10:42	
intl			Ordner	05.06.2006 03:17	
include			Ordner	23.04.2008 10:42	
help			Ordner	23.04.2008 08:36	
doc			Ordner	28.01.2005 08:52	
bin			Ordner	23.04.2008 08:45	
security2.fdb	684.032	34.229	File fdb	23.04.2008 08:36	4EBAB3C9
Readme.txt	2.824	1.067	Textdatei	28.03.2008 16:00	BFD6C23A
IPLicense.txt	24.405	7.643	Textdatei	09.07.2003 00:57	2929ADE6
IDPLicense.txt	26.519	7.947	Textdatei	09.07.2003 00:57	B414B3E7
firebird.msg	127.696	42.474	Outlook-Element	23.04.2008 08:38	4C420704
firebird.conf	22.284	7.637	Textdatei	03.12.2007 22:58	3194FE5B
aliases.conf	133	81	Textdatei	23.04.2008 10:42	6D19591C

It includes a pretty much "pre-installed" server, which you can simply copy to any directory as wished, and which you can integrate into your installation by simply calling batch files. Simply start the `install_classic.bat` or `install_super.bat`, depending upon which server you wish to install:

Name	Größe	Typ	Geändert am
fb_inet_server.exe	1.944 KB	Anwendung	23.04.2008 08:46
fb_lock_print.exe	160 KB	Anwendung	23.04.2008 08:46
fbclient.dll	384 KB	Programmbibliothek	23.04.2008 08:45
fbguard.exe	80 KB	Anwendung	23.04.2008 08:46
fbserver.exe	1.968 KB	Anwendung	23.04.2008 08:46
gbak.exe	192 KB	Anwendung	23.04.2008 08:46
gdef.exe	208 KB	Anwendung	23.04.2008 08:45
gfix.exe	72 KB	Anwendung	23.04.2008 08:45
gpre.exe	432 KB	Anwendung	23.04.2008 08:48
gsec.exe	72 KB	Anwendung	23.04.2008 08:45
gsplit.exe	15 KB	Anwendung	23.04.2008 08:46
gstat.exe	100 KB	Anwendung	23.04.2008 08:46
ib_util.dll	6 KB	Programmbibliothek	23.04.2008 08:42
icudt30.dll	1.088 KB	Programmbibliothek	23.04.2008 08:29
icuin30.dll	196 KB	Programmbibliothek	23.04.2008 08:28
icuuc30.dll	536 KB	Programmbibliothek	23.04.2008 08:28
install_classic.bat	1 KB	Stapelverarbeitung...	25.02.2008 11:26
install_super.bat	1 KB	Stapelverarbeitung...	07.09.2003 23:36
instclient.exe	15 KB	Anwendung	23.04.2008 08:47
instreg.exe		dung	23.04.2008 08:46
instsvc.exe		dung	23.04.2008 08:46
isql.exe		dung	23.04.2008 08:47
msvcp71.dll	488 KB	Programmbibliothek	18.03.2003 20:14
msvcr71.dll	340 KB	Programmbibliothek	21.02.2003 04:42
nbackup.exe	84 KB	Anwendung	23.04.2008 08:47
qli.exe	260 KB	Anwendung	23.04.2008 08:47
uninstall.bat	1 KB	Stapelverarbeitung...	07.09.2003 23:39

Typ: Stapelverarbeitungsdatei für MS-DOS
Geändert am: 07.09.2003 23:36
Größe: 84 Byte

The `instreg` utility does all the work, making the necessary entries in the right places, and installs everything required in the Registration. It usually installs the Firebird Guardian too, and finally starts the service.

This is the ideal solution for development applications which are being passed onto customers: simply pack the complete Firebird `ZIP` directory in with your application, so that when you call your Installer, the only work necessary is to call the appropiate batch file.

Performing a client-only install

Each remote client machine needs the client library that matches the release version of the Firebird server: `libgds.so` on Posix clients; `gds32.dll` on Windows clients.

Firebird versions from 1.5 onward require an additional client library, `libfb.so` or `fb32.dll`, which contains the full library. In these newer distributions, the "`gds`"-named files are distributed to maintain compatibility with third-party products which require these files. Internally, the libraries jump to the correct access points in the renamed libraries.

Also needed for the client-only install:

Windows

If you want to run Windows clients to a Linux or other Posix Firebird server, you need to download the full Windows installation kit corresponding to the version of Firebird server installed on the Linux or other server machine.

Simply run the installation program, as if you were going to install the server, selecting the *CLIENT ONLY* option in the *Install* menu.

Linux and some other Posix clients

Some Posix flavors, even within the Linux constellation, have somewhat idiosyncratic requirements for file system locations. For these reasons, not all `*x` distributions for Firebird even contain a client-only install option.

For the majority, the following procedure is suggested for Firebird versions lower than 1.5. Log in as root for this.

1. Search for `libgds.so.0` in `/opt/interbase/lib` on the machine where the Firebird server is installed, and copy it to `/usr/lib` on the client.
2. Create the symlink `libgds.so` or it, using the following command: `ln -s /usr/lib/libgds.so.0 /usr/lib/libgds.so`
3. Copy the `interbase.msg` file to `/opt/interbase`.
4. In the system-wide default shell profile, or using `setenv()` from a shell, create the `INTERBASE` environment variable and point it to `/opt/interbase`, to enable the API routines to locate the messages.

Excerpts of this article have been taken from the IBPhoenix "Firebird Quick Start Guide". Many thanks to Paul Beach (http://www.ibphoenix.com)!

Performing a minimum Firebird 1.5 client install

By Stefan Heymann (April 11th 2004)

This article describes how to run Firebird 1.5 based applications with the absolute minimum client installation required.

What you need

Your application needs access to the Firebird client library, `fbclient.dll`. The easiest way to do this is to put `fbclient.dll` in the same directory as your application's `.exe` file.

`fbclient.dll` needs access to two other DLLs: `svcp60.dll` and `msvcrt.dll`. Both are delivered together with the Windows installation of Firebird, so if you have a Firebird server installed on your development machine, you'll find these DLLs in the `bin` directory of your Firebird installation.

`msvcrt.dll` (Microsoft Visual C/C++ RunTime) is a part of Windows and resides in the `Windows\System` directory on Win9x machines and in `Windows\System32` on NT-based machines (NT4, W2K, XP, 2003). On Windows 95 and Windows 98 machines, it's too old for the `msvcp60.dll` that `fbclient.dll` uses. So you'll have to replace the `msvcrt.dll` by the one that comes with Firebird (or even a newer one).

`msvcp60.dll` can stay in your application directory.

Your application directory now looks like this:

```
<YourApp>.exe and other application files
fbclient.dll
msvcp60.dll
```

That's it. Easy!

What you have to write to the registry

Nothing - there's nothing you'll have to do to the registry.

What you have to do to the Windows\System directory

Only on Windows 95 and Windows 98 "First Edition" machines: you will need to replace `msvcrt.dll` with the newer version that comes with Firebird 1.5 (if there isn't already a new version installed).

Some version numbers of `msvcrt.dll`:

Windows 98 FE	5.00.7128	does NOT work
Windows 98 SE	6.00.8397.0	works
Firebird	1.5.0 6.00.8797.0	works
Windows XP SP1	7.0.2600.1106	works

What you have to do to your code (Delphi, IBObjects)

A "normal" InterBase access library uses `gds32.dll` as the client library. Firebird's client library is named `fbclient.dll`. If you use IBObjects (http://www.ibobjects.com/), you can set another client library name.

- Include `IB_Constants.pas` as the first unit in your `USES` clause.
- Put the following line in the `INITIALIZATION` part of your Unit: IB_Constants.IB_GDS32 := 'fbclient.dll';
- This line must be executed before the first database connect is performed.

Installing multiple instances with the Firebird Instance Manager

Pre-Firebird 2.1: If you already have a Firebird version installed on your machine, then you can subsequently rename the installation using the `fbinst` tool, which can be downloaded from: http://www.ibexpert.com/download/firebirdinstancemanager/. For example, rename your existing Firebird to

MyFirebirdVersion and then install the new Firebird version without any problems. The Firebird Instance Manager was developed by Simon Carter. It isn't an IBExpert tool but it is extremely helpful if you find yourself in such a situation.

Since Firebird 2.1 the Installer offers the possibility to install multiple instances.

IBExpert introduced its own IBExpertInstanceManager as one of the HK-Software Service Control Center? services in version 2008.08.08.

See also:
Download Firebird / Purchase InterBase
Firebird License Agreement
Copy of Firebird Information File
FirebirdClassicServerVersusSuperServer
Firebird SQL
Installation and the various Firebird documentation and articles found here: Documentation

Server versions and differences

Firebird is available for various platforms, the main ones are currently 32-bit Windows, Linux (i586 and higher, and x64 for Firebird 2.0 on Linux), Solaris (Sparc and Intel), HP-UX (PA-Risc), FreeBSD and MacOS X. Main development is done on Windows and Linux, so all new releases are usually offered first for these platforms, followed by other platforms after few days (or weeks).

There is also a choice of server architecture: Classic server or SuperServer. If you're not sure after reading this chapter, whether the Classic server or the SuperServer better meets your needs, then install the SuperServer.

Classic server

The Firebird Classic server offers multiple processes per connection and SMP (Symmetric Multi-Processing) support. Each connection uses one process. It supports multi-processor systems but no shared cache. I.e. each user connecting and requesting data, will have his/her data pages loaded into the cache, regardless of whether other users' request have already caused the server to load these pages. Which of course leads to a higher RAM necessity. However, as RAM and cache requirements are relevant to the size of the database file and the drive on which it is stored, the effects of this cache connection architecture doesn't necessarily have to be a bad thing.

The current Firebird 2.0.3 Classic Server is an excellent server. Should you have sufficient working memory, we recommend you use the Classic Server and set the cache per user somewhat lower.

Further information regarding the Classic server can be found in the *Classic Server versus SuperServer* article, in the InterBase Classic architecture chapter.

SuperServer

The Firebird SuperServer has one process and multiple threads, but no SMP (Symmetric Multi-Processing), i.e. a dual-core machine. It serves many clients at the same time using threads instead of separate server processes for each client. Multiple threads share access to a single server process, improving database integrity because only one server process has write access to the database. The main advantage is however that all connected users share the database cache. If a data page has already been loaded for one user, if the second user needs to access data on the same page, it doesn't need to be reloaded a second time into the cache. For further information regarding the SuperServer, please refer to the *Classic Server versus SuperServer* article, in the InterBase SuperServer architecture chapter.

Embedded server

The Embedded server allows only one local process per database, which of course means that it is unsuitable for a web server! The Firebird 2.1 Embedded Server version provides a useful enhancement: the client library is embedded in the server, this combination performing the work of both client and server for a single attached application. Only a few files are required without installation. It mainly consists of a slightly larger `fbclient.dll`, which is capable of providing the database server service to all installations. It is not necessary to install or start anything. This is particularly advantageous, for example, in the following situation:

You have an accounting application in the old 1997 version that you need to start today to view old data that was created and processed using this version. Normally you would have to search for the old version, install it, and - if for whatever reason it doesn't work anymore (or maybe you never managed to find it in the first place!) - you can't get to your data. Solution: pack your accounting application onto a DVD together with the correct Firebird embedded version. You can then start the application directly from the DVD without having to search and install anything. This is particularly useful when archiving data.

Firebird is, by the way, one of the few database systems that can read a database on a read-only medium.

Firebird 3.0 - the best of both worlds

Firebird 3.0 is intending to combine the advantages of both Classic and SuperServer: a SuperServer with SMP (Symmetric Multi-Processing) support. It will offer the shared cache, at the same time using multiple CPUs.

See also:
Firebird Classic Server versus SuperServer
Installing on Linux

Configuring Firebird

Before we take a look at the two Firebird configuration files, we would like to point out that the most frequently asked question regarding these subjects is, "I've changed the parameter in the `firebird.conf`/`aliases.conf` and nothing's happened!" The simple solution is: remove the hash (#)! It's the symbol used for commenting.

`aliases.conf`

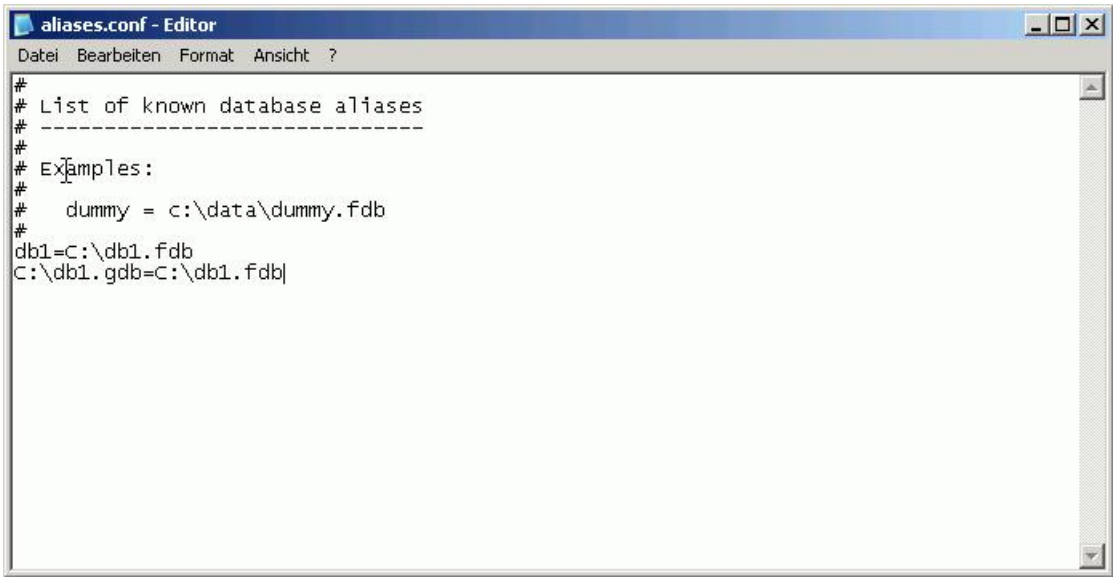
An alias is a pseudonym for the database connection string and database file name. The full connection string usually consists of the server name (or localhost) followed by the drive and path to the database file, with the database file name concatenating on the end. This informs the client, where he needs to send his data packets and access server data.

For security reasons it is not alway desirable for each client user to see the full connection string, and there are obvious problems which arise when the database is moved to another drive or machine, as each client has to be informed of the new connection string. For these reasons it is recommended to give databases an alias name. All alias names are set in `aliases.conf`. There are no syntactical restrictions to the naming of aliases.

Using an alias, users are not able to see where the database really is and, should it be relocated, the new connection string only needs to altered once in the `aliases.conf`. Let's look at an example:

The alias `db1` should refer to the database name, `db1.fdb`.

```
db1=c:\path\db1.fdb
```



This user alias has been specified for the database server. The client can also define such an alias connection when registering the database or subsequently in the IBExpert's Database Registration Info. The connection string is:

```
servername:aliasname
```

If the user wishes to connect to `db1`, he simply needs to enter

```
localhost:db1
```

in the *Database Alias* field. The `aliases.conf` file shows the server which database the client wishes to connect to.

When working with IBExpert, a database alias can be specified when registering the database. Refer to Register Database / Alias for further information.

Resolving the XP Windows System Restore problem

Windows XP has the unfortunate tendency to consider all files with the `.GDB` suffix to be a constituent of the Windows System Restore. This means that when you try to open your `DB1.GDB`, XP (default setting) first decides to make a copy of the file (just in case you need to restore it at some point), not allowing you access until it's completed. In the case of large database files, you can imagine how long this can take!

If you don't want to rename your database files just to suit Microsoft, then simply create an alias:

```
C:\db1.gdb = C:\db1.fdb
```

firebird.conf

Possible file locations are set in `firebird.conf`. The full set of `firebird.conf` parameters are described in detail in the `firebird.conf` file. The server needs to be restarted following any changes made in the `firebird.conf` for them to become valid. The following describes briefly the most important parameters:

RootDirectory

If you are using several installations of Firebird servers, use the `RootDirectory` parameter to specify where the active Firebird server can be found.

DatabaseAccess

An alias entry needs to exist. If a path is entered here, database files may only be stored in this path or its subdirectories.

```
DatabaseAccess = NONE
```

means that only file locations set in `aliases.conf` are available. The server can't access any other entries. This is a great security feature, because even when someone has a user name on the database server, he cannot create a database file, because it is not possible to specify an alias remotely.

ExternalFileAccess

Firebird has a mechanism enabling a table to be created externally, (i.e. not in the database), using the command:

```
create table external file
```

In order to allow such external files it is necessary to explicitly activate the `ExternalFileAccess` parameter. Options include: `None`, `Full` or `Restrict`. If you choose `Restrict`, provide a ';'-separated trees list, where external files are stored. Default value `None` disables any use of external files on your site.

UdfAccess

User-defined functions are used in Firebird to complement and extend the Firebird server's language. This parameter specifies where UDFs can be found. They are usually to be found in the subdirectory `/UDF`, and should - if possible - remain there. `UdfAccess` may be `None`, `Full` or `Restrict`. If you choose `Restrict`, provide a ';'-separated trees list, where UDF libraries are stored.

TempDirectories

Here you can specify where temporary files should be created. When the Firebird server receives a query including `ORDER BY` or similar, without an index, then Firebird has to sort the data somewhere. Firebird has a so-called Sort Buffer, which is principally a memory area where such sorting processes can be performed. If however you have a sorting operation that is 10 GB, Firebird needs somewhere to do this. From a certain size, when the Sort Buffer is no longer sufficient, it moves the job out into a temporary file, and you can specify here where these temp files should be.

Because of the intense batting backwards and forwards, you need to know where your temp file is in relation to your database. As soon as you need a temp file, it's because you don't have enough RAM or you've exceeded your internal limits. By its very nature, it's going to be reading things from the database cache and wanting to put things in the temp directory. So keeping those on separate disks will make a big difference. And you want to know where they are, to see how big they're getting.

What do you do if your database crashes mit-sort file? The temp files just sit there. So if you your system hangs and you need to reboot, you could suddenly have a lot of temp files. While they're being used they have a handle on them, so if you are allowed to delete or rename them, then it's fine because they're orphans.

The default value is determined using `FIREBIRD_TMP`, `TEMP` or `TMP` environment options. Every directory item may have optional size argument to limit its storage, this argument follows the directory name and must be separated by at least one space character. If the size argument is omitted or invalid, then all available space in this directory will be used.

Examples

```
TempDirectories = c:\temp;d:\temp
```

or

```
TempDirectories = c:\temp 100000000;d:\temp 500000000;e:\temp
```

DefaultDbCachePages

This influences the cache by setting the number of pages from any one database that can be held in the cache at once. By default, the SuperServer allocates 2048 pages for each database and the Classic allocates 75 pages per client connection per database. Before altering either of these values please refer to *Page size* and *Memory configuration*.

RemoteServiceName

This is the TCP Service name to be used for client database connections. It is only necessary to change either the `RemoteServiceName` or `RemoteServicePort`, not both. The order of precendence is the `RemoteServiceName` (if an entry is found in the `services.` file) and then the `RemoteServicePort`.

You don't need to change this if it's your only install.

E.g. `RemoteServiceName = gds_db`

RemoteServicePort

This is the TCP Port number to be used for client database connections. It is only necessary to change either the `RemoteServiceName` or `RemoteServicePort`, not both. The order of precendence is the `RemoteServiceName` (if an entry is found in the `services.` file) then the `RemoteServicePort`.

You don't need to change this if it's your only install.

E.g. `RemoteServicePort = 3052`

RemoteBindAddress

Allows incoming connections to be bound to the IP address of a specific network card. It enables rejection of incoming connections through any other network interface except this one. By default, connections from any available network interface are allowed.

CpuAffinityMask

This parameter only applies to SuperServer on Windows.

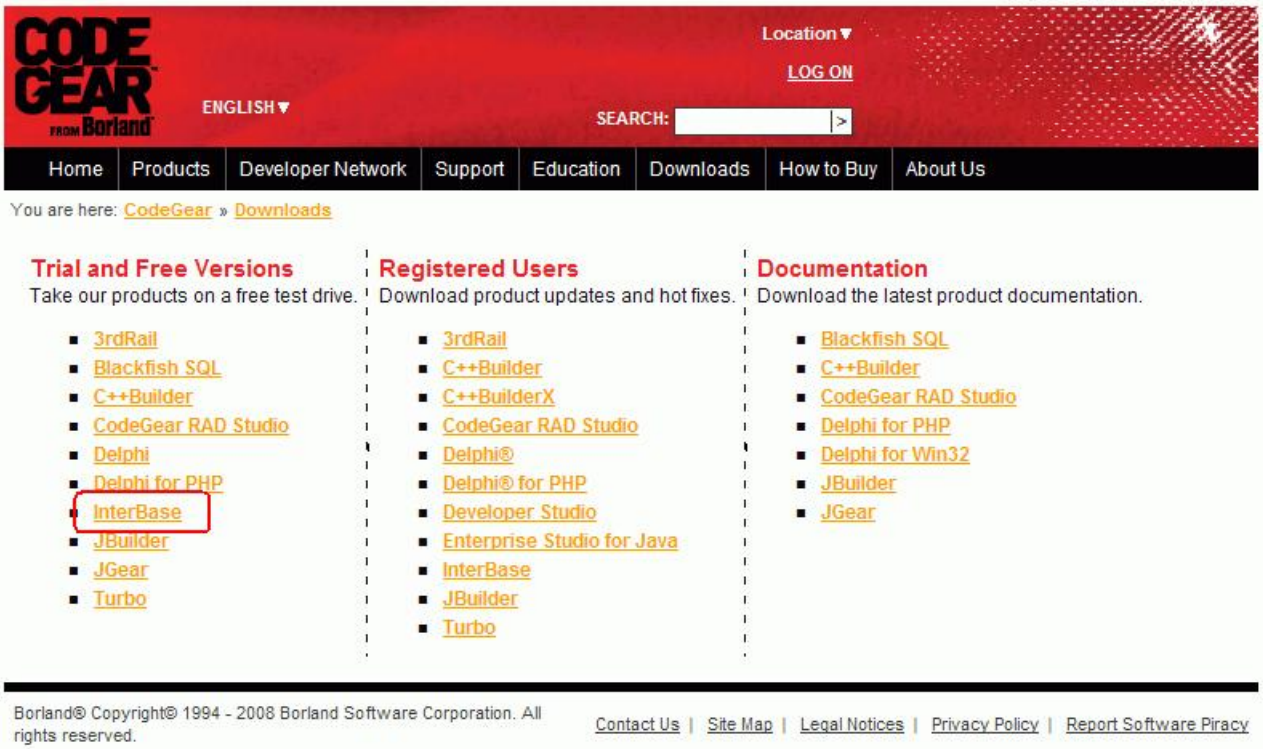
In an SMP (Symmetric Multi-Processing) system, this sets which processors can be used by the server. The value is taken from a bit map in which each bit represents a CPU. Thus, to use only the first processor, the value is 1. To use both CPU 1 and CPU 2, the value is 3. To use CPU 2 and CPU 3, the value is 6. The default value is 1. It doese make sense however to allow Firebird to use at least 2 CPUs, so that if the traffic on one of them gets halted due to, for example, a query going wrong, all other traffic can use the second CPU.

```
CpuAffinityMask = 1
```

Download and install InterBase®

This guide will lead you through the process of downloading and installing the free trial version of InterBase. For those having purchased InterBase®, the installation routine is the same (just skip the download instructions).

The current InterBase® trial version (at the time of writing this) was version 2007. It is a full InterBase server version and runs for 90 days. It can be downloaded free of charge from http://www.codegear.com/downloads.



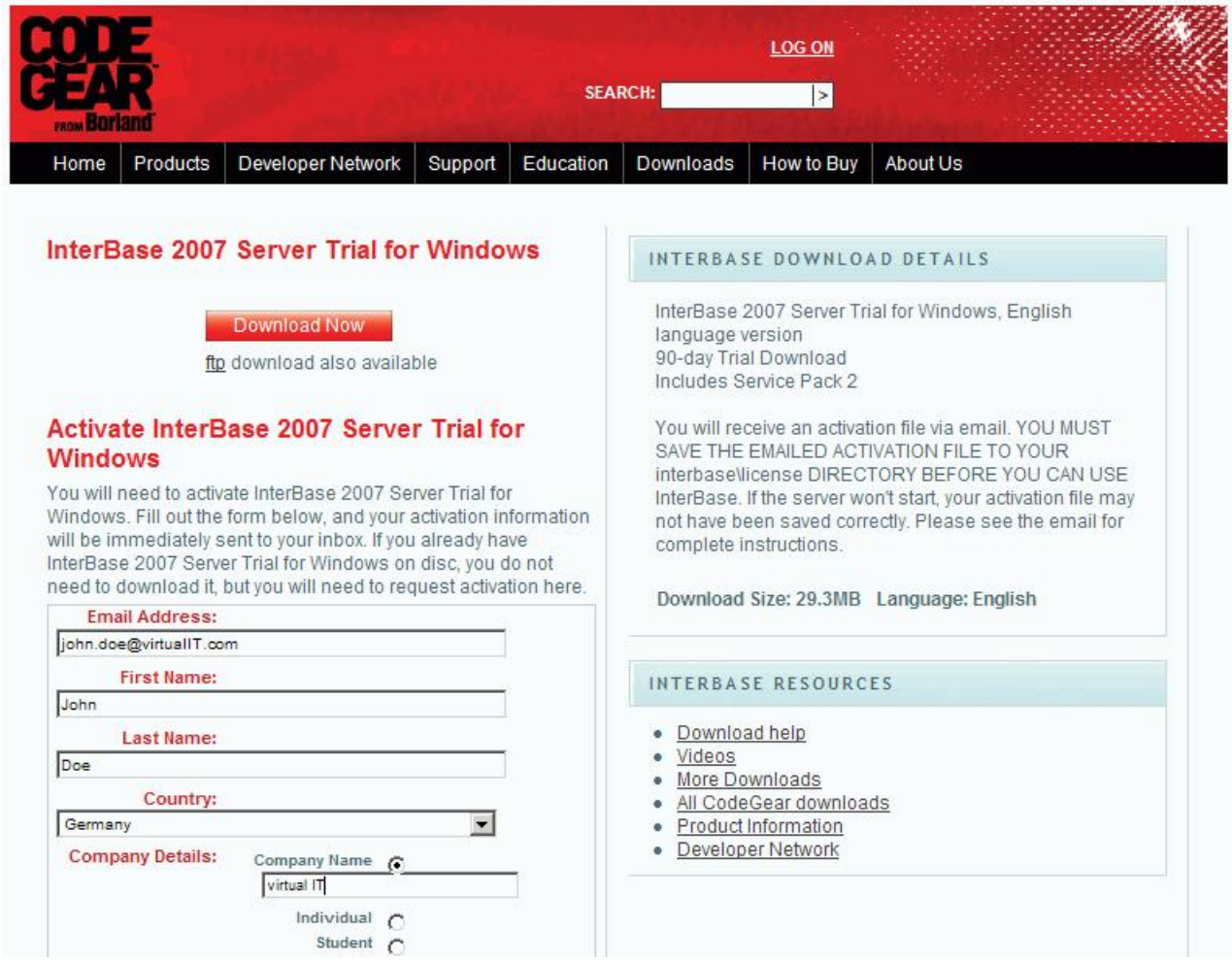
Click on *InterBase*, and then scroll down the list of Server versions and select the one you require.

InterBase 2007 Server Trial for Mac OS X InterBase 2007 Server Trial for Mac OS X, English language version 90-day Trial Download	17MB	English
InterBase 2007 Server Trial for Windows InterBase 2007 Server Trial for Windows, English language version 90-day Trial Download Includes Service Pack 2	29.3MB	English
InterBase 2007 Server Trial for Linux InterBase 2007 Server Trial for Windows, English language version 90-day Trial Download Includes Service Pack 2	65.3MB	English
InterBase 2007 Server Trial for Solaris InterBase 2007 Server Trial for Solaris, English language version 90-day Trial Download Includes Service Pack 2	70.2MB	English
InterBase 2007 Server Trial for Windows InterBase 2007 Server Trial for Windows, Japanese language version 90-day Trial Download	69.7MB	Japanese
InterBase 2007 Server Trial for Linux InterBase 2007 Server Trial for Linux, Japanese language version 90-day Trial Download	128MB	Japanese
InterBase 7.5.1 Server Trial for Windows InterBase 7.5.1 Server Trial for Windows, English language version 90-day Trial Download	34.5MB	English
InterBase 7.5.1 Server Trial for Linux InterBase 7.5.1 Server Trial for Linux, English language version 90-day Trial Download	70.6MB	English
InterBase 7.5.1 Server Trial for Solaris	76.3MB	English

Click the *Download* button and agree to comply with the *Export Controls*, to download the InterBase software to your hard drive.

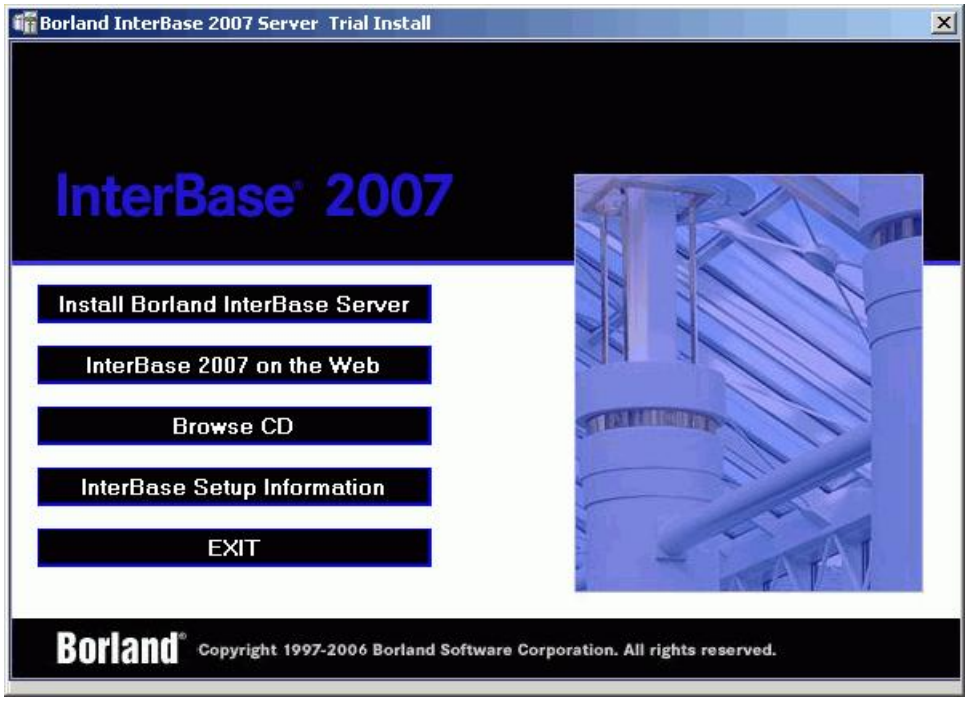
You will then need to enter your name, email and basic company information to receive your activitation certificate. You will need to activate InterBase 2007 Server Trial for Windows, otherwise it won't run. Fill out the online form and your activation information will be immediately mailed to your inbox. If you already have the InterBase 2007 Server Trial for Windows on disc, you do not need to download it, but you will still need to request activation here.

You must save the emailed activation file to your InterBase `/license` directory before you can use InterBase. If the server won't start, your activation file may not have been saved correctly. The email provides complete instructions.



Extract the downloaded `ZIP` file (for example in Windows to `C:Program FilesInterbase`) and start the relevant `install_[platform].exe` file.

To start the installation simply double-click the install executable.



For those installing InterBase for the first time, we recommend first clicking the *InterBase Setup Information* button (or open `IBSetup.html` in the installation package to open: *Installation, Registration, and Licensing Information for Borland® InterBase® 2007*.

The *Install Borland InterBase Server* button guides you through the installation: Check the software to be installed, and follow the prompts to accept the license agreement. Confirm whether you wish to use *Multi Instances*; if you do, change the *Instance Name* and *TCP Port* from the default values, `gds_db` and `3050`. Then confirm which options you wish to install, confirm the directory to be installed into or select a directory of your choice. After prompting a couple more times, InterBase is then installed.

The *Registration Wizard* then automatically starts for those who have purchased InterBase. Users of the Trial version should follow the instructions in the Product Registration email from CodeGear.

What is IBExpert?

Visit our product site for further details.

Test IBExpert for yourself - simply download the Trial Version (`setup_trial.exe`). These files are fully functional versions in the last stable build. They run for 45 days without any restrictions.

Alternatively purchase a full registered IBExpert version; again details can be found on our website.

Download and install IBExpert on Windows

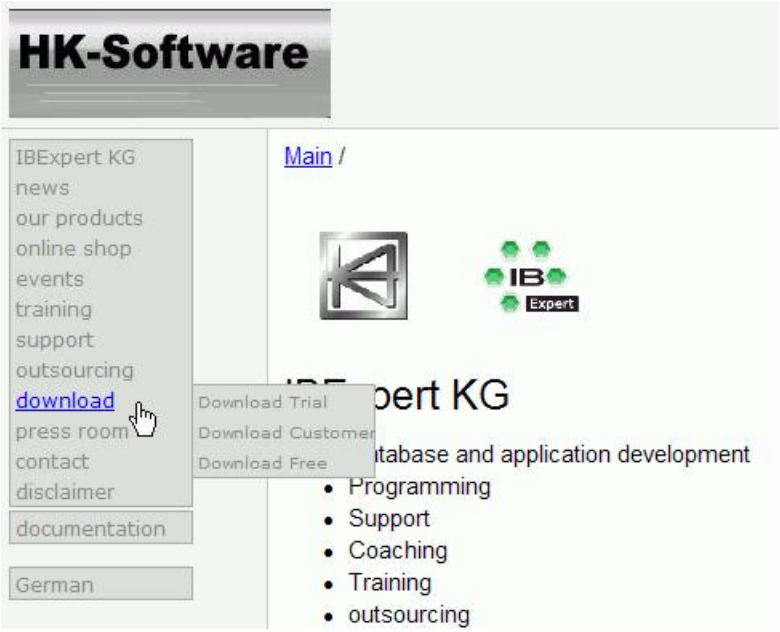
Customer Version

IBExpert can be downloaded from the IBExpert download pages. There are a number of versions - please refer to IBExpert licenses for further information.

If you are installing a new IBExpert version update (after December 2007) over an older IBExpert version (before December 2007) you will need to uninstall older versions first, as we have updated the IBExpert installer. You can do this simply and quickly by selecting all IBExpert products in the *Windows Control Center / Add or Remove Software*.

All registered databases are stored in the directory, `C:\Documents and Settings\\Applicationdata\HK-Software\IBExpert` or, if used, in the User Database. Please backup these files before uninstalling.

The download page on the IBExpert website offers a number of download options:



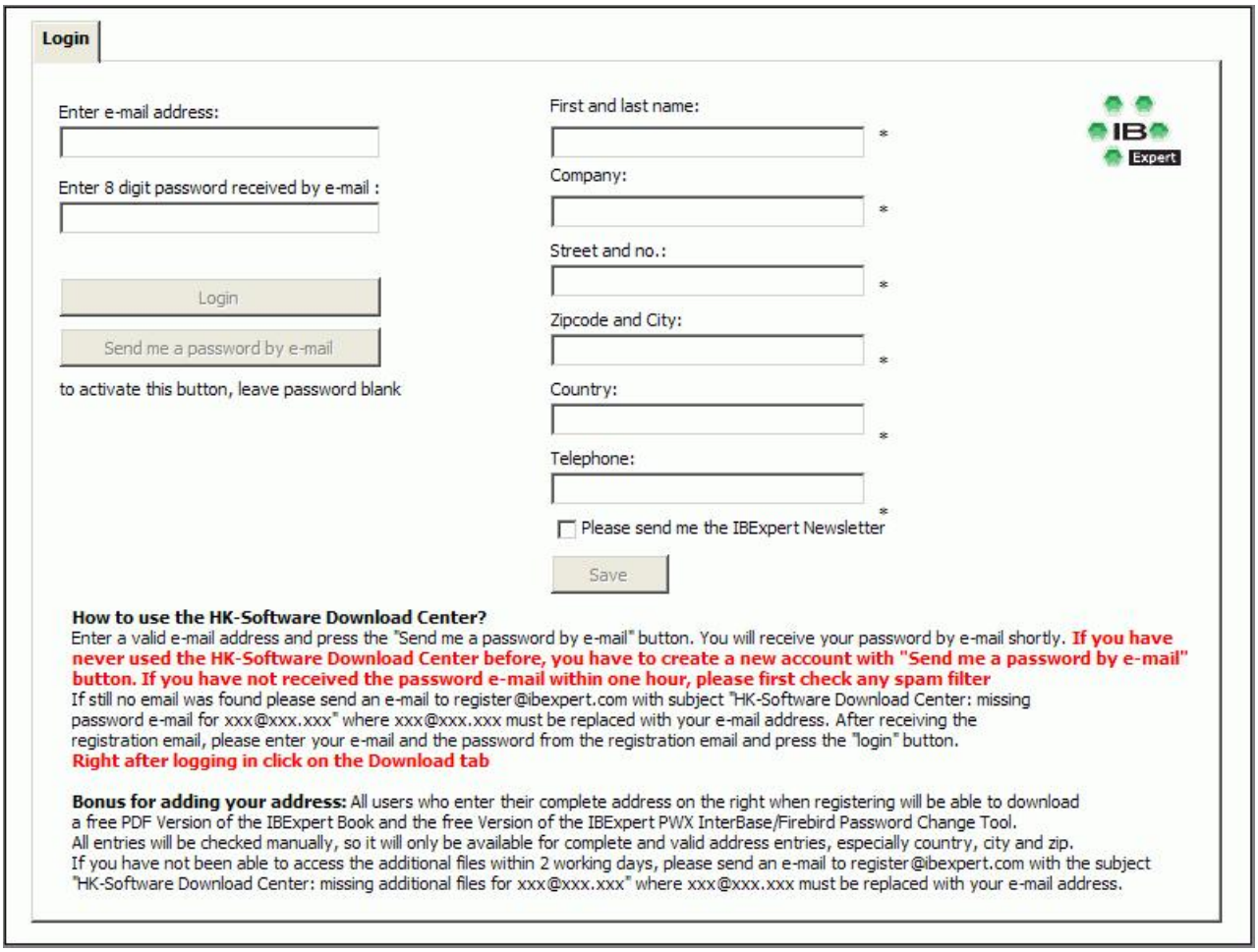
Registered customers should click on the Customer Download link. Enter your user name and the password supplied with the registration confirmation. The **Username** is a combination of key A and key B (for example `1234567887654321` when key A is `12345678` and key B is `87654321`). The **Password** is always `ibexpert`.

The current IBExpert version can be found by scrolling down to `setup_customer.exe`: these files include the unlimited use of the full version. These `setup_customer.exe` files comprise the full IBExpert Developer Studio versions, and replace the previous (before April 2006) executables.

For customers installing their first registered IBExpert customer version, you will be asked to register the product the first time you start the application. Please check that the computer name and company name which appears in the Registration window is the same as the computer name and company name quoted on your license form. Then simply enter Key A and Key B and click the *Register* button. You should receive a confirmation message stating that your IBExpert version has been successfully registered. Customers with site or VAR licenses need to copy the license file into the IBExpert directory before starting IBExpert for the first time, in order to avoid this key request.

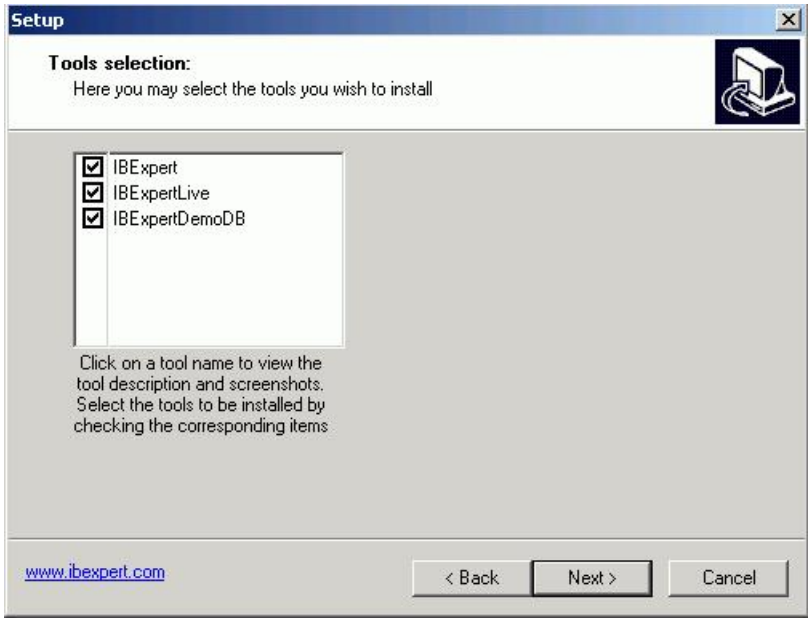
Personal Edition

Those wishing to download the free Personal Edition (for more information please refer to IBExpert Personal Edition), click on Download Free to register at the IBExpert Download Center:



Once you have registered you will be sent a password by e-mail which allows you access to the IBExpert Personal Edition download file. You simply need to login, click the *Download* tab to switch to the *Download* page, and select the file required.

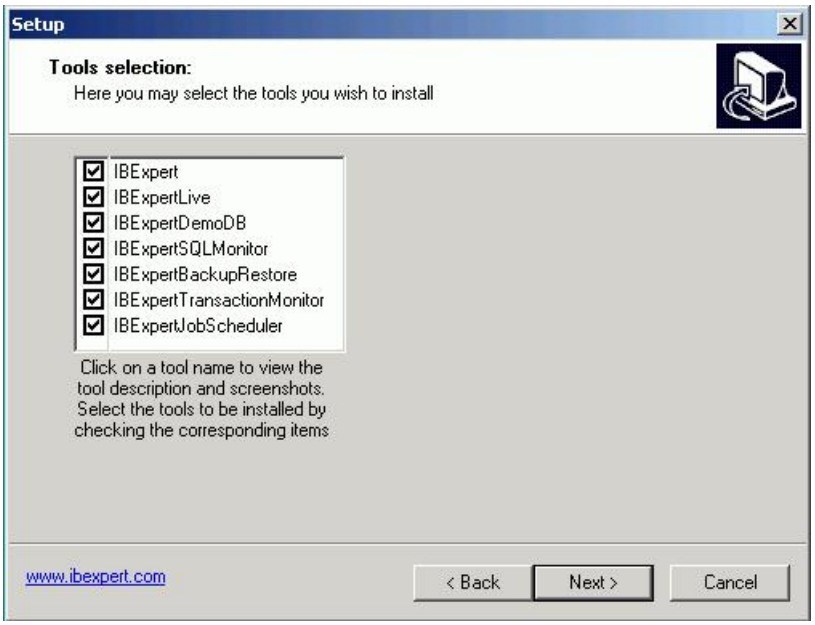
The Install Wizard offers those IBExpert Developer Studio Tools available in the Personal Edition:



Trial Version

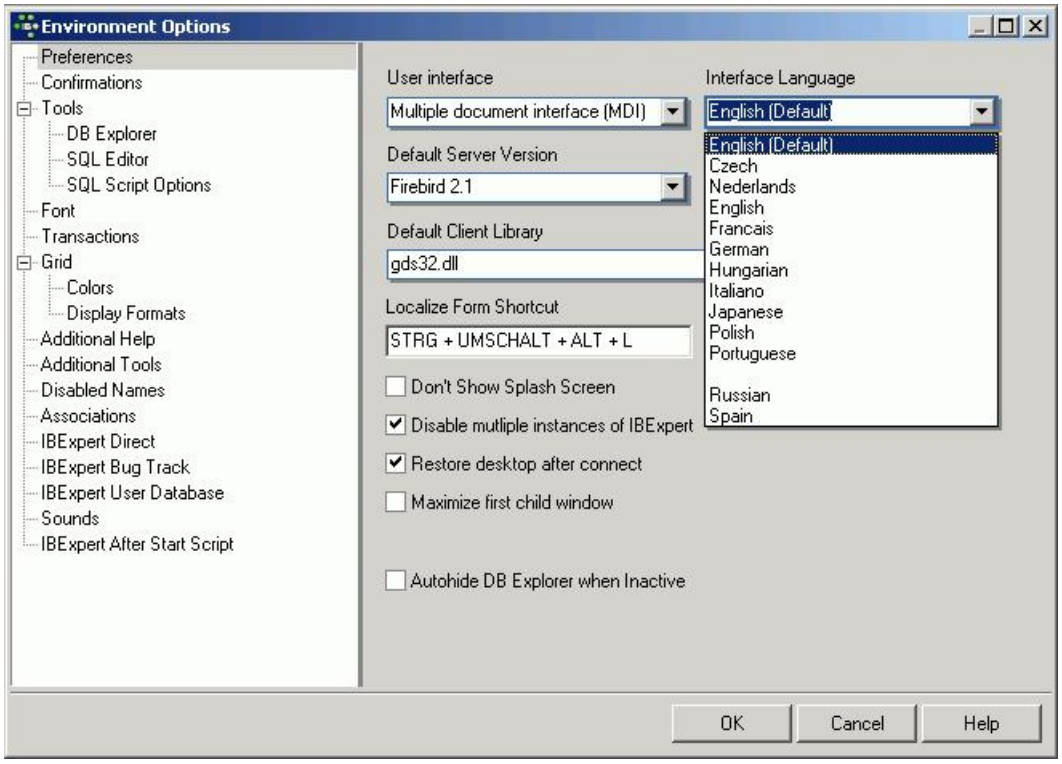
For those wishing to download the IBExpert Trial Version, go to Download Trial and click Download to download the `setup_trial.exe file`.

The IBExpert Customer and Trial versions both offer the full selection of all IBExpert Developer Studio Tools:



Following confirmation of the License Agreement and confirmation or alteration of the installation directory, IBExpert is automatically installed and started.

To alter the IBExpert interface language, use the IBExpert menu Options / Environment Options. Use the drop-down list found under *Interface Language* to select the language of your choice. This dialog also offers default options for the specification of the database version and client library.



Should you encounter any problems whilst attempting to download IBExpert, please send an e-mail (in either the English or German language) to register@ibexpert.biz, with a detailed error description.

To keep you informed of all new developments, we recommend you retain IBE Direct which is automatically activated in IBExpert. Further information regarding IBE Direct and adjusting the default settings can be found in the IBExpert Help Menu / IBExpert Direct.

We also recommend you subscribe to the IBExpert newsletter, which informs you of new developments and new versions (including documentation of all new features). Simply send a mail to news@ibexpert.com entering SUBSCRIBE in the subject heading.

Installing IBExpert on Linux

For tips and tricks regarding the installation of IBExpert on a Linux platform, please refer to our database technology article: *Using IBExpert and Delphi applications in a Linux environment, accessing Firebird*.

See also:
Select interface language

IBExpert Personal Edition

The IBExpert Personal Edition is a free version, offering new users the chance to get acquainted with IBExpert at their own pace. It is however somewhat limited in its functionality, and does not include the following features:

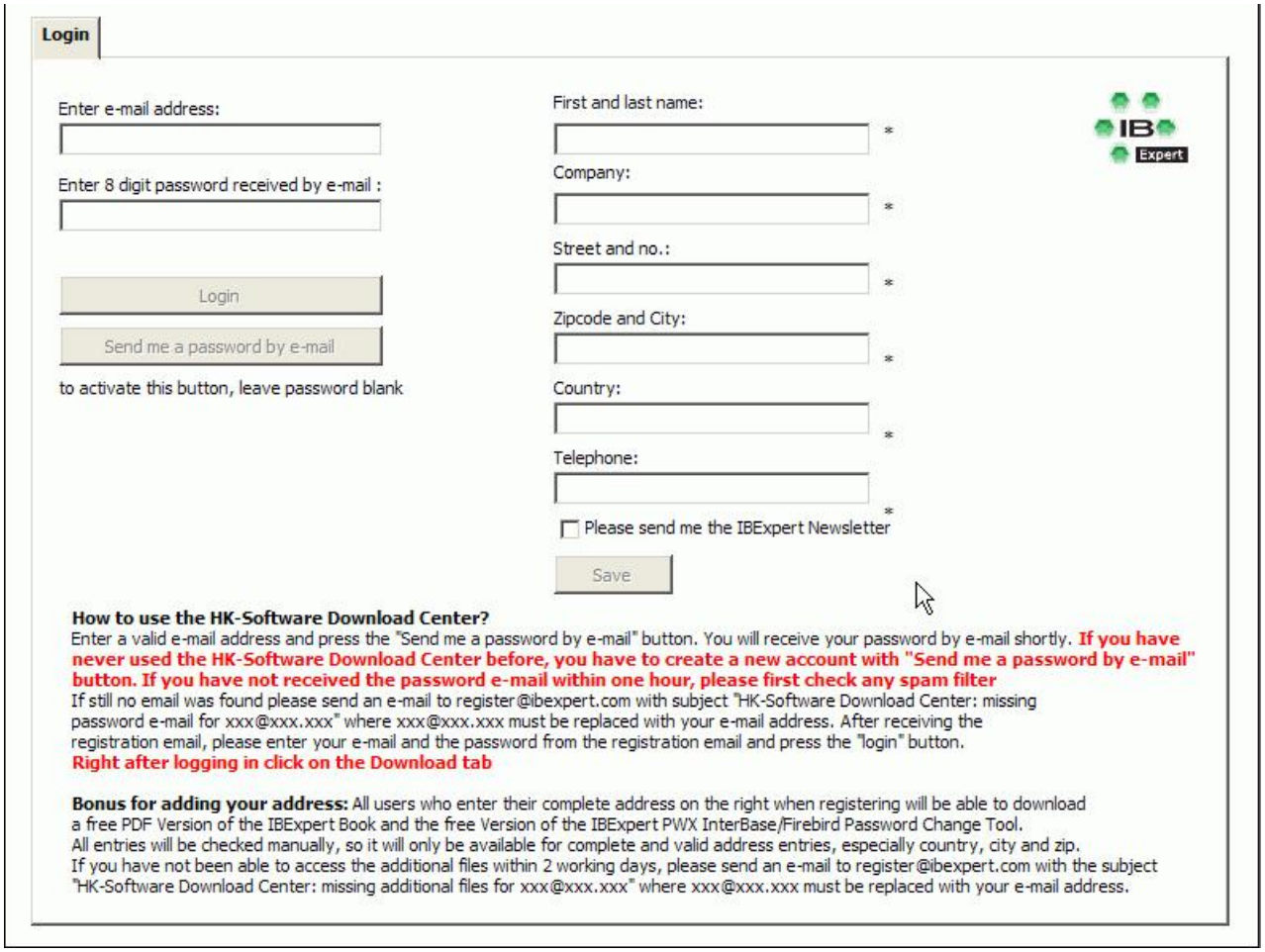
- Data Analysis
- Database Designer
- SP/Triggers Debugger
- Visual Query Builder
- Report Manager
- Test Data Generator
- Blob Editor
- Grant Manager
- SP/Triggers/Views Analyzer
- Database Comparer
- Log Manager
- Table Data Comparer
- IBEScript and IBEBlock
- some other features such as autogranting privileges, recomputing selectivity of all indices etc.

These features can be viewed and tested in the IBExpert Trial Version.

IBExpert version 2006.06.18 introduced a new URL to download the IBExpert Personal Edition:

IBExpert Download Center: http://www.ibexpert.com/downloadcenter

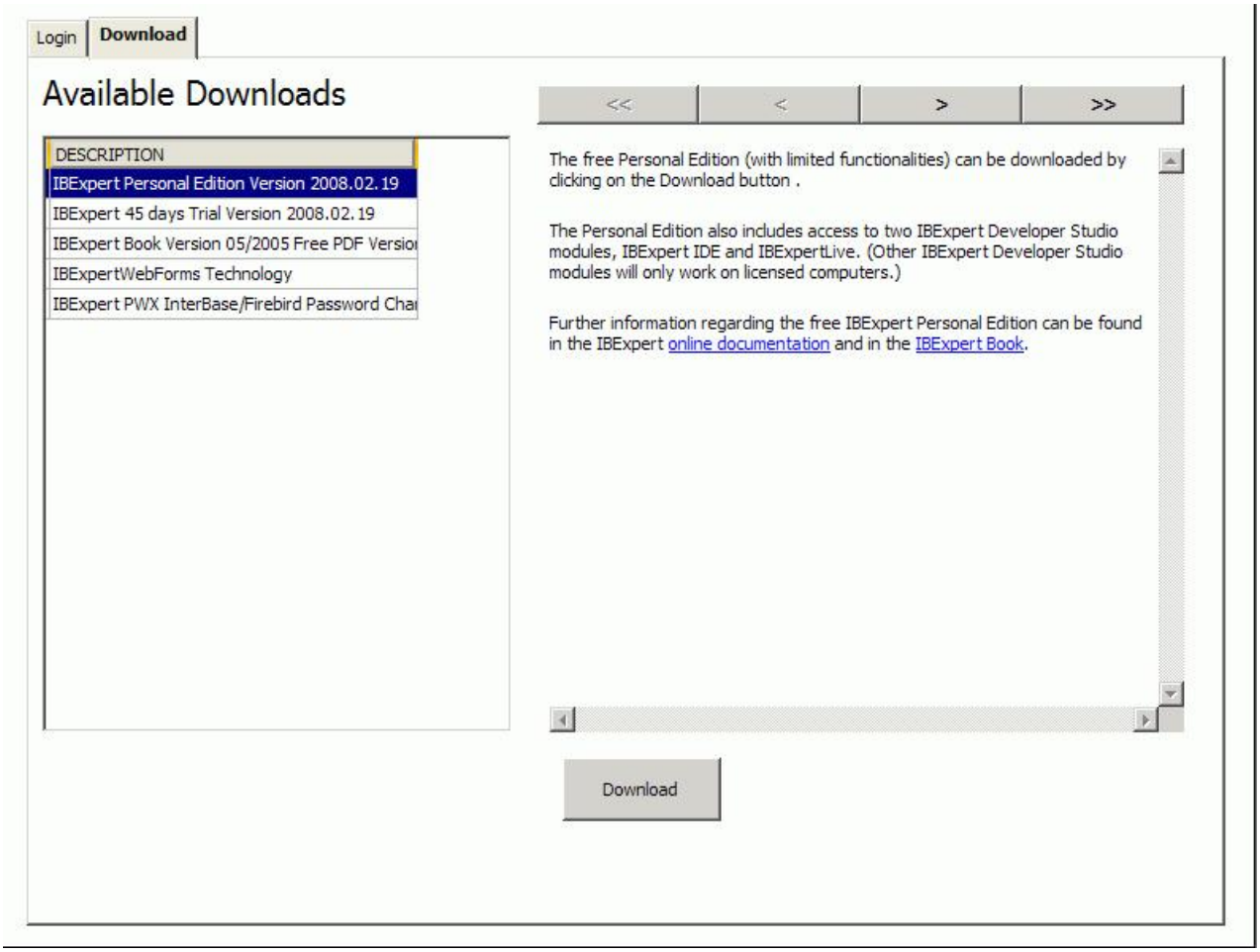
You will need to enter a valid e-mail address to receive a personal password, allowing you access to the the IBExpert Download Center:



Simply follow the directions for new and existing users, as detailed in the dialog.

Please note that if you enter your full address on the right-hand side of the registration dialog, you can attain access to the download PDF of the IBExpert Book - Tools for Database Developers, with over 600 pages of documentation about IBExpert, Firebird and InterBase.

Once you have received your password you can login into the IBExpert Download Center and download either the IBExpert Personal Edition, the IBExpert Book PDF file or information regarding the new IBExpertWebForms:



The IBExpert Download Center is the first real life application created with IBExpertWebForms, a new technology which was introduced as a full Trial Version in IBExpert version 2007.06.05.

Further information regarding the free IBExpert Personal Edition can be found in the IBExpert online documentation and in the IBExpert Book.

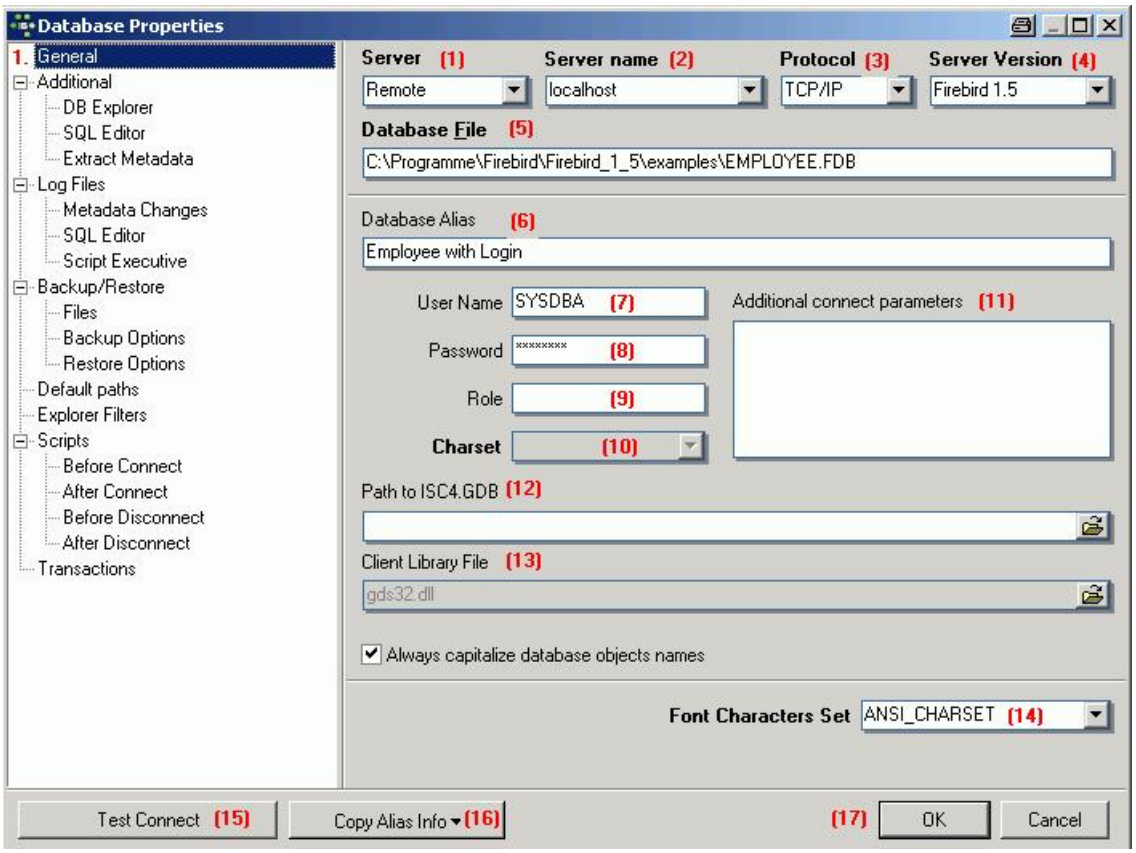
Registering a database (using the EMPLOYEE) example

In order to administrate a database using IBExpert, it is first necessary to register the database. For detailed information regarding database registration, please refer to Register Database.

Here we will briefly show how to register a database, based on the sample `EMPLOYEE` database supplied with both Firebird and InterBase.

First open the Register Database dialog, using the IBExpert menu item Database / Register Database, right-clicking in the Database Explorer (left-hand panel) and selecting the Register Database menu item, or using the key shortcut [Shift + Alt + R].

The Register Database dialog appears:



(1) Server: first the server storing the database needs to be specified. This can be local (localhost) or remote (see Create Database). By specifying a local server, fields (2) and (3) are automatically blended out, as they are in this case irrelevant.

(2) Server name: must be known when accessing remotely. The standard port for InterBase and Firebird is 3050. However this is sometimes altered for obvious reasons of security, or when other databases are already using this port. If a different port is to be used for the InterBase/Firebird connection, the port number needs to be included as part of the server name (parameter is server/port). For example, if port number 3055 is to be used, the server name is `SERVER/3055`. This is sometimes the case when a Firewall or a proxy server is used, or when another program uses the standard port. For using an alias path for a remote connection, please refer to the article remote database connect using an alias.

(3) Protocol: a pull-down list of three options: TCP/IP, NetBEUI or SPX. TCP/IP is the worldwide standard (please refer to Register Database for more information).

(4) Server versions: this enables a server version to be specified as standard/default from the pull-down list of options. This is necessary for various internal lists. For example, possible key words can be limited this way.

(5) Database File: by clicking on the folder icon to the right of this field, the path can easily be found and specified and the database name and physical path entered. For example for Firebird:

```
C:ProgramsFirebirdFirebird_1_5examplesEMPLOYEE.FDB
```

for InterBase:

```
C:ProgramsInterbaseexamplesEMPLOYEE.GDB
```

If no database alias has been specified, the database name must always be specified with the drive and path. Please note that the database file for a Windows server must be on a physical drive on the server, because InterBase/Firebird does not support databases on mapped drive letters.

(6) Database Alias: descriptive name for the database (does not have to conform to any norms, but is rather a logical name). The actual database name and server path and drive information are hidden behind this simple alias name - aiding security, as users only need to be informed of the alias name and not the real location of the database. For example:

```
Employee
```

(7) User Name: the database owner (i.e. the creator of the database) or SYSDBA.

(8) Password: if this field is left empty, the password needs to be entered each time the database is opened. Please refer to Database Login for further information. The default password for SYSDBA is masterkey. Although this may be used to create and register a database, it is recommended - for security reasons - that this password be changed at the earliest opportunity.

(9) Role: an alternative to (7) and (8);can initially be left empty.

(10) Charset (abbreviation for Character Set): The default character set can be altered and specified as wished. This is useful when the database is designed to be used for foreign languages, as this character set is applicable for all areas of the database unless overridden by the domain or field definition. If not specified, the parameter defaults to NONE (the default character set of EMPLOYEE.FDB), i.e. values are stored exactly as typed. For more information regarding this subject, please refer to Charset/Default Character Set. If a character set was not defined when creating the database, it should not be used here.

(11) Additional connect parameters: input field for additional specifications. For example, system objects such as system tables and system-generated domains and triggers can be specified here. They will then automatically be loaded into the Database Explorer when opening the database alias.

(12) Path to ISC4.GDB: This can be found in the InterBase or Firebird main directory. This database holds a list of all registered users with their encrypted passwords, who are allowed to access this SERVER. When creating new users in earlier InterBase versions (<6), IBExpert needs to be told where the ISC4.GDB can be found. Since InterBase version 6 or Firebird 1 there is a services API. So those working with newer versions may ignore this field!

(13) Always capitalize database objects' names (checkbox): this is important as in SQL Dialect 3 entries can be written in upper or lower case (conforming to the SQL 92 standard). InterBase however accepts such words as written in lower case, but does not recognize them when written in upper case. It is therefore recommended this always be activated.

(14) Font character set: this is only for the IBExpert interface display. It depends on the Windows language. If an ANSI-compatible language is being used, then the ANSI_CHARSET should be specified.

(15) Test connect: the Comdiag dialog appears with a message stating that everything works fine, or an error message - please refer to the IBExpert Services menu item, Communication Diagnostics for more details.

(16) Copy Alias Info: alias information from other existing registered databases can be used here as a basis for the current database. Simply click on the button and select the registered database which is to be used as the alias.

(17) Register or Cancel: after working through these options, the database can be registered or cancelled.

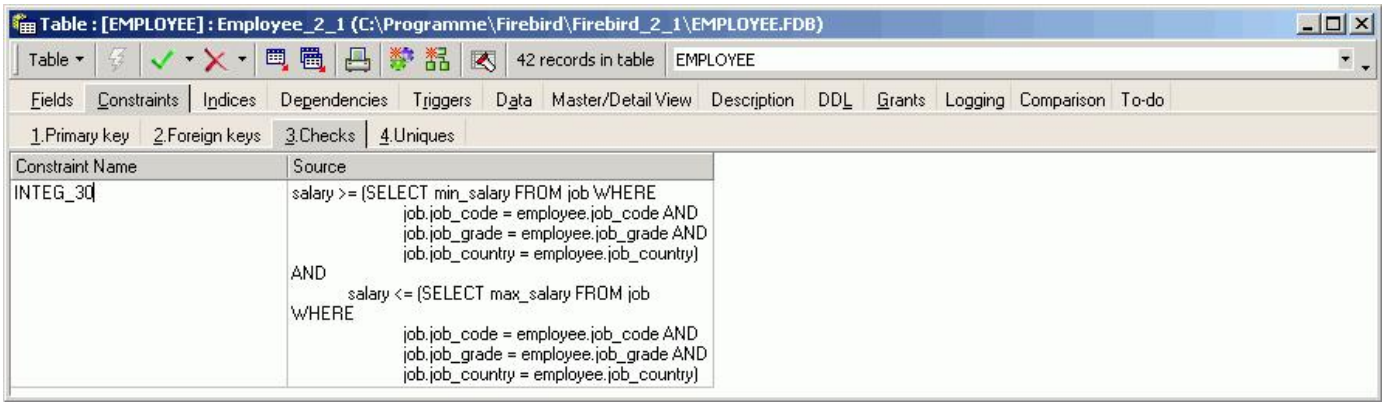
Details of further options (listed in the left-hand panel in the Register Database dialog) may be found under Register Database (individual subjects are listed in the upper gray panel in the online documentation). These are not compulsory, and may be altered at a later date, if wished, using the Database / Database Registration Info menu item.

Following successful registration of EMPLOYEE database, it will appear in the on the left-hand side. Simply double-click on the database name to connect to it.

Working with a database

A registered database can be connected simply by double-clicking on the database name in the DB Explorer.

Alternatively use the IBExpert menu item Database / Connect to Database, click the *Connect Database* icon in the toolbar, or use the key shortcut [Shift + Ctrl + C]. The database and its objects appear in a tree form in the DB Explorer:



For information with regard to the details displayed in the DB Explorer, please refer to Register Database / Additional and the IBExpert Options menu, Environment Options / Tools for alternatives regarding the DB Explorer.

The individual database objects may be opened by double-clicking on the object name. For further information about the individual objects, please refer to Database Objects.

For further information regarding IBExpert navigation, please refer to IBExpert Screen. Options and templates may be specified and adapted using the IBExpert Options menu. Other important IBExpert features can be found in the IBExpert Tools menu and IBExpert Services menu.

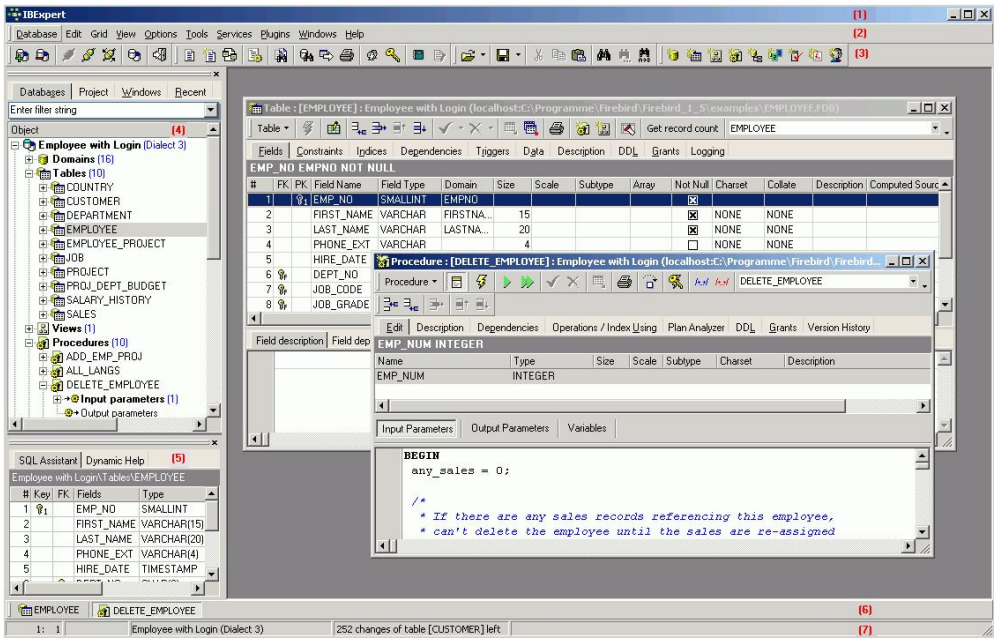
The IBExpert online documentation provides not only a comprehensive documentation for using IBExpert, but also offers many tips for those new to database development. The online documentation can be viewed under http://ibexpert.net/ibe/pmwiki.php?n=Doc.IBExpert or alternatively individual subjects may be viewed context-sensitively, using the [F1] key from any IBExpert dialog or the DB Explorer. The documentation includes a search function and a *Recent Changes* function. Or you can download the complete documentation files onto your hard drive.

And if you can't find an answer to your problem there, please mail us at documentation@ibexpert.com.

See also:
Database Objects
IBExpert Screen
SQL Editor
IBExpert Help menu

IBExpert screen

When IBExpert is started, the standard IBExpert screen appears as follows:



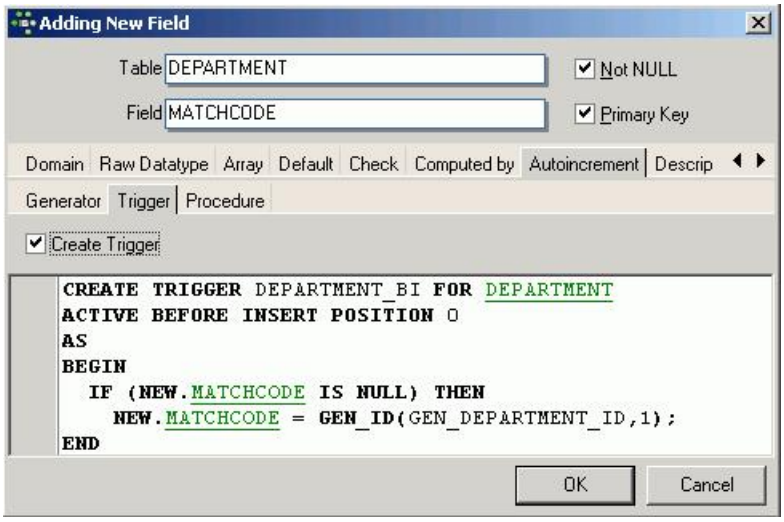
The standard IBExpert settings display a large working window, with the menu **(2)** and toolbars **(3)** at the top of the screen, a windows bar **(6)** and status bar **(7)** at the bottom, and the DB Explorer **(4)** on the left, divided from the SQL Assistant (lower left) **(5)** by a splitter.

The IBExpert View menu can be used to blend the DB Explorer, status bar, windows bar and toolbars in or out.

Further visual options can be specified by the user in the IBExpert Options menu.

IBExpert Splash screen

The IBExpert splash screen appears when IBExpert is started. It displays the IBExpert logo and version number.



The splash screen may be disabled if wished, by checking the *Don't Show Splash Screen* option, found under Options / Environment Options on the initial Preferences page.

(1) Title bar

The title bar is the blue horizontal bar at the top of the main IBExpert screen, and at the top of all IBExpert editors. It displays the program or editor name on the left, and in the right hand corner there are four small icons (from left to right):

1. Print (only on the IBExpert screen with the MDI Interface; with the SDI Interface it appears on the active window/editor)
2. Minimize IBExpert / Editor window
3. Maximize IBExpert / Editor window
4. Exit IBExpert / Exit Editor

(2) Menu

The IBExpert menu bar can be found at the top of the screen:

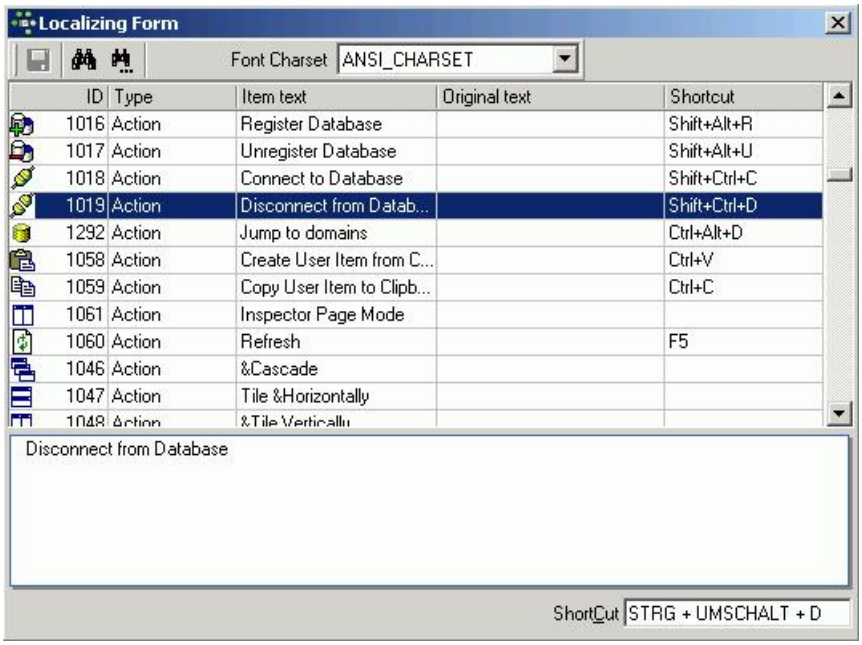
The individual menu headings conceal drop-down lists, opened simply by clicking on one of the words with the mouse or by using [Alt + {underlined letter}], e.g. the Database menu can be started by clicking with the mouse on the word database, or by using the key combination [Alt + D].

The most frequently-used menu items can also be found in the toolbars, represented as icons, or using the right mouse button in either the DB Explorer or the main editors. Alternatively keyboard shortcuts can also be used.

Keyboard Shortcuts (Localizing Form)

Many menu items can also be executed using so-called keyboard shortcuts (a combination of keys). Where available, these are listed to the right of the menu item name in the menus, and when the cursor is placed over a toolbar icon.

[Ctrl + Shift + Alt + L] works in almost all IBExpert forms and calls the Localizing Form, where you can refer to a complete list of all available shortcuts relevant to the active dialog. It is possible to specify your own shortcut for opening the Localizing Form in the IBExpert Options menu item, Environment Options, under *Localize form shortcut*.



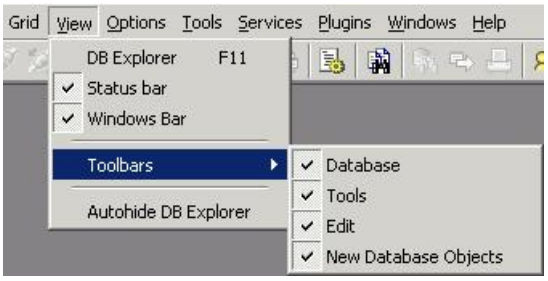
(3) Toolbars

The toolbar is a row of symbols (called icons), representing different menu items. By clicking on an icon with the mouse, a pre-defined menu item is executed. This shortcut is ideal for those operations performed often, as they save the necessity of repeatedly searching through the main menus.

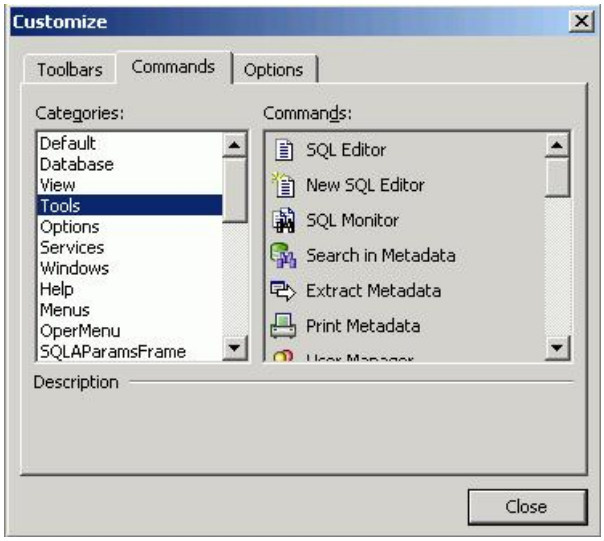
Toolbars can be found in IBExpert in the main window and in the main editors. As with most Windows applications the toolbars are positioned as standard in a horizontal row directly below the main menu in the upper part of the window, or in the upper part of the dialogs. They can however be positioned as wished within the window (main or dialog) using drag 'n' drop.

When the cursor is placed over an icon the respective menu command and keyboard shortcut are displayed.

The user can specify which toolbars he wishes to be displayed in the main IBExpert window using the menu item View / Toolbars.



The individual icons can be specified using the *Customize...* menu item, opened by holding the mouse over the toolbar and right-clicking.

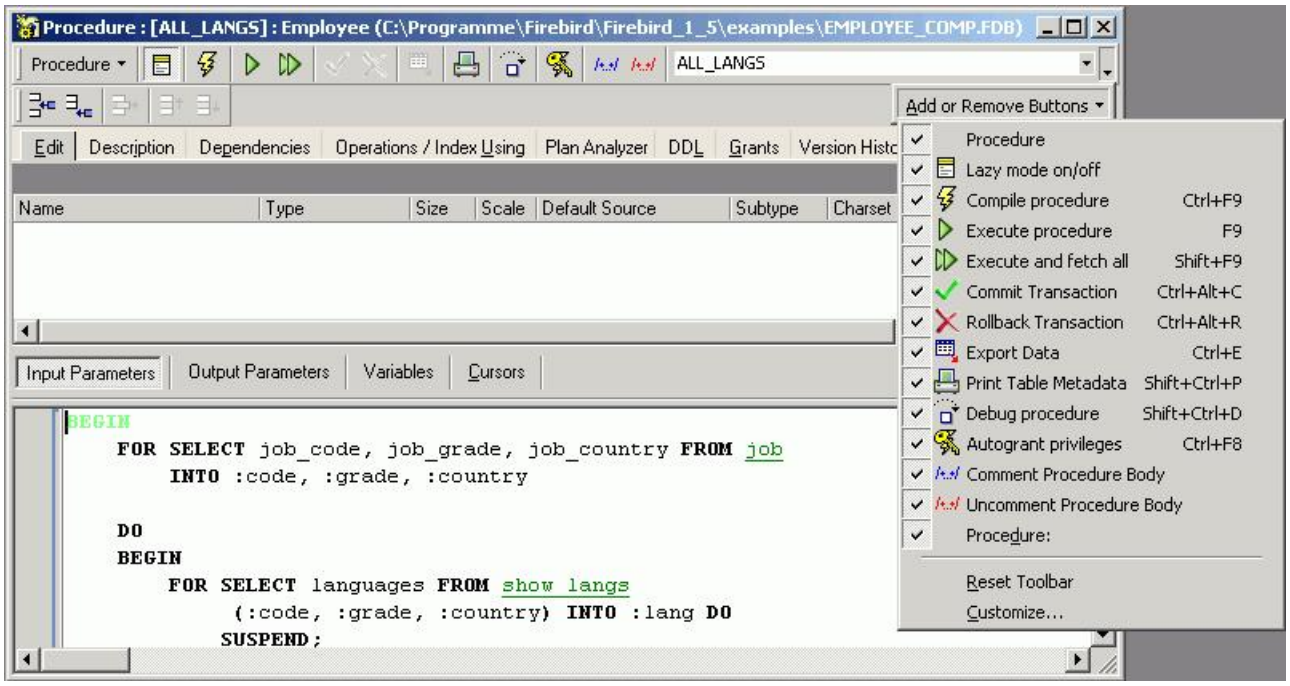


The *Customize Tools* page displays a list of the toolbar options available. User-defined toolbars can be created here if wished, or reset to the original IBExpert toolbar.

The *Command* page enables the different menu options listed under *Categories* to be selected, and the icons (in the right-hand list) added or removed to toolbars using drag 'n' drop.

The *Options* page allows certain menu and icon options to be checked if wished.

The *Editor* toolbars can be customized by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list, or using the above method by selecting the last menu item *Customize...*



Should you ever experience problems with any of the toolbars in IBExpert, simply delete `IBExpert.tb`, found in `Documents and Settings<user>Application Data\HK-Software\IBExpert` and then restart IBExpert.

The individual IBExpert toolbars are listed in more details in the Addenda.

Icons

Icons are a principal feature of graphical user interfaces. An icon is a small, square graphical symbol.

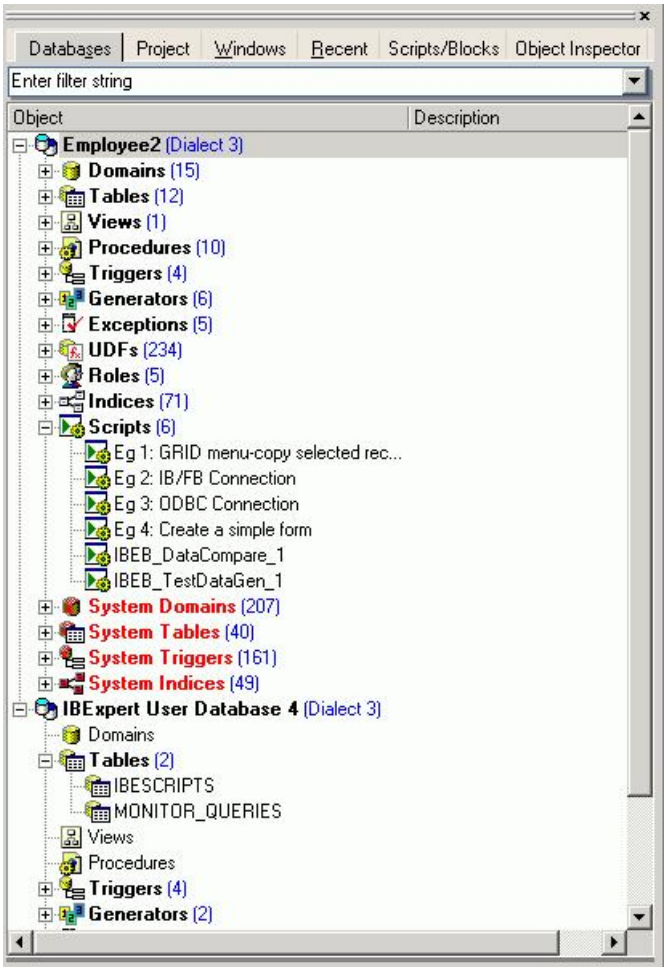
Each icon represents a menu item, the description of which appears, when the mouse is held over it. Icons can be used as shortcuts by those users who work mainly with a mouse (as opposed to the keyboard).

Icons are usually grouped together in a toolbar, which offers a series of symbols all relating to a certain subject, e.g. new database object, grants etc.

(4) Database Explorer

The IBExpert Database Explorer is a navigator which considerably simplifies the work with InterBase/Firebird databases? and the database objects.

The *Database Folder* displays all registered databases at a glance. A database connection can be made simply by double-clicking on the database name.



Each connected database is displayed in a logical tree form, including a list of all the database objects created in this database. If the database contains objects of some of these types, the name of the respective object branch appears in bold. The blue number in brackets behind the object caption shows the number of objects already created for this database.

Detailed information regarding the highlighted database object can be viewed in the SQL Assistant (below the DB Explorer).

The object tree branches can be expanded or reduced by double-clicking the object heading or clicking the"+" or the "-" sign to the left of these headings (alternatively use the "+" and "-" keys to open a highlighted object heading). The individual objects themselves can be opened with a double-click or by pressing the [Enter] key.

The object description can be seen to the right of the object name, provided a description was inserted at the time of creation, and providing the DB Explorer is opened wide enough (the width of the DB Explorer can be expanded or reduced by dragging the right-hand splitter (i.e. the divider between the DB Explorer and the Main Window) with the mouse).

Should you experience any problems with double-click expanding, or your object descriptions are not displayed at all, please check the IBExpert Options menu item Environment Options under the branch, DB Explorer, to ensure that these options have been checked. It is also possible to specify color display here for system objects, the *Database Folder* and inactive triggers.

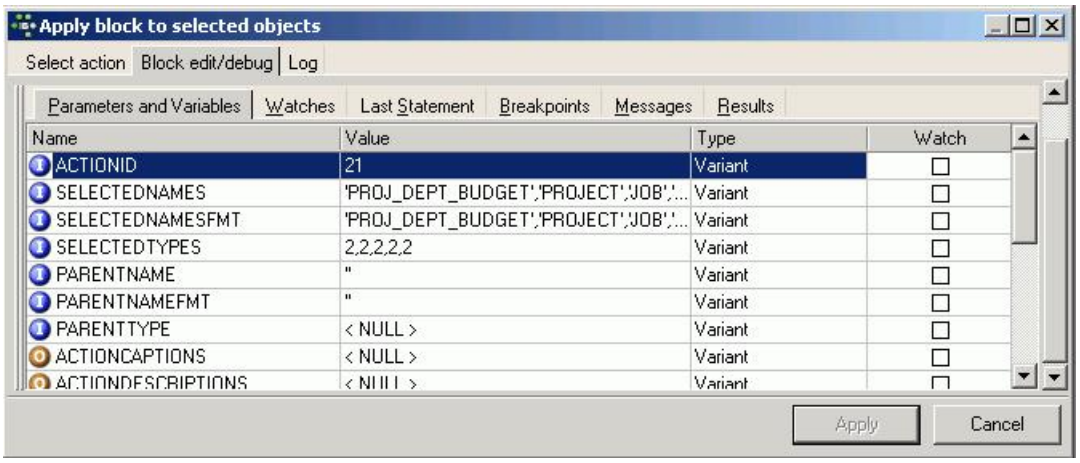
When a database, the object captions or the objects themselves are highlighted, the DB Explorer menu can be opened by right-clicking the mouse. The contents of the Database Explorer can be refreshed using [F5].

Using the control panel and right mouse button many basic metadata and data operations can be performed directly from the DB Explorer, such as creating, editing and dropping a database and its objects. Since IBExpert version 2006.08.12 it is also possible to unregister more than one database at the same time. In IBExpert version 2004.12.12.1 the option to activate/deactivate only selected procedures/triggers was added. Just select the required SP/triggers holding the [Ctrl] or [Shift] keys and choose the *Deactivate/Activate* item in the DB Explorer context menu. A further option was added in IBExpert version 2005.06.07 to sort database/folder nodes in ascending or descending order. And IBExpert version 2006.12.11 introduced the possibility to autogrant privileges for several selected objects at a time.

In IBExpert version 2004.9.12.1 a separate node was added for database indices. It is also possible to display system indices (indices for system tables). Use the IBExpert menu item Database Registration Info / DB Explorer / Additional / Show System Indices to enable/disable the display of system indices.

In IBExpert version 2004.12.12.1 support for InterBase 7.5 embedded user authentication was added. There is now a separate node for embedded users in the Database Explorer. It is possible to create, alter and delete embedded users using the DB Explorer context menu.

IBExpert version 2008.02.19 introduced the new menu item, *Apply IBEBlock to selected object(s)*. This feature is based on the IBEBlock functionality and allows you to create your own set of code blocks to process selected object(s). Inplace debugging is available.

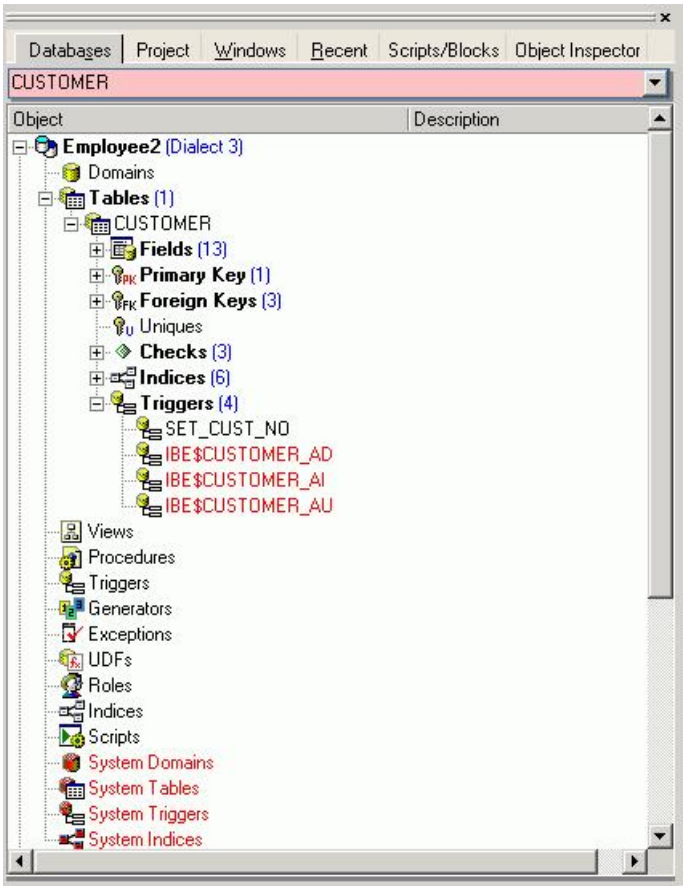


Since IBExpert version 2006.01.29 it is possible to execute Firebird 2.0 blocks, IBEBlocks and IBEScripts stored in registered databases or in the User Database from the DB Explorer, by using the relevant context-senstive menu item, when a script is highlighted, or by opening the script (double-click to open the Block Editor) and executing with [F9].

The text input field at the top of the DB Explorer (directly underneath the tabs) can be used to filter object names, e.g. to search for an object, `EMP`, simply type `EMP`. If `EMP*` or `EMP%` is typed, IBExpert displays all objects beginning with `EMP`; for an object ending in `EMP`, type `*EMP` or `%EMP`. To display objects which have a

substring in their name, it is necessary to type `*EMP*` or . It is also possible to use `?` For example, to display objects whose names start with `EMP` and are exactly 6 symbols in length. In this case type `EMP???`. Regular expressions are, of course, also allowed.

Please note that this option does not, however, search for individual fields - if this is required, use the IBExpert Tools menu item, Search in Metadata.



Certain display default filters can also be defined, under Register Database / Explorer Filters. And under Database Registration Info or Register Database, system tables?, system generated domains and triggers and object details (fields, triggers etc. relating to a specific object) can be displayed or blended out as wished, by clicking on the Additional / DB Explorer branches.

The DB Explorer includes the following tabs:

- Database Folder (described above)
- Project View
- Diagrams (only visible when the Database Designer is in use)
- Windows Manager
- Recent List
- Scripts/Blocks (new to IBExpert version 2005.12.04)
- Inspector Page Mode

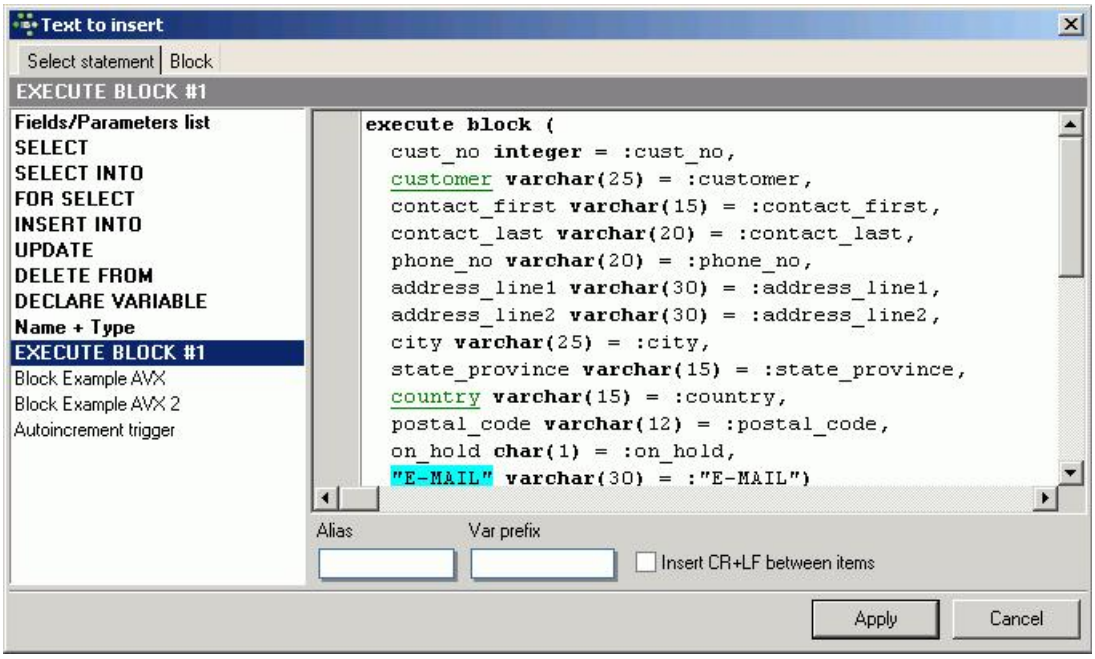
[F11] blends the DB Explorer in and out. And please also refer to the IBExpert Menu item View / Autohide DB Explorer. This option namely enables the DB Explorer to disappear automatically when any editor is opened - allowing a larger working area. It is blended back into view simply by holding the mouse over the left-hand side of the IBExpert main window.

Drag 'n' Dropping Objects into Code Editors

Objects may be dragged 'n' dropped from the DB Explorer and SQL Assistant into many of the IBExpert Tools and Services code editor windows, for example, the SQL Editor and Query Builder. Since version 2004.2.26.1 this has been greatly improved: when an object node(s) is dragged from the DB Explorer or SQL Assistant, IBExpert will offer various relevant versions of text to be inserted into the code editor. And since IBExpert version 2006.03.06 the charcase of keywords and identifiers specified under Options / Editor Options / Code Insight is taken into account.

Since IBExpert version 2004.8.5.1 it is possible to store server info (server type, server name, server version, connection protocol) and client library name for database folders.

IBExpert version 2008.02.19 introduced the possibility to create your own sets of statements that will be composed when you drag-n-drop object(s) from the Database Explorer into any code editor. This feature is based on IBEBlock.



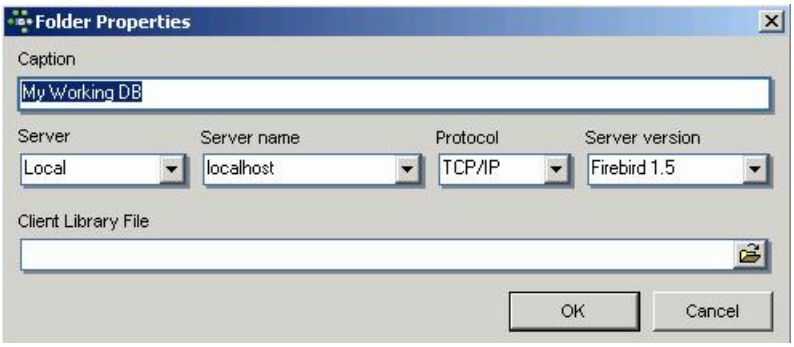
Database Folder

The DB Explorer *Database Folder* can be used to specify a selection of databases as wished, so that it is not necessary to search through all available databases each time a specific database is required. The database folder allows a hierarchical classification of the Database Registration. This is for example useful for system vendors with many customers and databases, and simplifies, for example, the logging in to customer databases via a router.

When a database is registered, it is automatically displayed here in the folder list. Connected databases are displayed in bold, disconnected in normal type. *Please note:* it is possible to blend out all unconnected databases using the DB Explorer right-click menu item, *Hide Disconnected Databases*.

A new database folder can be created in the DB Explorer by highlighting the connected database for which a folder is to be created, right-clicking and selecting *New Database Folder ...* (or [Ctrl + N]).

It is then possible to rename the database folder, by selecting the folder and using the right-click context-sensitive menu or [Ctrl + O]:

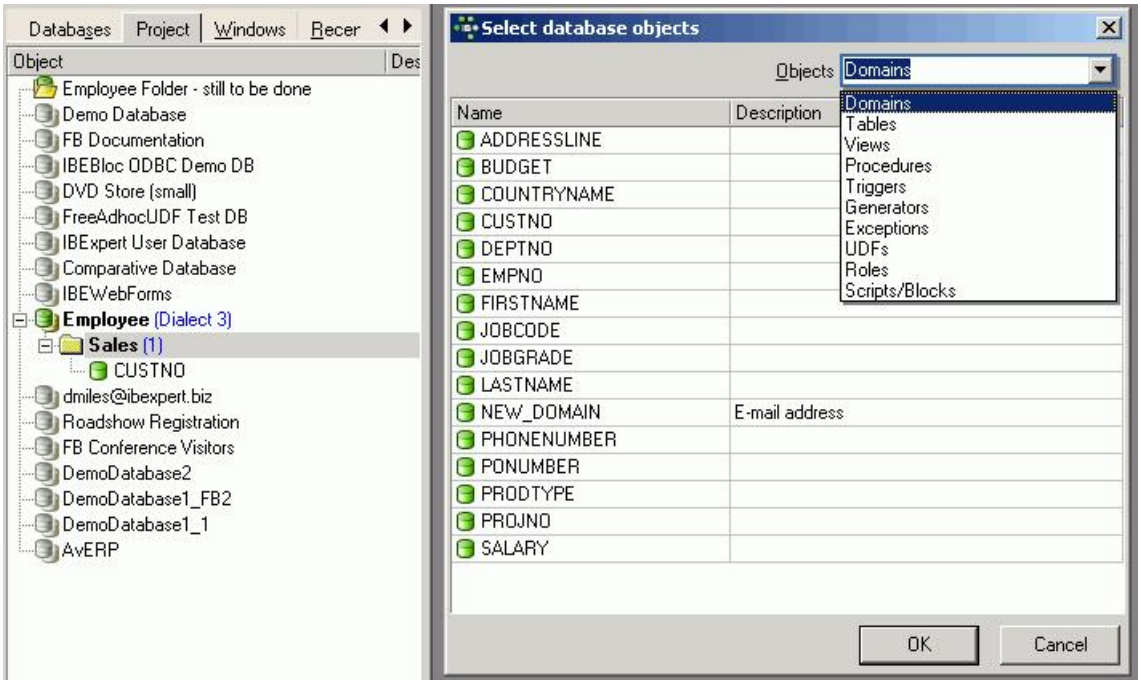


Since IBExpert version 2004.8.5.1 it is also possible to store server information (server type, server name, server version, connection protocol) and client library name for database folders.

A folder can also be deleted (again, using the right-click menu or [Ctrl + Del]). Please be careful when using this delete command, as IBExpert does not ask for confirmation before deleting the folder!

Project View

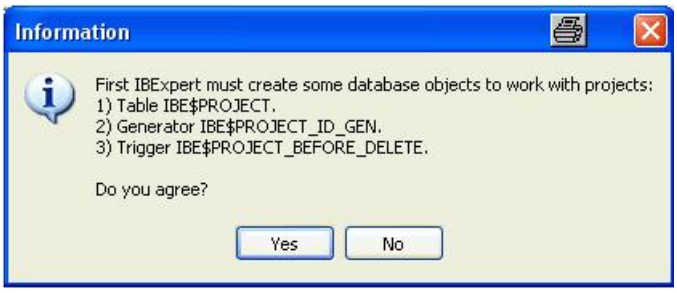
In the DB Explorer, projects can be defined to streamline the overview of database objects currently being worked with.



Database objects within a database can be hierarchically classified (user-specified) as wished. For example, for an *Accounts* project, only those objects necessary for all accounting processes are included, a *Sales* project would include certain objects used in *Accounts* and also, in addition, sales-specific objects.

This is ideal for large software projects in an enterprise.

The first time a folder or object is inserted in the project tab, IBExpert asks for confirmation whether it should create certain system tables for the project page.



This only needs to be confirmed once. Following this, folders and objects can be inserted as wished using the right mouse button menu, [Shift + Ctrl + F] or drag 'n' drop in the Inspector Page Mode, to organize databases individually and personally.

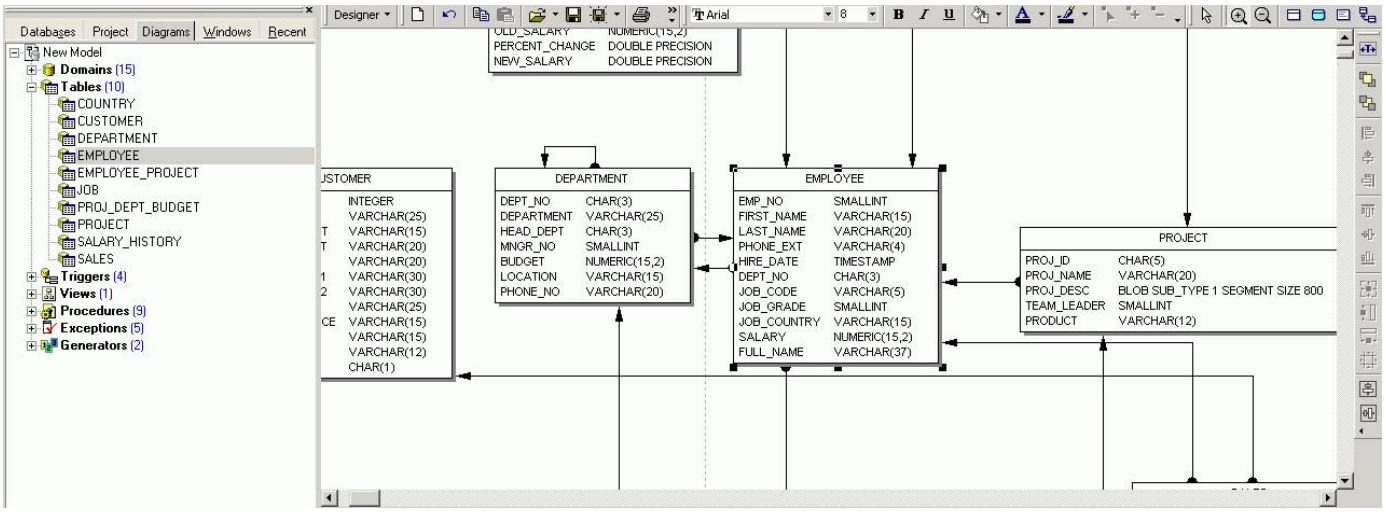
The context-sensitive right-click menu offers a number of further options:

These menu options allow new folders to be created, objects to be added to or deleted from a project (and searched for within the Explorer tree). User items may be created and copied; and the visual display customized (*ShowSQL Assistant*, *Inspector Page Mode*, *Hide Disconnected Databases*). Since IBExpert version 2004.2.26.1 there is also the added option to sort items in alphabetical order, using the menu item *Sort child nodes alphabetically*.

Diagrams (Database Designer)

The *Diagrams* page was added in IBExpert version 2004.9.12.1. It provides a *Model Navigator* to navigate models in the Database Designer quickly and easily.

%



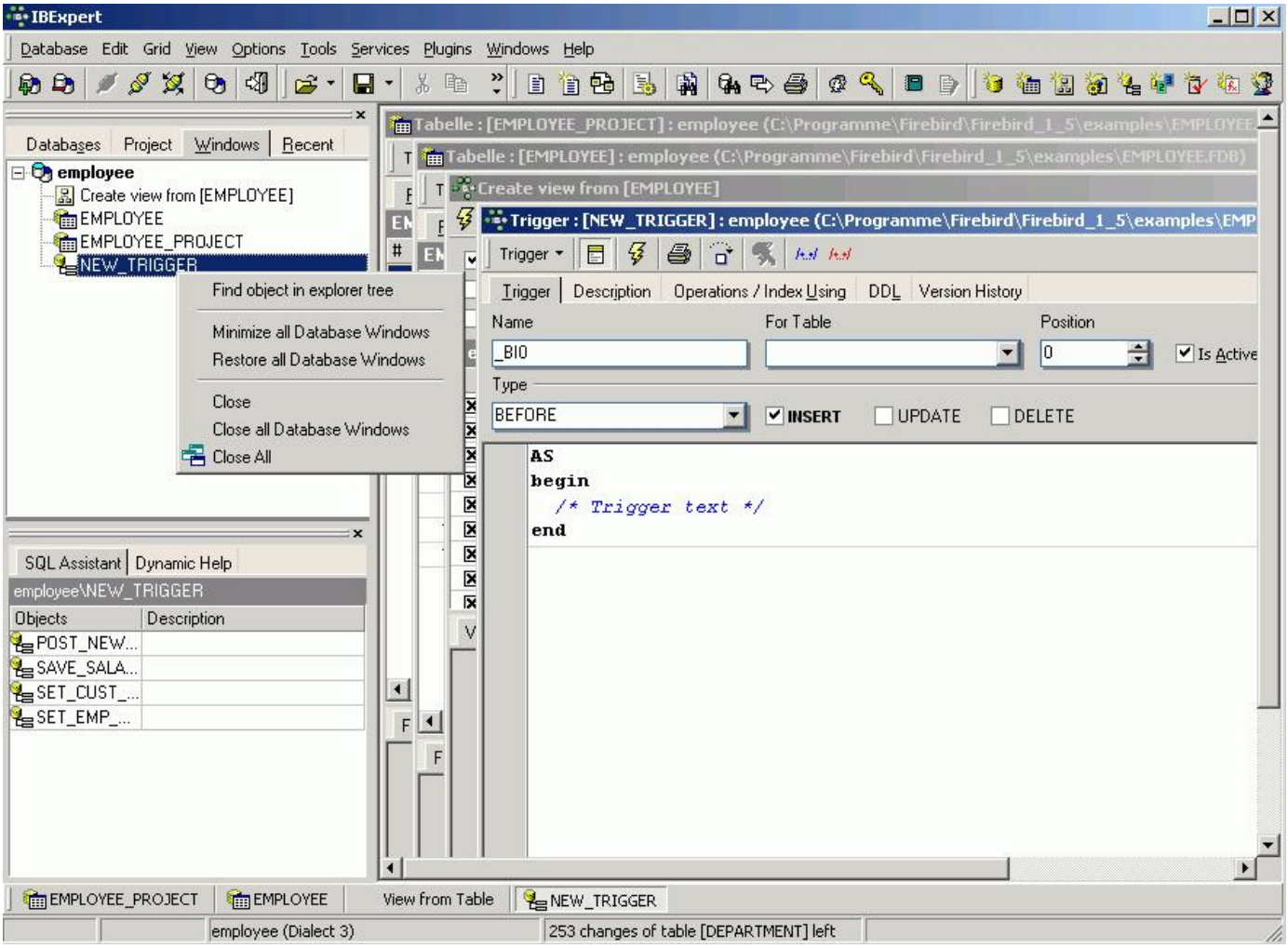
Simply click on an object in the DB Explorer, and it is immediately marked in the main *Database Designer* window. Double-clicking on a selected object automatically opens the Model Options page in the Database Designer.

Please also refer to the Model Navigator in the SQL Assistant.

Windows Manager

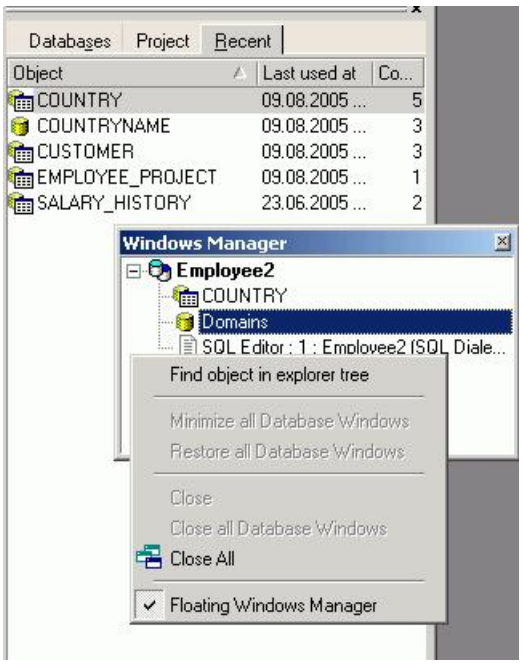
The Windows Manager can be opened using the IBExpert Windows menu item Windows Manager, the key combination [Alt + O], or - of course - by simply clicking on the Windows tab heading directly in the DB Explorer.

In the DB Explorer, the Windows page displays a list of all open windows, and allows the user to change quickly and easily from one window to the next by simply clicking on the object name in the list.



The right mouse button can be used to close individual or all windows, or to find the selected object in the DB Explorer database tree.

A floating *Windows Manager* has been implemented since IBExpert version 2005.08.08. It is now possible to float the Windows tree using the right-click context menu. The floating Window can be returned to the DB Explorer by unchecking the context menu-item *Floating Windows Manager*.



The open windows can also be viewed and selected in the windows bar, directly above the status bar at the bottom of the IBExpert Screen.

Recent List

By clicking on the *Recent* tab in the DB Explorer, a list of the most recent objects worked upon appears.

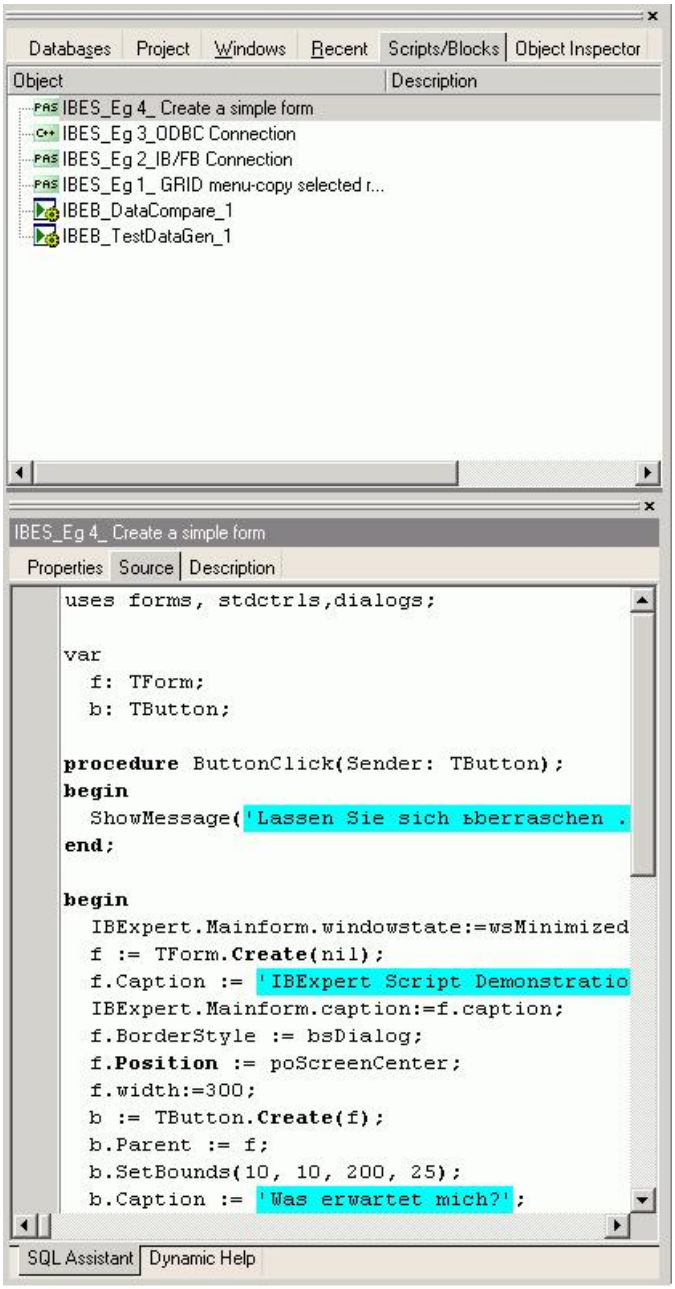
This list can be sorted by object name, date or count in ascending or descending order, by simply clicking on the column header. The object can be reopened by double-clicking.

Databases | Project | Windows | Recent | Help

Objects	Last Used At	Count
EMPNO	24/04/2003 09:23:12	2
PHONE_LIST	24/04/2003 09:22:49	11
ADD_EMP_PROJ	23/04/2003 20:57:25	14
SHOW_LANGS	23/04/2003 20:57:25	3
SHIP_ORDER	22/04/2003 20:46:09	1
ORG_CHART	22/04/2003 20:46:05	4
MAIL_LABEL	22/04/2003 20:46:01	1
GET_EMP_PROJ	22/04/2003 20:45:58	4
DEPT_BUDGET	22/04/2003 20:45:48	2
DELETE_EMPLOYEE	22/04/2003 20:45:41	6
ALL_LANGS	22/04/2003 20:45:33	17
POST_NEW_ORDER	22/04/2003 20:31:21	10
COUNTRY	22/04/2003 20:24:32	11
CUSTOMER	22/04/2003 20:22:00	17
DEPARTMENT	15/04/2003 09:19:39	9
EMPLOYEE	15/04/2003 09:19:39	15
JOB	09/04/2003 12:33:48	19
BUDGET	08/04/2003 12:30:27	5
COUNTRYNAME	08/04/2003 12:26:36	3
SET_EMP_NO	07/04/2003 10:37:06	4
SET_CUST_NO	07/04/2003 10:37:01	6
SAVE_SALARY_CHANGE	07/04/2003 10:36:54	14
PROJECT	03/04/2003 11:39:55	2
RDB$1	01/04/2003 13:23:08	1
ADDRESSLINE	01/04/2003 12:47:33	6
RDB$3	25/03/2003 19:54:37	4
RDB$12	25/03/2003 19:54:11	1
SALARY_HISTORY	25/03/2003 19:51:13	5
EMPLOYEE_PROJECT	25/03/2003 19:49:13	6
PROJ_DEPT_BUDGET	06/03/2003 12:29:56	5
RDB$75	05/03/2003 12:50:10	4
TEST_TABLE1	05/03/2003 12:44:53	2
JOBCODE	05/03/2003 12:44:10	2
SALES	04/03/2003 11:22:21	2
PONUMBER	04/03/2003 10:38:01	1

Scripts/Blocks

This page is new to IBExpert version 2005.12.04. It displays all existing IBEScripts and IBEBlocks saved locally in the database.



There are two ways to store the blocks and scripts: (i) in a registered database or (ii) in the IBExpert User Database, which can be activated using the IBExpert Options Menu / Environment Options / User Database.

To create a new script in a registered database, click on the *Scripts* node in the connected database, and use the context-sensitive (right-click) menu to create a new script. You can also create IBEBlocks and Firebird 2 blocks (EXECUTE BLOCK) in this way within your database. Each script or block must have a unique name (up to 100 characters) within the database.

To create a new script in the User Database, first enable the option in the IBExpert Options menu / Environment Options / User Database and restart IBExpert. You should now see a new table in the Database Explorer: *Scripts/Blocks*. This allows you to create scripts and blocks using the context-sensitive menu from the *Scripts/Blocks* tree and also organize them in folders.

We strongly recommend using the IBExpert User Database as a main storage for IBExpert, even if you do not need the scripts/blocks feature.

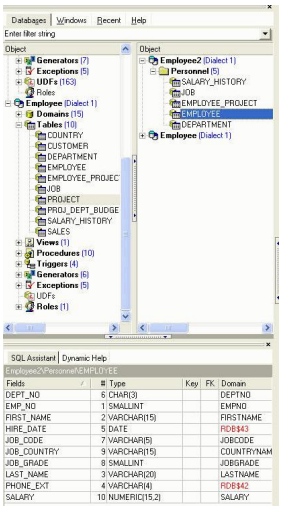
Since IBExpert version 2006.01.29 it is possible to execute Firebird 2.0 blocks, IBEBlocks and IBEScripts stored in registered databases or in the IBExpert User Database directly from the DB Explorer. Simply use the DB Explorer right-click context menu or open the script in the Block Editor and execute using [F9] . IBExpert version 2008.08.08 introduced the possibility to recreate selected views based on IBEBlock and the `ibec_GetViewRecreateScript` function using the DB Explorer context-sensitive menu, *Apply Block*.

Please refer to IBEBlock and IBEScripts for further information concerning the many possibilities of these comprehensive features. Refer to BlockEditor for information regarding the creation, alteration and execution of blocks and scripts.

Inspector Page Mode

When either the *Database* Page or the *Project* Page in the IBExpert DB Explorer is active (i.e. visible in the foreground), it is possible to compare the two to each other by switching on the *Inspector Page Mode*.

This can be done using the right-click menu and selecting *Inspector Page Mode*, to produce two adjacent windows:

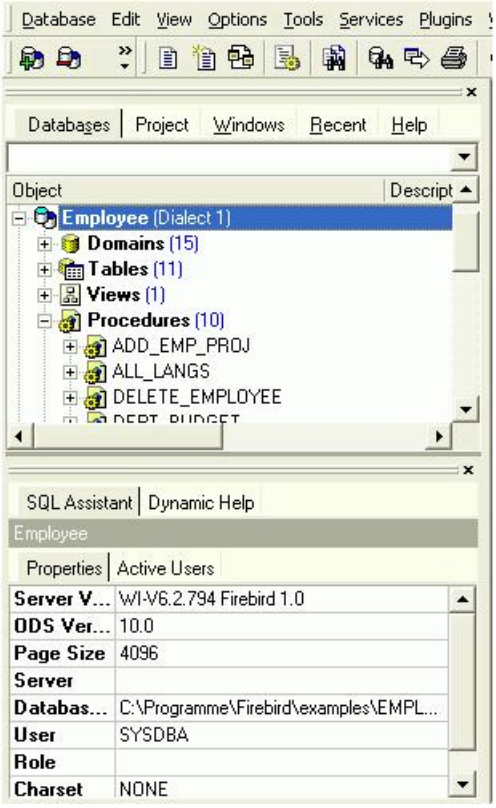


Objects can be dragged 'n' dropped from one window to the other, allowing a quick and easy selection of those objects necessary for a project.

To return to a single window display in the DB Explorer, simply right-click and the select the menu item *Inspector Page Mode* again.

(5) SQL Assistant

The IBExpert SQL Assistant offers additional detailed information regarding the highlighted database, object or group of objects in the DB Explorer. It can be found in the lower left-hand part of the screen, directly below the DB Explorer.



When a database in the DB Explorer is highlighted, the *Properties* page displays the actual server version of InterBase or Firebird (this can be subsequently corrected in the Database Registration if specified wrongly or previously unknown). The *Active Users* page shows which users are currently logged on to the database.

Selecting an object group in the DB Explorer displays a list of the corresponding objects. Selecting a single object displays detailed object information and content in the SQL Assistant.

New to IBExpert version 2006.10.14:

- When a database node is selected in the DB Explorer, the SQL Assistant displays the full path to a client library that is used while working with the database.
- the SQL Assistant displays the client library version number for an active database node.

When a table is selected in the DB Explorer, the fields are not only displayed in the SQL Assistant, but can also be selected and incorporated into any of the SQL Editors using drag 'n' drop. Since version 2004.2.26.1 this has been greatly improved. When an object node(s) is dragged from the DB Explorer or SQL Assistant, IBExpert will offer various relevant versions of text to be inserted into the Code Editor.

The SQL Assistant can be blended in and out as wished using [Ctrl + A] or the DB Explorer right-click menu item *ShowSQL Assistant*.

Dynamic Help

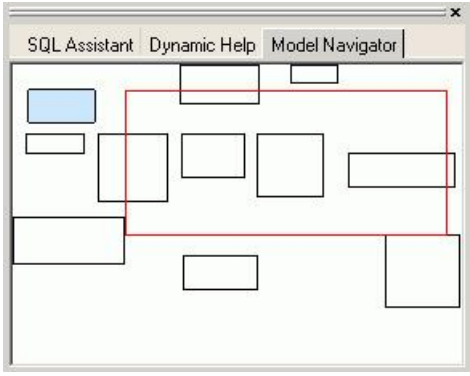
The *Dynamic Help* page can be found in the SQL Assistant (underneath the DB Explorer) and offers context-sensitive help.

Since IBExpert version 2004.2.26.1, this has been replaced by a new context-sensitive dynamic help system. Pressing [F1] in any of the IBExpert forms now opens a new web-based Help page. It is also possible to download all Help files from http://www.ibexpert.info/documentation/documentation.zip and unzip this in the IBExpert main directory with subdirectories (there must be a new subdirectory called documentation).

If a local Help document is available, it will be opened in the browser. Otherwise the browser will open the page from our web server.

Model Navigator (Database Designer)

The *Model Navigator* page was added in IBExpert version 2004.9.12.1. It provides a visual orientation to aid navigation of models in the Database Designer.



The red rectangle indicates which part of the database model is currently being displayed in the main Database Designer window. It is possible to move this rectangle by drag 'n' dropping with the mouse - much quicker and easier than moving about in the main Database Designer window.

Please also refer to the Diagrams page in the DB Explorer which lists all model objects in the usual DB Explorer tree form.

(6) Windows bar

The IBExpert windows bar is a horizontal bar and can be found in the lower area of the screen, directly above the status bar:

This displays the number and type of open windows in IBExpert; the symbols indicating the editor type (e.g. Table Editor, Procedure Editor, etc.), followed by the object name or editor type.

(7) Status bar

The IBExpert status bar is a horizontal bar found in the lower area of the screen, directly below the windows bar:

1: 1		Employee (Dialect 1)	253 changes of table [TEST_TABLE1] left	49 MB left

This displays information concerning the current status of, for example, the connected database, the IBExpert window contents and memory.

253 changes of table left

Each table in an InterBase/Firebird database has its own metadata changes counter. The metadata of each table can be altered 255 times (add or remove columns, change field type etc.). This limitation is because Firebird/InterBase sets an internal 1 byte flag, which is stored alongside each data set, representing the so-called record structure version. For example, you have 1,000 data sets in a table with five fields. You extend the table to six fields, and then add a further 1,000 data sets. The old first 1,000 data sets are not revised at all, but are still stored with the old data structure, unless you have instructed the server to set the data content of the sixth field for these old data sets at `NULL` or a specified default value. If this new field is created with a `NOT NULL` constraint, these old fields will all need to be updated. The internal flag simply ensures that a maximum of 255 such changes are possible.

When any of these counters reaches the value of 255 it is not possible to alter any tables any further, and a database backup and restore is necessary. The backup and restore ensure that all data sets are now stored with the current single valid record structure, and you can continue to make further table alterations.

IBExpert indicates in the status bar how many changes may be made in the table with the lowest value (*253 changes of table [table_name] left*) in the database before being forced to perform a database backup and restore. This message may be deactivated if wished, using the IBExpert menu item, Database / Register Database or Database / Database Registration Info, and checking the option *Don't display metadata changes counter info* on the *Additional* page.

Exit

Exit is the command used to close IBExpert. The program can be closed by using either the menu item Database / Exit, or clicking the black X button in the top right-hand corner of the screen. Alternatively the key combination [Alt + F4] may be used.

IBExpert requires confirmation that you really wish to exit the program - either click on Yes or press the Return/Enter key. Should you wish to eliminate this default setting, uncheck the Confirm Exit box found in the IBExpert Options / Environment Options menu under Confirmations.

Any editors left open at the time of exiting, will automatically be loaded the next time that IBExpert is started, unless the following default setting is switched off: Options / Environment Options / Preferences - uncheck *Restore Desktop after Connect*.

All connected databases are automatically disconnected when IBExpert is shut down.

See also:
Environment Options
IBExpert Toolbars
Toolbar options SQL Editor
Database Objects

Where to go from here

If you're just starting out, take the time to read through these documentation sources intended for beginners:

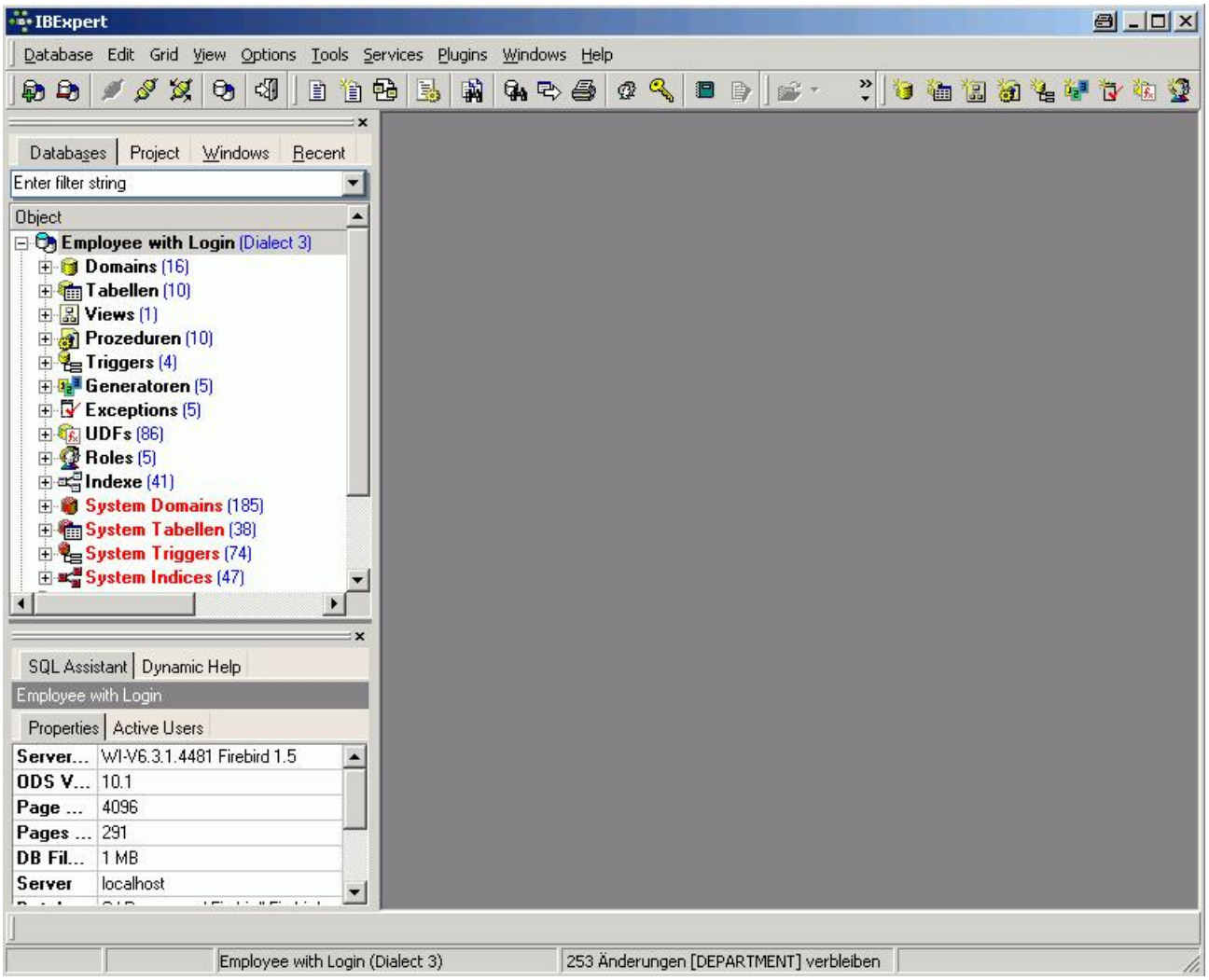
- Firebird Administration using IBExpert - an introduction for DBAs using Firebird or InterBase together with IBExpert.
- Firebird 2 Administration Handbook - an introduction for DBAs using Firebird and its command-line tools.
- Firebird Development using IBExpert - an introduction for developers using Firebird or InterBase together with IBExpert.
- Firebird 2 Cheat Sheet - this provides a summary of the most common definitions and functions
- Definitions and Field Definitions are IBExpert's own definitions and explanations of database basics
- Glossary - definitions of many terms that may be new to you.

IBExpert Database menu

A relational database is a collection of tables related to each other, each storing a specific set of data. A database also contains indices, business rules and processes, for the database administration. It can be considered to be a collection of pages, each page being of a pre-defined size, which is determined when the database is created.

The data itself may contain any information, be it for business accounts, sales, scientific measurement logging or personal addresses and finances. The information stored in a database may be shared by more than one application.

Available databases can be viewed in IBExpert in the left-hand panel, the DB Explorer. Connected registered databases are displayed in bold type.



The relational system assumes the following:

1. The physical storage model and the logical data storage in files are independent of each other.
2. All data is stored in tables.
3. Users do not need to know which files are stored how and where. Access occurs via tables, which represent a logical view of data.
4. A data set's physical position in the database is irrelevant to the user.
5. The relational database administrates all information necessary for internal access optimization internally, using indices.
6. The relational database undertakes the data integrity checks independently.

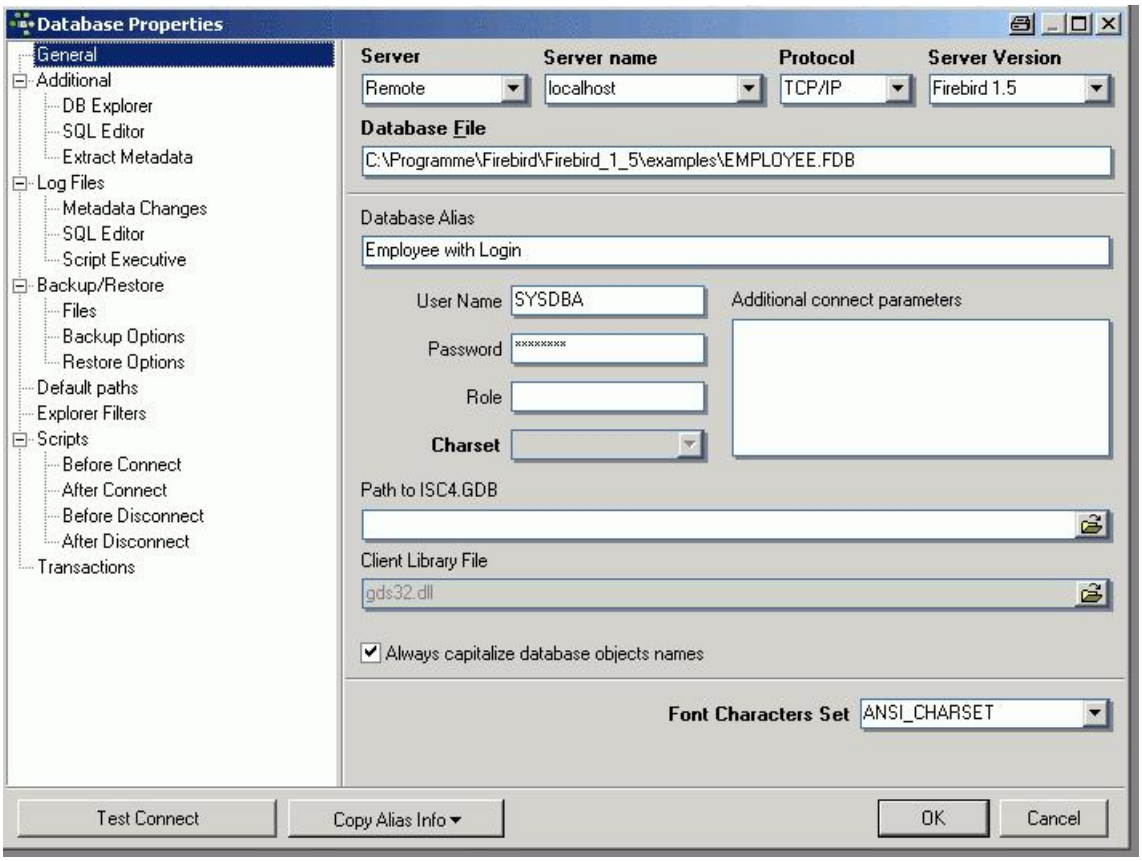
InterBase/Firebird administrates data in database objects. Within the database, the following database objects (database metadata) can be created and maintained:

1. Domains
2. Tables
3. Generators
4. Constraints
5. Indices
6. Views
7. Triggers
8. Stored Procedures
9. Exceptions
10. Blob Filters
11. User-Defined Functions (UDFs)

See also:
Database toolbar

Database Registration Info

Information appertaining to any of the registered databases can be viewed in IBExpert in the Database Properties dialog, started using the menu item Database / Database Registration Info... or the DB Explorer right-click menu:



The information displayed here is that which was entered, when the database was originally registered (please refer to Register Database for details).

The tree in the left panel shows the various registration options available. Certain items may be amended here. Again please refer to Register Database for further information.

New in version 2.5.0.47: it is possible to automatically connect to a database when starting IBExpert. Use the following menu: Database Registration Info / Additional and check: Open database when IBExpert starts.

New in version 2004.04.01.1: under Database Registration Info / Additional there are now two additional options:

- Disable plan request in SQL Editor
- Disable performance analysis.

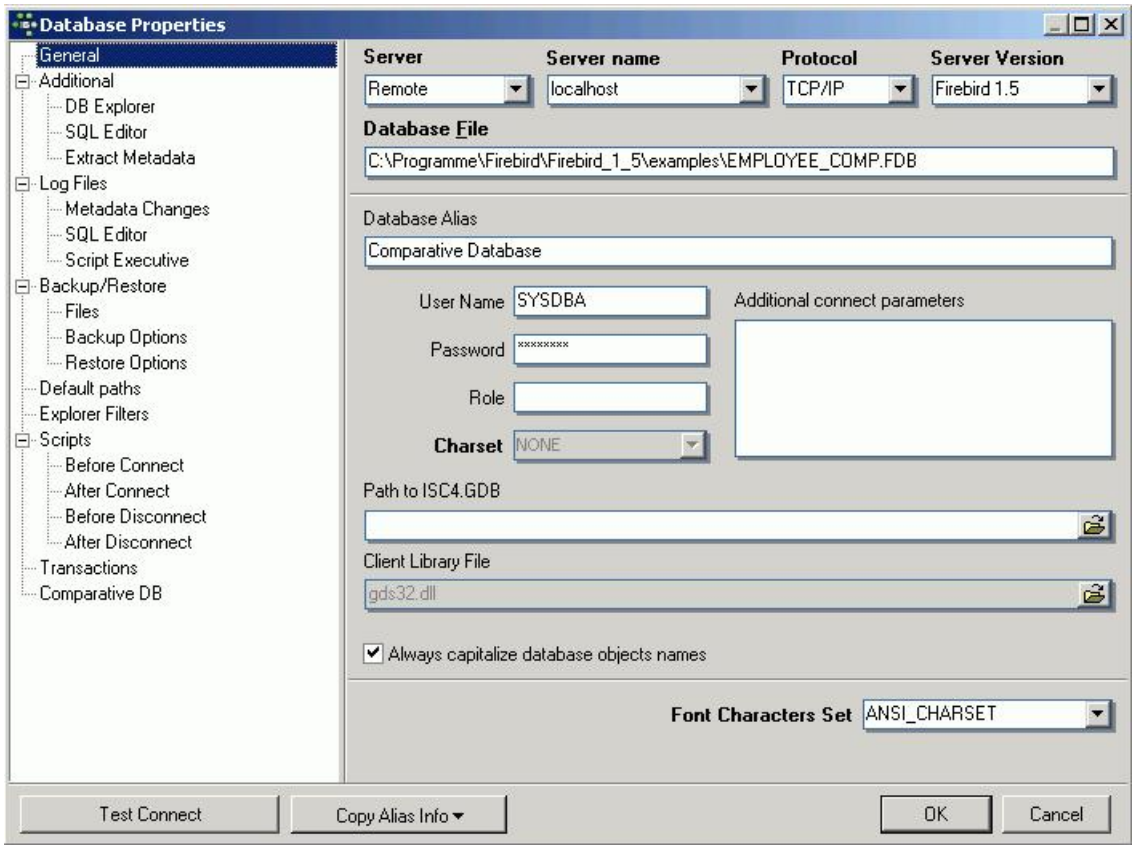
New to version 2003.12.18.1: the added possibility to execute SQL scripts before and after connecting to the database and before and after disconnecting from the database. And under Database Registration Info / Additional there is now the additional option - *Always prompt for a user name and password*. If this option is activated, IBExpert will display a login prompt dialog each time you try to connect to the database.

See also:
Register Database
Default character set

Register Database

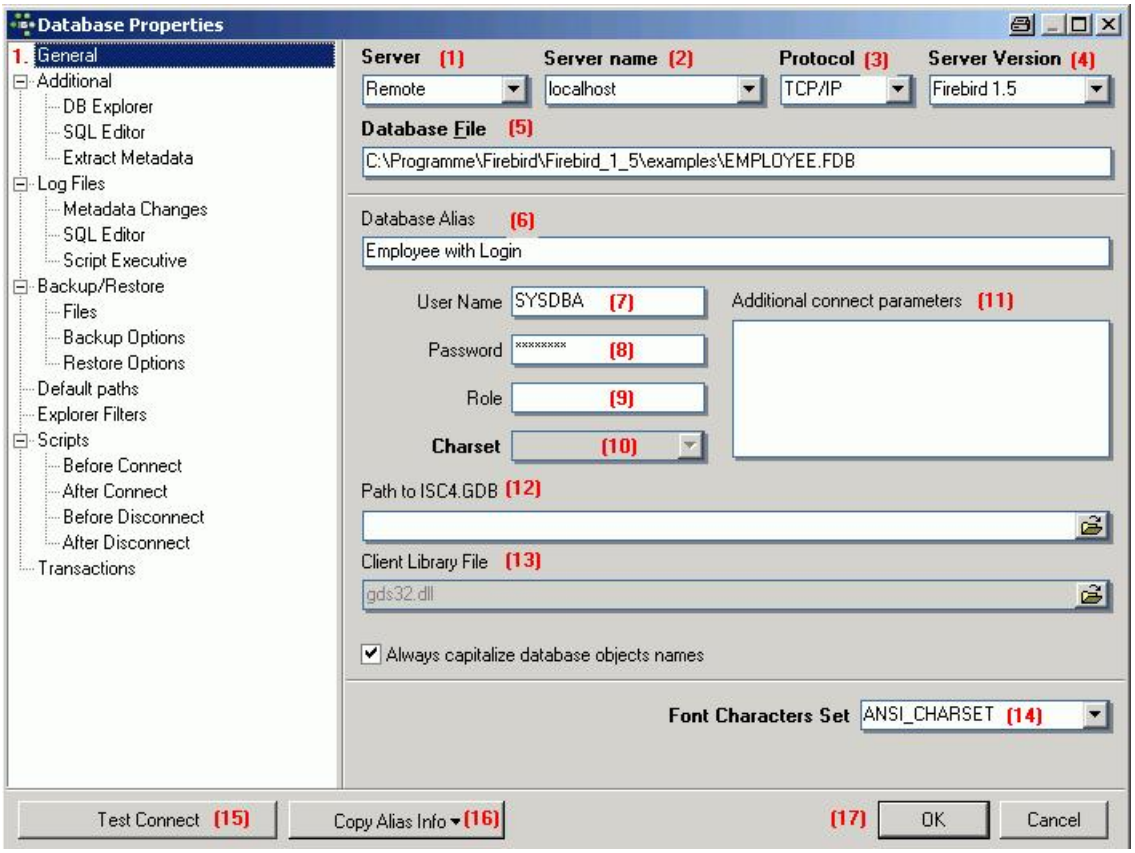
Database registration is necessary, in order for IBExpert to recognize the presence of a database. It is possible to specify certain options, settings and defaults here. The Database Registration Editor can be opened using the IBExpert menu item Database / Register Database, or key combination [Shift + Alt + R]. It is automatically generated when the *Register Database After Creating* checkbox is flagged in the Create Database dialog.

The Database Registration dialog is split into two sections: on the left-hand side a tree overview of the various registration options is displayed; the right input panel shows the information and setting options available for each tree subject.



General

The following entry fields allow the user to specify certain general properties and defaults for the database to be registered.



(1) Server: firstly the server storing the database needs to be specified. This can be local or remote (see Create Database). By specifying a local server, fields (2) and (3) are automatically blended out, as they are in this case irrelevant.

(2) Server name: must be known when accessing remotely. The syntax is as follows:

- `Windows SERVER_NAME:C:pathdatabase.gdb`
- `Linux SERVER_NAME:/path/database.gdb`

The standard port for InterBase and Firebird is `3050`. However this is sometimes altered for obvious reasons of security, or when other databases are already using this port. If a different port is to be used for the InterBase/Firebird connection, the port number needs to be included as part of the server name. For example, if port number `3055` is to be used, the server name is `SERVER/3055`.

(3) Protocol: a pull-down list of three options: TCP/IP, NetBEUI or SPX. TCP/IP is the worldwide standard.

(4) Server versions: this enables a server version to be specified as standard/default from the pull-down list of options. To specify a default server version, use the IBExpert Options menu item / Environment Options / Preferences to select your preferred server version.

(5) Database File: by clicking on the folder icon to the right of this field, the path can easily be found and specified and the database name and physical path entered. The database name must always be specified with the drive and path when registering a database. Please note that the database file for a Windows server must be on a physical drive on the server, because InterBase/Firebird does not support databases on mapped drive letters.

For example for Firebird:

- `C:ProgramsFirebirdFirebird_1_5examplesEMPLOYEE.FDB`

for InterBase:

- `C:ProgramsInterbaseexamplesEMPLOYEE.GDB`

(6) Database Alias: descriptive name for the database (does not have to conform to any norms, but is rather a logical name). The actual database name and server path and drive information are hidden behind this simple alias name - aiding security, as users only need to be informed of the alias name and not the real location of the database. The connection string usually consists of the server name (or localhost) followed by the drive and path to the database file, with the database file name concatenating on the end. If an alias and its string are already specified in the `aliases.conf` on the server, the client can, with the newer Firebird versions, use the connection string, `servername:aliasname`. The `alias.conf` shows the server where the client wants to go.

Please refer to the *Firebird Administration* chapter, Alias, files and paths for detailed information about database aliases.

(7) User Name: the database owner (i.e. the creator of the database) or SYSDBA.

To the right of the user name a checkbox option has been introduced for Firebird 2.1 database registrations, *Trusted Authentication*.

(8) Password: if this field is left empty, the password needs to be entered each time the database is opened. Please refer to Database Login for further information. The default password for SYSDBA is `masterkey`. Although this may be used to create and register a database, it is recommended - for security reasons - this password be changed at the earliest opportunity.

(9) Role: an alternative to (7) and (8); can initially be left empty.

(10) Charset (abbreviation for Character Set): Here the default character set can be specified. This is useful, when the database is created to be used for foreign languages, as this character set is applicable for all areas of the database unless overridden by the domain or field definition. If not specified, the parameter defaults to `NONE`, i.e. values are stored exactly as typed. For more information regarding this subject, please refer to Charset/Default Character Set. If a character set was not defined when creating the database, it should not be used here.

(11) Additional connect parameters: input field for additional specifications. For example, system objects such as system tables and system generated domains and triggers can be specified here. They will then automatically be loaded into the DB Explorer when opening the database alias.

(12) Path to ISC4.GDB: This can be found in the InterBase or Firebird main directory. This database holds a list of all registered users with their encrypted passwords, who are allowed to access this SERVER.

When creating new users in earlier InterBase versions (<6), IBExpert needs to be told where the `ISC4.GDB` can be found. Since InterBase version 6 or Firebird 1 there is a services API. So those working with newer versions may ignore this field!

(13) Always capitalize database objects" names (checkbox): this is important as in SQL Dialect 3 as entries can be written in upper or lower case (conforming to the SQL 92 standard). InterBase however accepts such words as written in lower case, but does not recognize them when written in upper case. It is therefore recommended this always be activated.

(14) Font character set: this is only for the IBExpert interface display. It depends on the Windows language. If an ANSI-compatible language is being used, then the `ANSI_CHARSET` should be specified.

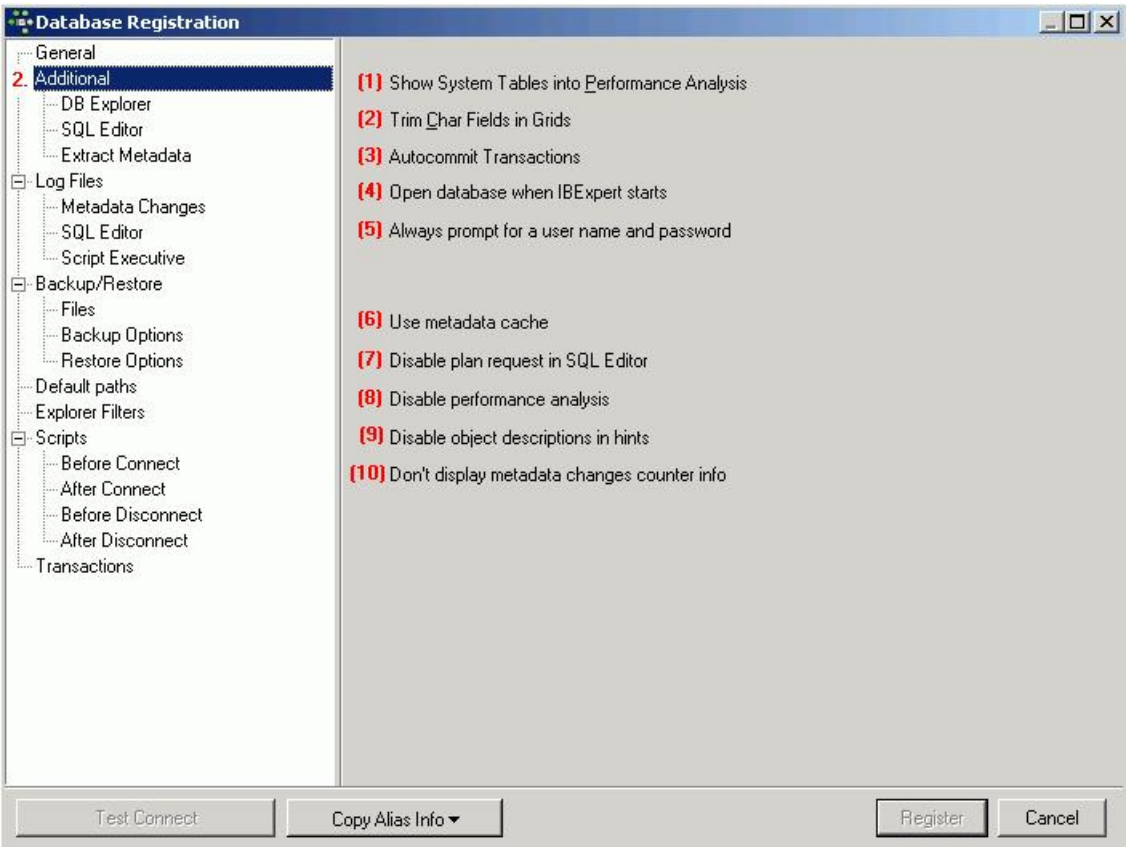
(15) Test connect: the Comdiag dialog appears with a message stating that everything works fine, or an error message - please refer to the IBExpert Services menu item, Communication Diagnostics for more details.

(16) Copy Alias Info: here alias information from other existing registered databases can be used as a basis for the current database. Simply click on the button and select the registered database which is to be used as the alias.

(17) Register or Cancel: after working through all the options listed in the tree view on the left, the database can be registered or cancelled.

Additional

The Database Registration / Additional options are as follows:



(1) Show System tables into Performance Analysis: the developer can choose whether he also wishes to have the database system tables (in addition to the user-defined objects) included in the Performance Analysis found in the SQL Editor, Stored Procedure Editor and Visual Query Builder.

(2) Trim Char Fields in Grids: adapts field length to ideal length in all grids (see Table Editor / Data and SQL Editor / Results as well as the IBExpert Grid menu).

(3) Autocommit Transactions: This allows all transactions to be committed immediately (i.e. IBExpert no longer asks for confirmation of a commit command and there is *NO* option to rollback). This is an *EXTREMELY* dangerous option! For example, if an irreversible DROP command has been wrongly entered (e.g. instead of typing a FIELD_NAME the DATABASE_NAME is mistakenly entered), it is still automatically committed.

(4) Open database when IBExpert starts: New in version 2.5.0.47: checking this option automatically connects this database when IBExpert is started.

(5) Always prompt for a user name and password: New in version 2003.12.18.1: if this option is activated, IBExpert will display a login prompt dialog each time you try to connect to the database.

(6) Use Metadata cache: e.g. when accessing remotely using a modem line, the InterBase server can only be accessed at a limited speed. IBExpert needs to know which information it needs to fetch, and this may take some time. If the metadata cache is checked, IBExpert does not download the complete database each time, only the information that it really needs.

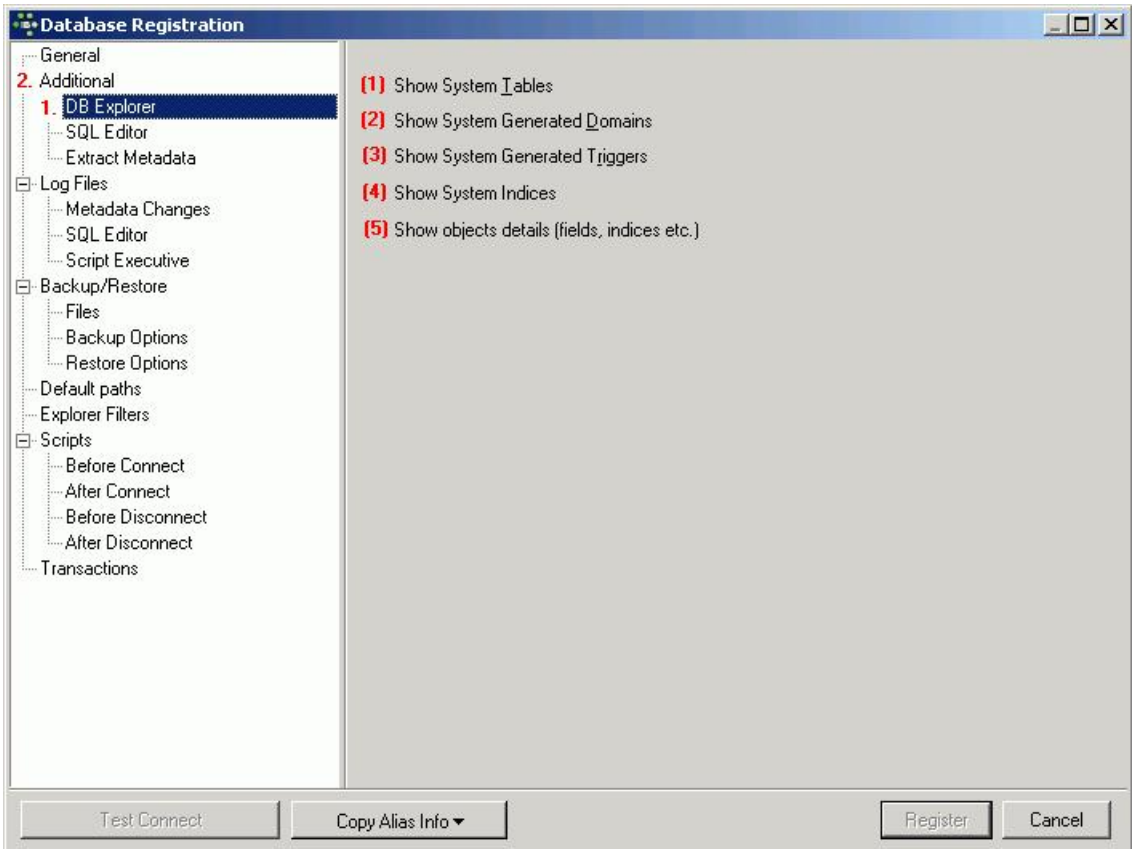
(7) Disable plan request in SQL Editor: New option in version 2004.04.01.1: this deactivates the query plan displayed in the lower panel of the Results page in the SQL Editor.

(8) Disable performance analysis: New option in version 2004.04.01.1: this deactivates the Performance Analysis page in the SQL Editor. This may be desirable, when working remotely on a slow modem connection.

(9) Disable object description in hints: These hints appear when you move the mouse cursor over the column captions in the Data Grid. If descriptions in these hints are not disabled IBExpert executes some SELECTS to get them from the database. If you"re working with the database using a slow modem connection this decrease the performance dramatically.

(10) Dont *display metadata changes counter info;* *This deactivates the message* 253 changes to [TABLE] left", which is displayed in the status bar.

Additional/DB Explorer



(1) Show System Tables: tables generated by InterBase/Firebird are displayed in the IBExpert DB Explorer in red.

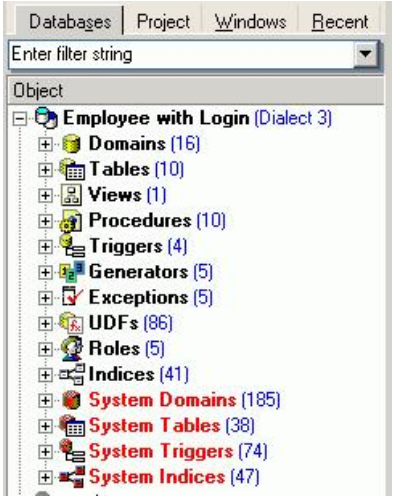
(2) Show System Generated Domains: domains generated by InterBase/Firebird are displayed in the IBExpert DB Explorer in red.

(3) Show System Generated Triggers: triggers? generated by InterBase/Firebird are displayed in the IBExpert DB Explorer in red.

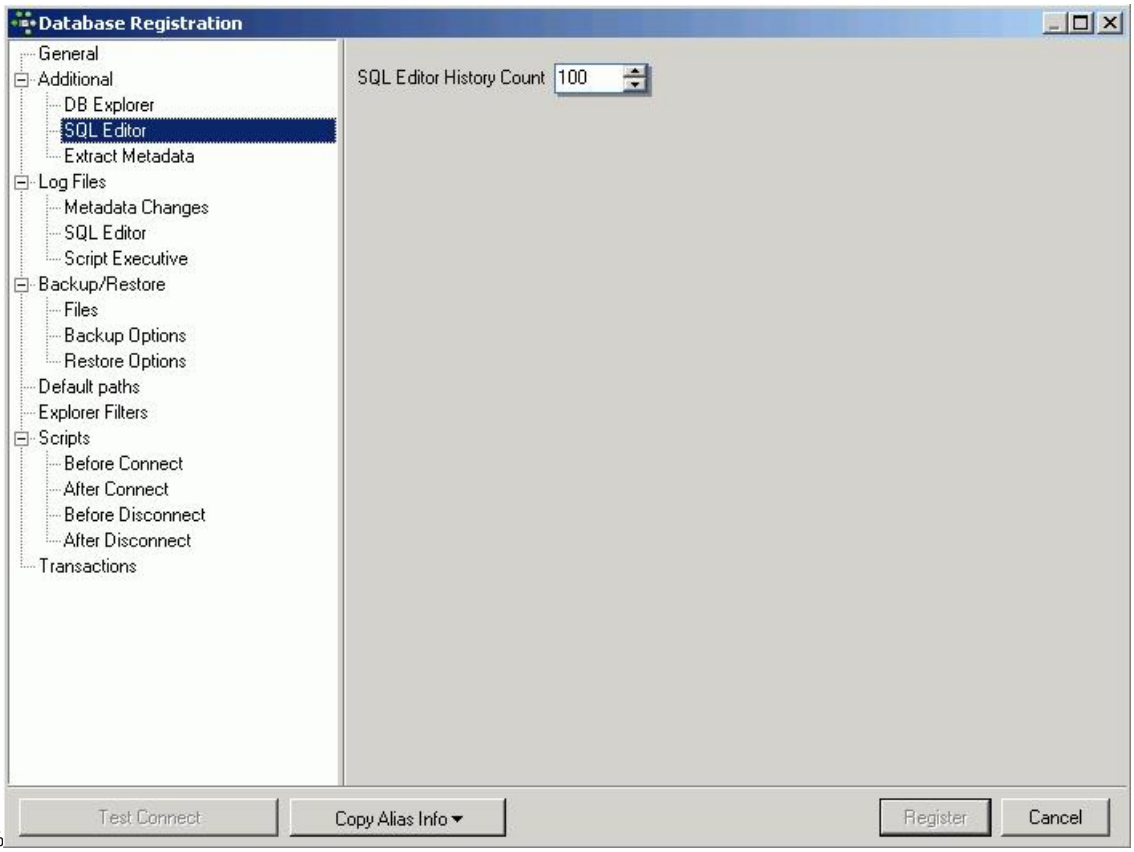
(4) Show System Indices: indices generated by InterBase/Firebird are displayed in the IBExpert DB Explorer in red.

(5) Show objects details: (fields, indices etc.)

For database development it is wise to have all these items visible in the DB Explorer.



Additional/SQL Editor

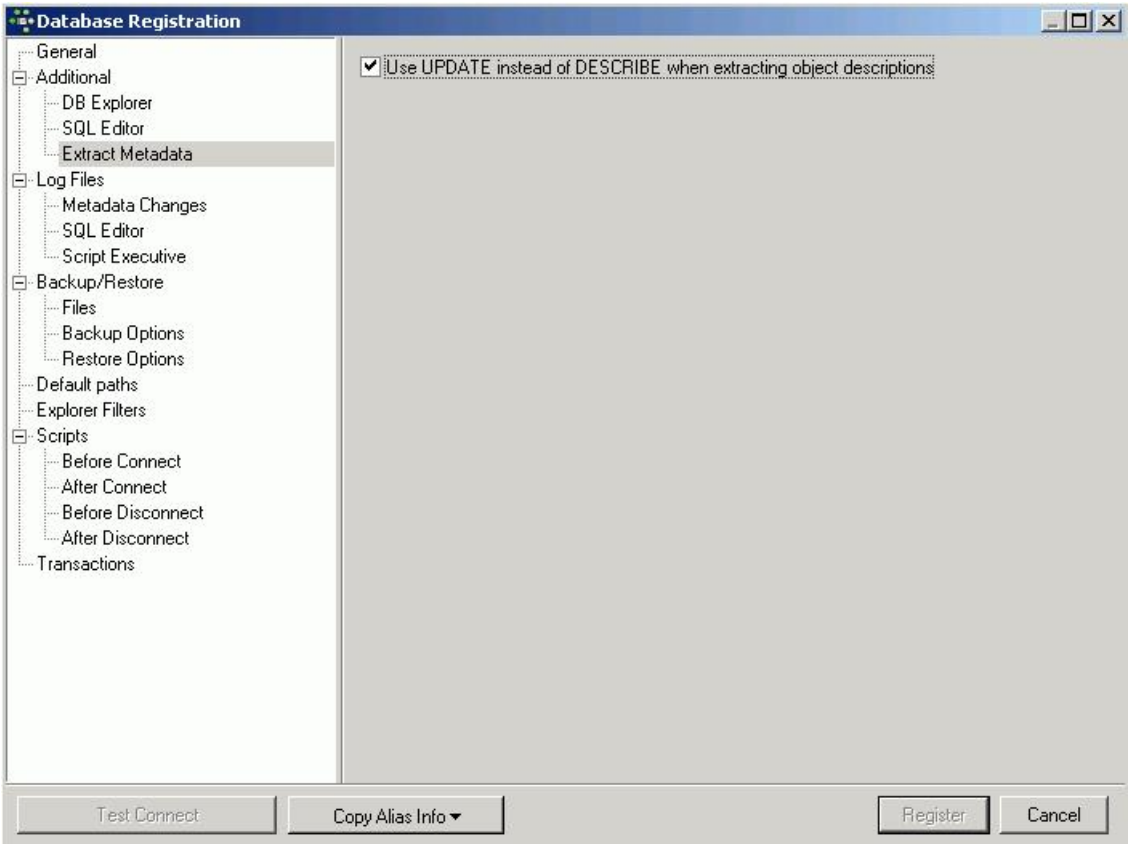


%

The *SQL Editor History Count* determines the number of SQLs that are saved and displayed in the IBExpert SQL Editor. Here the default value of 100 can be adjusted as wished.

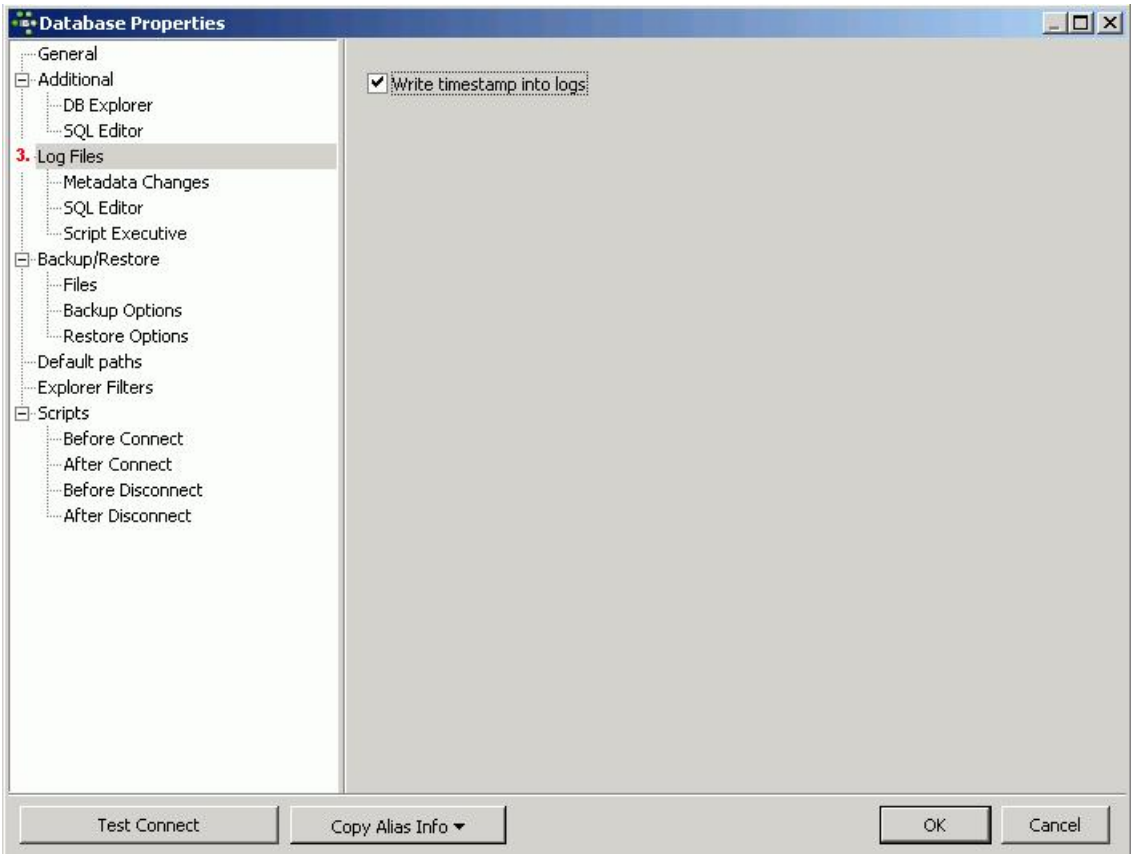
Additional/Extract Metadata

New to IBExpert version 2005.04.24.1: this option allows you to check the new IBExpert feature "Extract Metadata"-"Use UPDATE instead of DESCRIBE" on the *Options* page in the Extract Metadata dialog. If it is enabled, IBExpert will generate an `UPDATE RDB$xxx SET RDB$DESCRPTION ...` statement instead of `DESCRIBE` while extracting metadata.



Log Files

If you would like IBExpert to protocol all statements that change metadata and/or are executed from the SQL Editor, use this section to enter path and file names. This is useful for keeping a record of which changes were made to the data structure in IBExpert.



Write Timestamp into logs: the timestamp option is useful for noting date and time on logs.

IBExpert version 2008.02.19 introduced the possibility to include a date part into log file names. This allows you to create daily/monthly logs automatically.

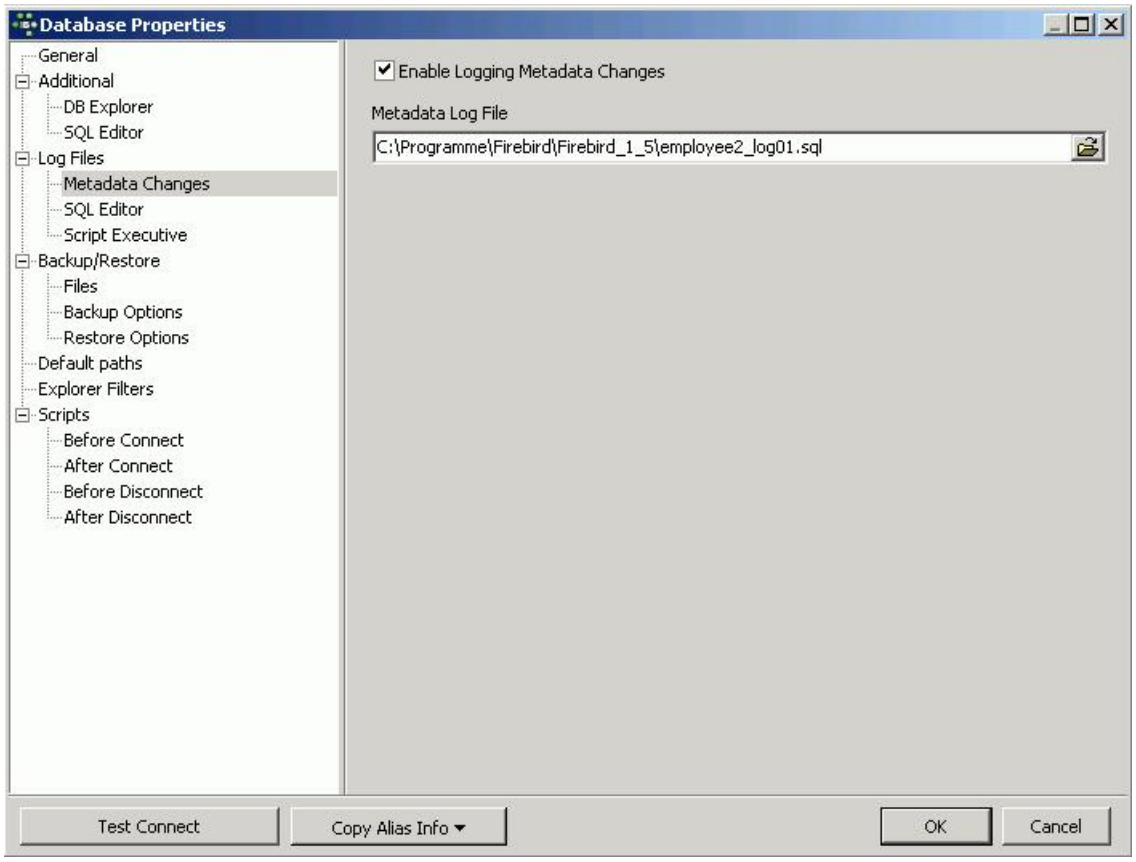
The following substrings in a log file name will be replaced with a current date:

- `=date=yyyy-mm-dd`
- `=date=yyyy-mm-dd%=<date format string>%`
- `=date=yyyy-mm-dd` is a short form of the date template and is equal to `=date=yyyy-mm-dd%=yyyy-mm-dd%`.

Examples

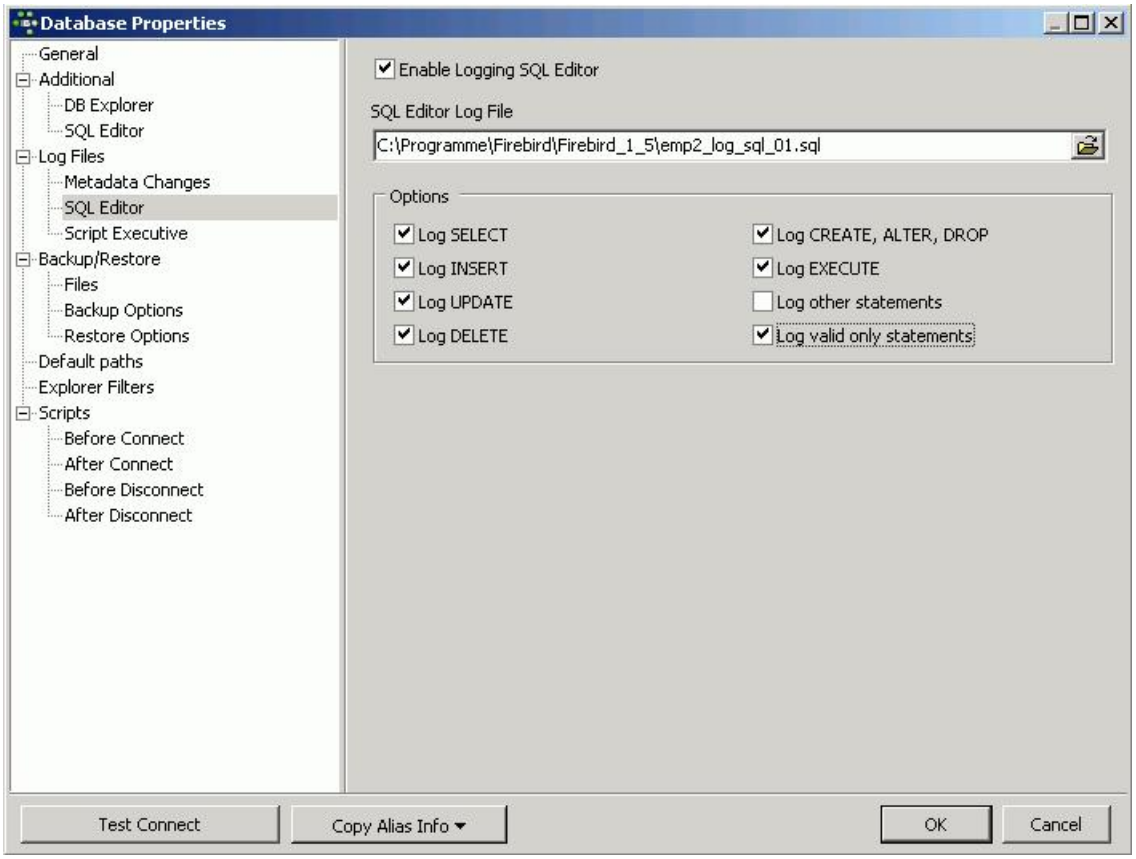
`D:\MyLogs\TestDB\=date=yyyy-mm-dd.sql` - file name for a simple daily log. `D:\MyLogs\TestDB\=date=yyyy-mm-dd%=mmmm of yyyy%\=date=yyyy-mm-dd%=yyyy.mm.dd%.sql` - a separate directory (`'January 2008'` etc.) will be created for each month.

Log Files - Metadata changes



Enable Logging Metadata Changes: allows all changes to metadata to be logged, in order to follow all alterations to the data structure.

Log Files - SQL Editor



Enable Logging SQL Editor: Allows all SQL Editor work to be logged - a useful option, which should be checked. Should the log files become too large, older logs can always be deleted at regular intervals.

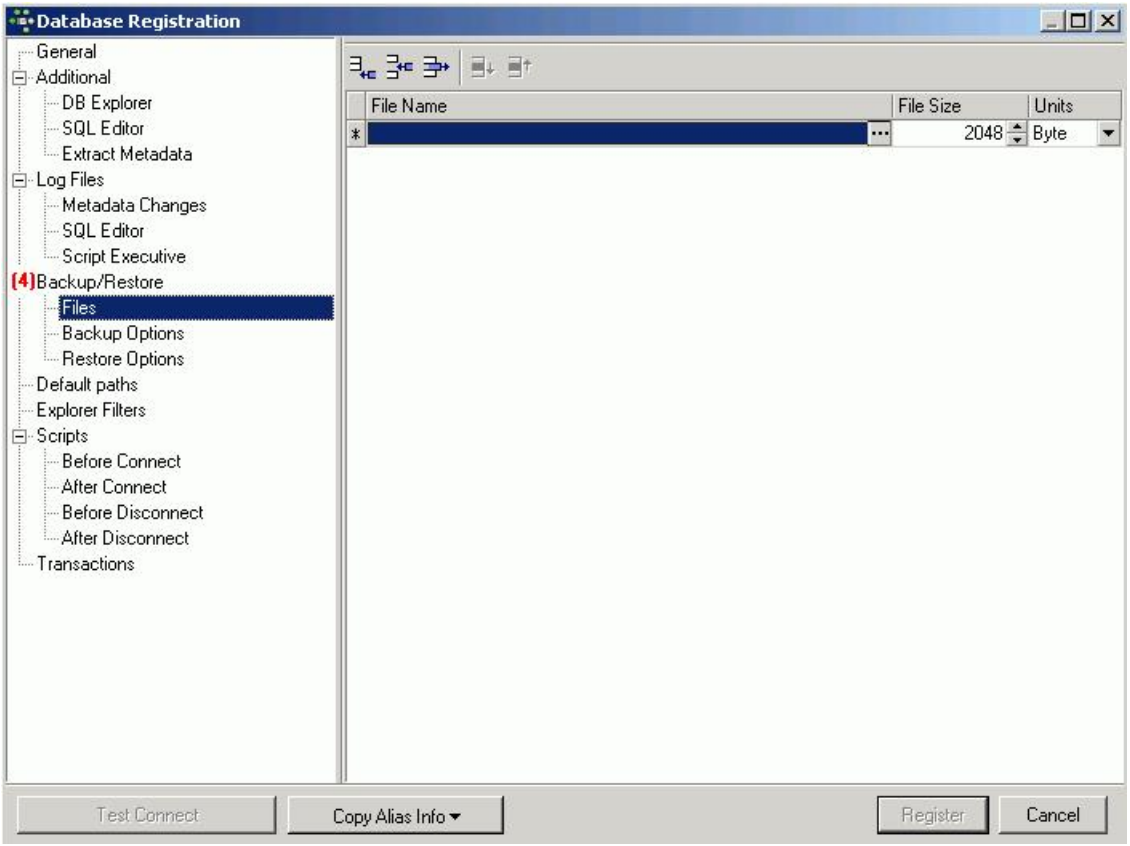
Log Files - Script Executive'

ttp://ibexpert.net/ibe/img/img_2607_Debi0006.gif

Enable Logging Metadata Changes: checkbox to specify whether all alterations to metadata should be logged or not.

Backup/Restore

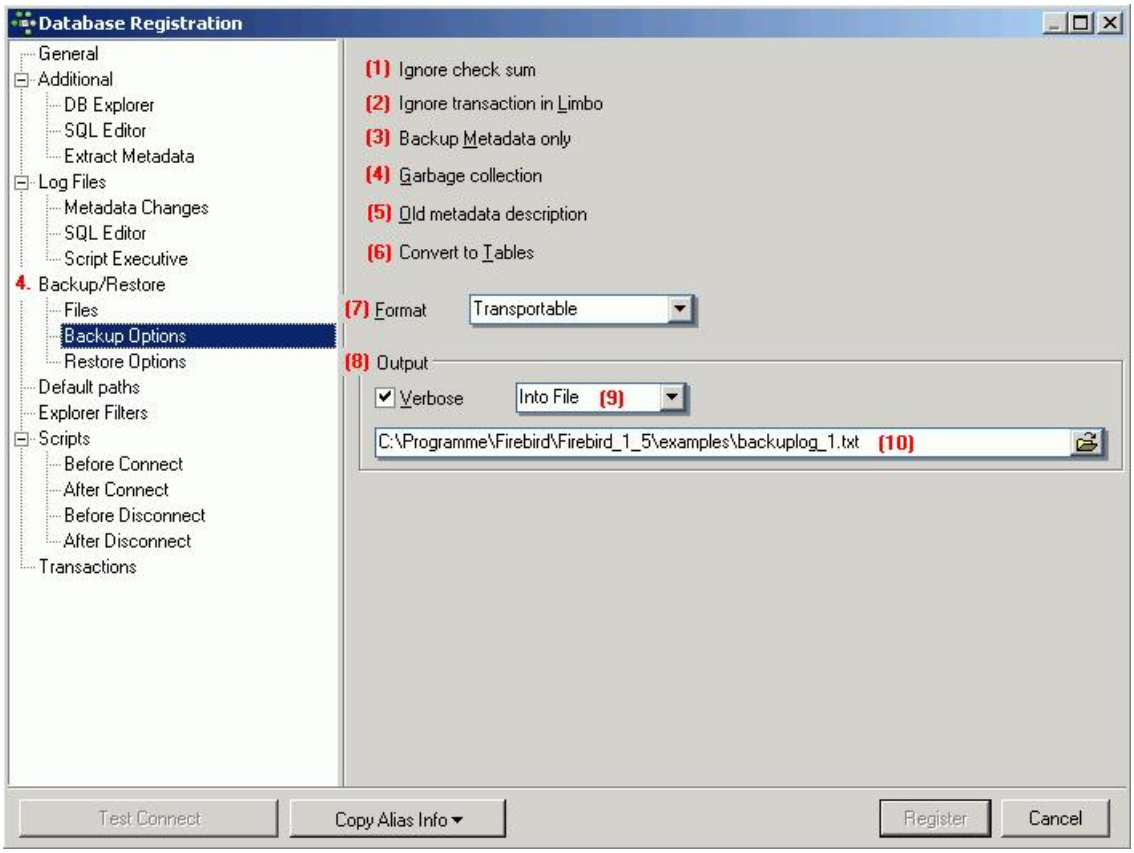
Files



Backup and restore file names and options can be specified for each database alias. This makes it easier to backup a database with a single mouse click from the IBExpert Services menu.

Using the first icon on the left a file name can be specified as the default file for backups. When left empty, the backup file name must be specified for each backup. For versions since Firebird 1.0 or InterBase 6.5 the file size is irrelevant (64B file system). Secondary backup files can also be specified here.

Backup Options



(1) **Ignore check sums:** ignores any check sum errors and continues to backup the database. This option should be selected if a backup is being performed because database errors are suspected. If this option is not checked, the backup is aborted if a check sum error is found. This is one possibility to force a backup for a corrupt database. Please note that checksums are not maintained in UNIX versions.

(2) Ignore Transactions in Limbo: in limbo transactions are those which are supposed to run across two or more databases and have been started, but neither finally committed nor rolled back at the time of the database backup. This option backs up only the most recent, committed transactions. It allows you to back up a database before recovering corrupted transactions. Generally, you should recover in limbo transactions before performing a backup.

(3) Backup Metadata only: results in an empty copy of the database, as only the database definition (metadata) is saved, not the data itself. This option is similar to using Windows ISQL to extract a database to a file.

(4) Garbage collection: checks every row, removing outdated versions, empty pages and parts of them.

Because each page is carefully examined, the backup takes longer. Should a backup need to be executed rapidly, the garbage collection can be switched off here. Only the deleted and *NOT* the older versions of updated data sets are dumped. The distribution of page occupation can be viewed in the database statistics. The garbage collection in InterBase/Firebird can also be started using the SELECT command.

(5) Old Metadata Description: this enables a backup and restore to older InterBase versions.

(6) Convert to Tables: this concerns so-called external files. Following a backup the external files are also incorporated, and then restored as tables.

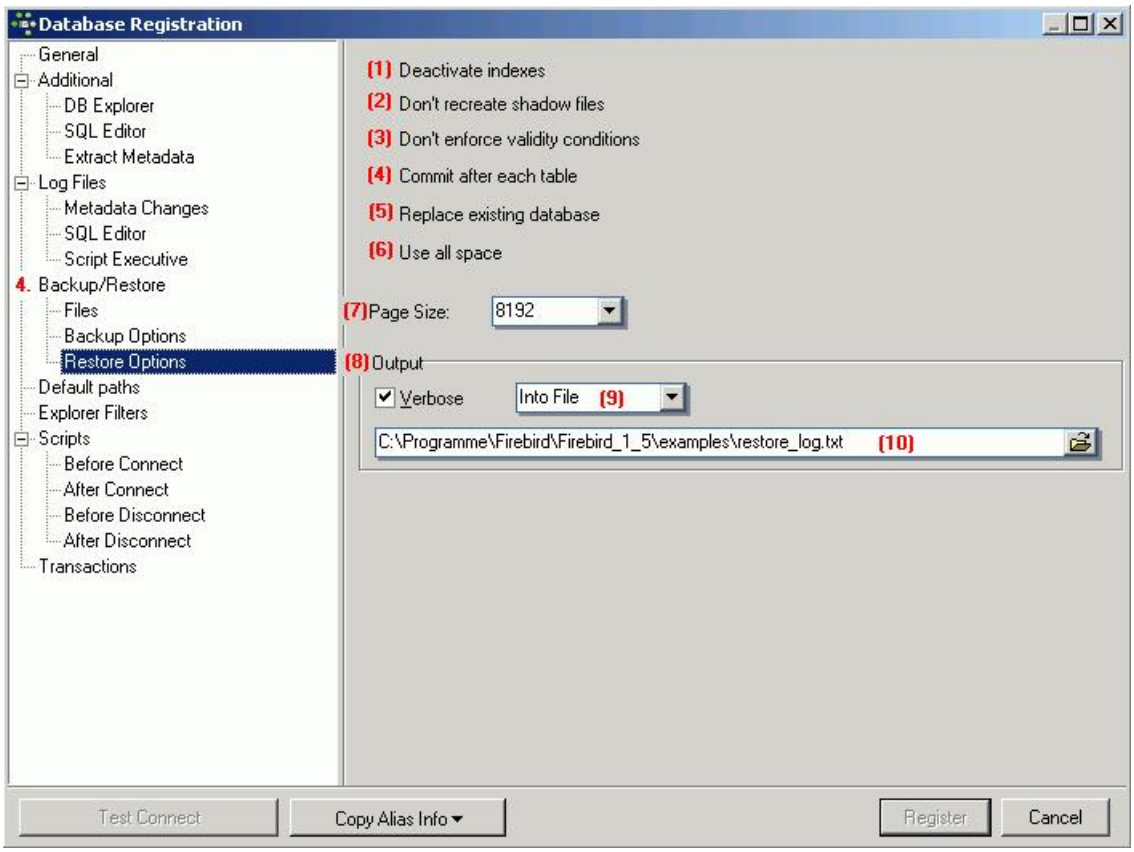
(7) Format: the options *transportable* or *non-transportable* are offered here. As a rule always choose "transportable", so that the database can be easily transported to other platforms such as Linux.

(8) Verbose Output: Writes step-by-step status information to the output log. This option is useful if the backup is failing, and the reasons need to be tracked down.

(9) The output log options: *on-screen* or *into file* are offered here.

(10) File name, path and drive; can be specified here, if the *into file* output option has been chosen.

Restore Options



(1) Deactivate indexes: This option does not restore indices as part of the restore process. It is used to improve restore performance. If this option is not checked, InterBase/Firebird updates indices after all tables have been filled with the restored rows. This option can also be used if duplicate values are suspected in indices that are flagged as unique. After the duplicate values have been found and corrected, the indices can be reactivated.

(2) Don"t recreate shadow files: this option deletes the database shadow definition. This option is required if the destination database does not support shadows, if you are migrating from an earlier version of InterBase where shadows were not supported, or if the machine where the shadow resides is not available.

(3) Don't enforce validity conditions: this option does not restore constraints, i.e. it deletes the validity constraints from the database's metadata definition. It is important to save a copy before a restore is performed with this option checked.

This option is necessary if the validity constraints were changed after data had already been entered into the database. When a database is restored, InterBase/Firebird compares each row with the metadata; an error message is received if incompatible data is found. Once the offending data has been corrected, the constraints can be added back.

(4) Commit after each table: this option restores metadata and data for each table in turn as a single transaction, and then commits the transaction. This option is recommended, so that should a problem occur during the restore, at least all correct tables are restored. It is particularly useful, if corrupt data is suspected in the backup, or if the backup is not running to completion. Normally, InterBase/Firebird first restores all metadata and then the data.

(5) Replace existing database: this should. as a rule, be toggled, as it makes no difference if there is no database present as yet. Although leaving this option unchecked provides a measure of protection from accidentally overwriting an existing database file that may still be needed.

(6) Use all space: only relevant if restoring the database to a CD. In this case 100% space of each page is used, and not, as is usual, 80%.

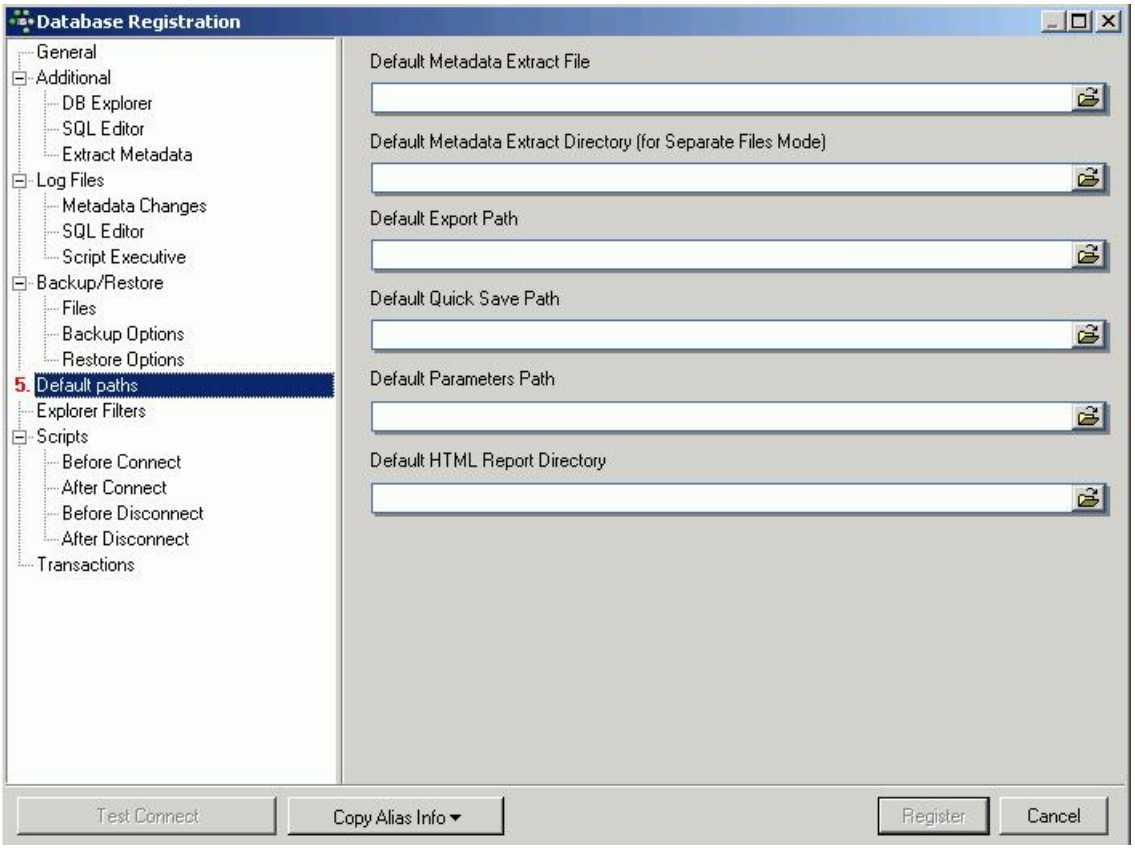
(7) Page size: Changes the default size of each page. There are numerous reasons for wanting to change the database page size (please refer to page size)
.

(8) Verbose Output: Writes step-by-step status information to the output log. This option is useful if the backup is failing, and you need to track down the reason.

(9) The output log options: *on-screen* or *into file* are offered here.

(10) File name, path and drive: can be specified here, if the *into file* output option has been chosen.

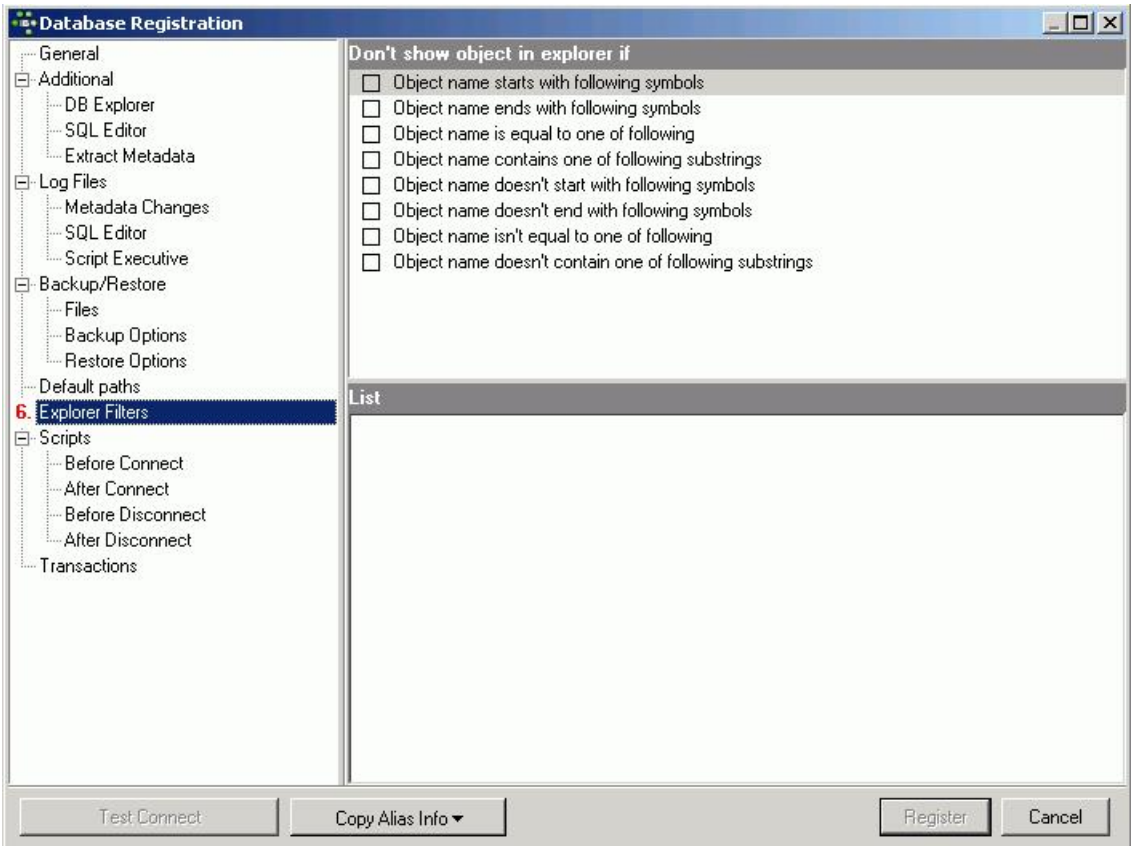
Default paths



Here standard default drives, paths and files may be specified, if wished, for the following:

- Metadata Extract File
- Metadata Extract Directory (for Separate Files Mode)
- Export Path
- Quick Save Path
- Parameters Path
- HTML Report Directory

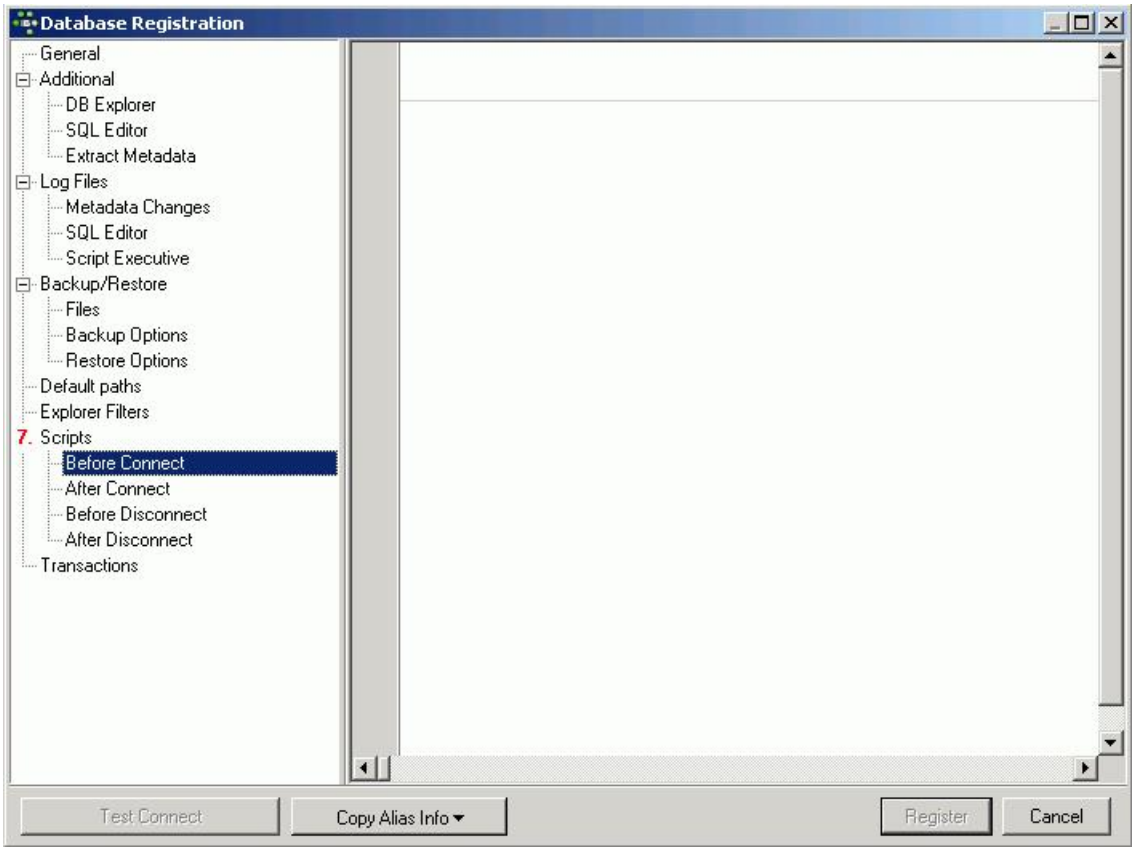
Explorer Filters



This is only of interest for extremely large and complex databases with multiple registrations. It refines the selection of database objects displayed in the IBExpert DB Explorer. The database object names displayed can be filtered according to one or more of the conditions listed.

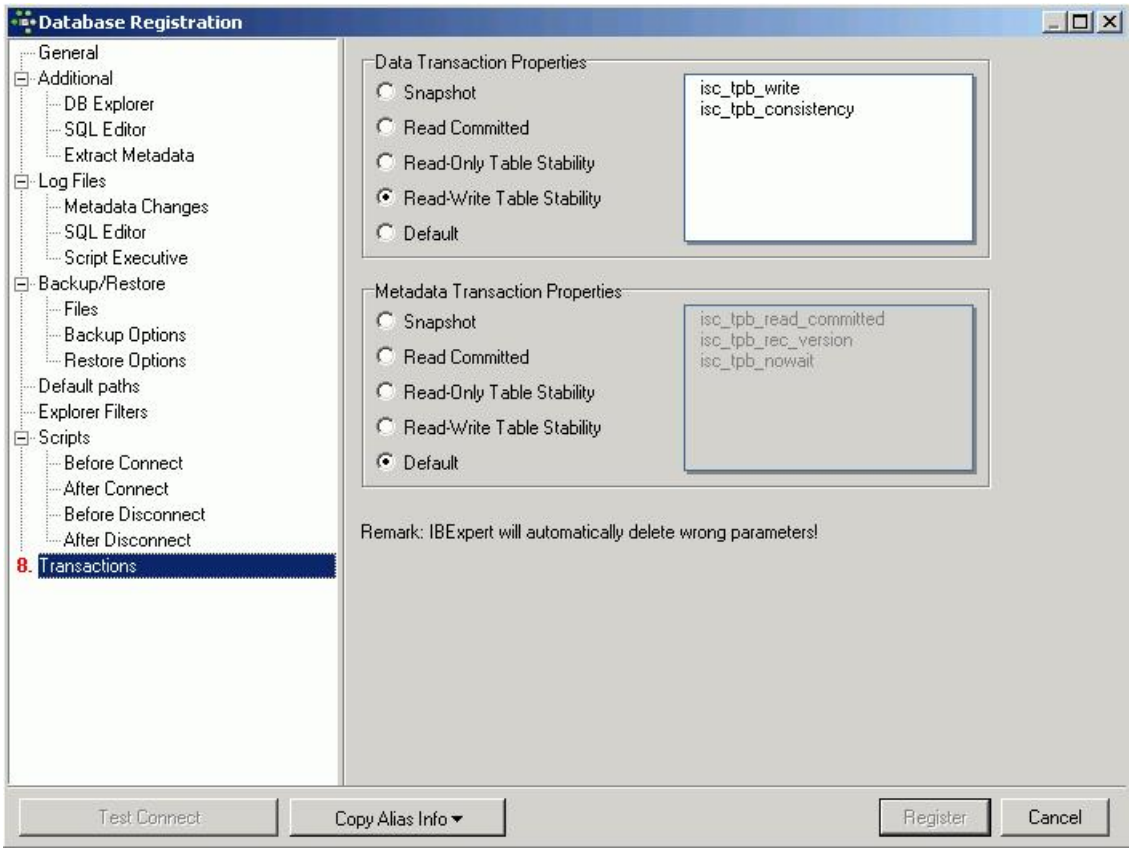
Scripts

Since IBExpert version 2003.12.18.1 there is the added possibility to execute SQL scripts before and after connecting to the database and before and after disconnecting from the database:



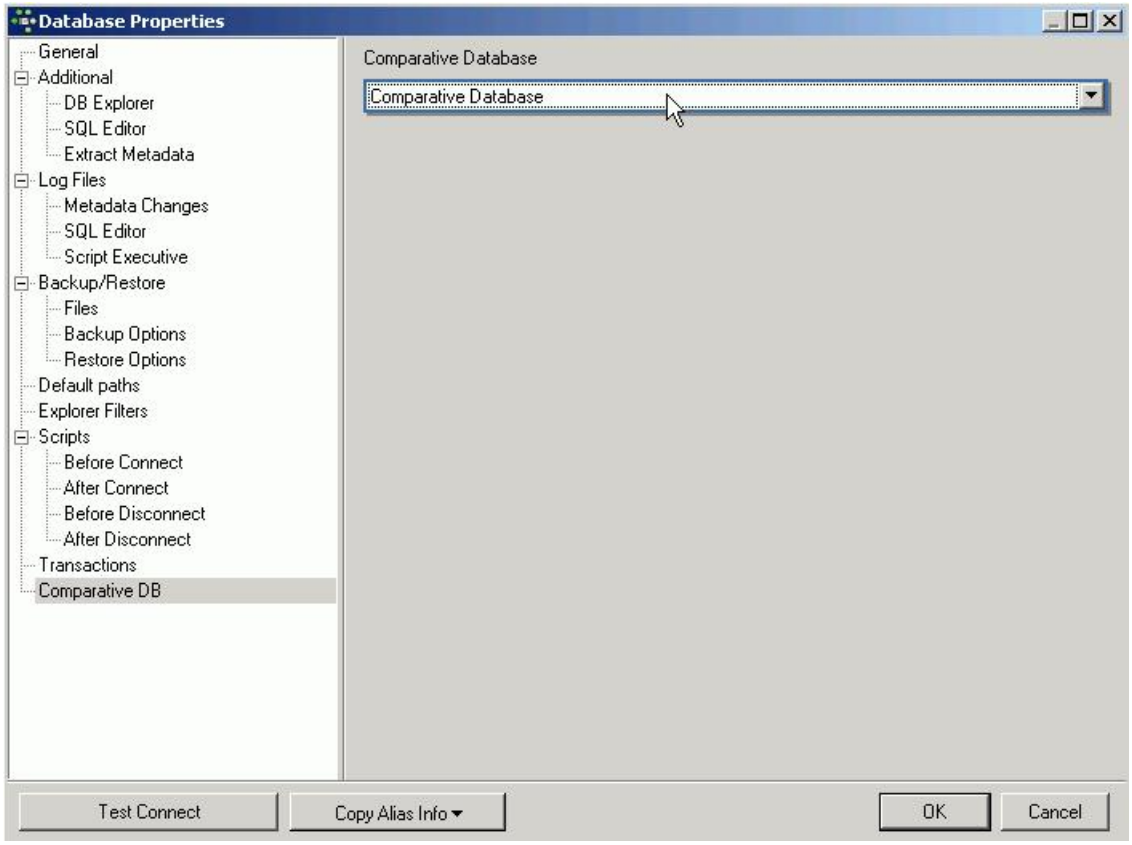
Transactions

New to IBExpert version 2005.06.07: this page allows you to specify different transaction isolation levels for registered databases.



Comparative Database

This option was introduced in IBExpert version 2006.03.06 and allows you to compare an object with one in another (comparative) database.



See also:
Database toolbar

Unregister Database

It may be desirable to unregister one or more databases in IBExpert, for example when a remote link to a customer database will never be needed again. Unregistering a database does not delete the database; it merely deletes the registration necessary for working with IBExpert.

If you are unsure whether a registered database will ever be needed again, but are tired of having it displayed in the DB Explorer every time work is started, it is possible to blend out unconnected databases using the DB Explorer right-click menu item *Hide Disconnected Databases*.

A database can be unregistered using the IBExpert menu item Database / Unregister Database, the DB Explorer right-click menu, or the key combination [Shift + Alt + U].

IBExpert asks for confirmation:

before finally unregistering the database.

Alternatively you can use the IBExpert Database Explorer to unregister more than one database at a time (this feature was introduced in IBExpert version 2006.08.12).

See also:
Database toolbar

Connect to an existing Database

After starting IBExpert, you will see the Database Explorer on the left side. Before a database connection can be made, the database must be registered (please refer to Register Database).

A database connection can be made to a registered database simply by double-clicking on the database alias name, displayed in the DB Explorer. There are also a number of menu options: either using the IBExpert menu item Database / Connect to Database, or the following icon:

in the Database toolbar. Alternatively the DB Explorer right-click menu may be used, or the key combination [Shift + Ctrl + C].

Since version 2.5.0.47 it is possible to automatically connect to a database when starting IBExpert. Use the following menu: Database Registration Info / Additional / and check: *Open database when IBExpert starts*.

Should there be any problems connecting to the database, use the IBExpert Services menu item Communication Diagnostics.

An example connecting to a remote database using the IBExpert Database menu item Database Registration Info:

> Server = `Remote`Server Name = `<network name of the server or its ip address> e.g. OUR_SERVER Protocol = TCP/IP`DB File Name = `<path to the db file on the server PC> e.g. "D:DataMyDB.fdb"`

Of course Firebird/InterBase should be installed properly on the server PC (where your database is placed) and the Firebird/InterBase client (`fbclient.dll` or `gds32.dll`) on your local PC.

Accessing a Firebird embedded database with Win1252 (or other character set)

This tip comes from Gerhard Knapp.

In order to connect to a Firebird embedded database with WIN1252 (or other character set) using IBExpert:

1. Rename `fbembed.dll` to `fbclient.dll` (always recommendable; not just in this case!).
2. Define this `fbclient.dll` including drive and path in the IBExpert Database Registration.
3. Specify WIN1252 in IBExpert.
4. Copy the subdirectory `intl` from the Program Files directory, where `fbclient.dll` is installed, into the directory `C:Program FilesHK-SoftwareIBExpert 2.0`!!

You should then have no further access problems.

Further information:

When `fbembed.dll` is renamed `fbclient.dll`, it is also a fully-fledged client, i.e. if an application needs to access an embedded database on a Firebird server, the `fbclient.dll` is more than sufficient.

Database login

If a password is not entered at the time of registering the database (see Register Database), it needs to be logged into each time the database is opened.

Specify a username and associated password. If the user is not authorized or the password is not correct, an error message appears.

Optionally, a role may be specified. If the role has previously been `GRANT`ed to the username, all access privileges assigned to that role for the duration of the current session apply for that user.

If the user is an authorized user for that server, and if the password is correct, access is granted to the database.

Remote database connect using an alias

This article was written by Claudio Valderrama (http://www.cvalde.net/ - The InterBase Unofficial Site), February 2002

Many developers wish to avoid the client having to give the engine the full path of the database in the same machine (node) where the engine runs? It is not only inconvenient when the database's location is changed, it is also a low level that the client shouldn*t be concerned about. Finally, many developers have concerns with the security. Ideally, the physical location of the engine and the databases shouldn*t be disclosed to the client. Only an Alias should be visible.

It's incredible that for years, a built-in solution in the engine (that works whenever the server is a NT machine) has been lying in the heart of the code and nobody made it public, less even documented in some help file. Perhaps because it unfortunately is a Win32 only solution, nothing that can be used on Linux, so the location of a `gdb` is not truly transparent.

The syntax is very simple. It has the form:

```
\server!share_name!database.gdb@@
```

or the form

```
server:!share_name!database.gdb
```

It's not a true alias, since you still know the name of the database and of course, the server machine should be known. But it helps if you need to move the database around NT servers, without having to change configuration files or recompiling programs. Here, "server" is the NetBEUI name of the NT machine, followed by the pseudo-UNC paths that IB/FB uses. Alternatively, "server" is the TCP/IP name of the NT machine, but followed by backslashes, not the typical slashes the IB*s TCP syntax uses. (Really, using slashes or backslashes is not important in a typical full path, since the engine makes the adjustments, but in this case, the syntax to recognize the share demands backslashes.) The difference is that instead of a full path inside the server, a share*s name in the server is used, surrounded by exclamation marks.

This share points in turn to the full path of the database, so you only have to append the database's name. It has nothing do to with client-side mappings.

How it works: the client library recognizes a UNC-like path and knows it's NetBEUI. Otherwise, it recognizes a TCP-like syntax thanks to the colon. Then it connects to the required server with the right network protocol and passes the remnant of the path, stripping the server's name. A routine inside the engine, named `expand_share_name`, will look for the backslash followed by the exclamation mark, then if a matching "`!`" occurs, it takes the name inside the two pairs ("`!`" and "`!`") and will open the registry (`RegOpenKeyEx`) at

```
SYSTEM\CurrentControlSet\Services\LanmanServer\Shares
```

to extract the data (`RegQueryValueEx`) in the value `<share_name>`, that's supposedly the name of a registered share in the server machine. It proceeds to decode the data and gets the "`Path`" component inside the multi-string data that*s the physical path. It loads this path in its argument and returns to the caller that will continue testing to see finally if the database*s name is valid and exists.

For example, given a share"s name "`myshare`", the registry key shown above contains a list of values that denote shares. You can find there the implicit ones such as `IAS1$` (very bad, get rid of it since it points to the IIS admin dir), the `NETLOGON` share and "`myshare`". Reading the data in the value "`myshare`", the following can be seen:

```
MaxUses=4294967295.Path=H:PROY.Permissions=127.Remark=for fb.Type=0..
```

The dots denote the `NULL ASCII` value, since this is a multi-string. The engine looks for "`path`" and gets the string that follows, namely `H:PROY`, then appends the backlash if missing. Hence, the engine uses information in the server itself to decode the full path. This path will prefix the database name when the function `expand_share_name` returns to the caller.

An advantage is that you don*t need to grant permissions on this share. You can deny anyone any right (even if NT prompts if you are sure) and you can go further: you can stop the service responsible for handling requests of NetBEUI shares. The engine reads the registry directly, so it doesn*t query the network layer. It's a true hack, a commodity to avoid the inclusion of hard-coded paths in the client. If you want to change it, just change the share"s information, without granting anyone any right on the share. Since the engine reads that registry location each time a connection string should be analyzed, it will get the changed name in the next attachment request. If you disabled some networks services, so that changing the share is not possible through high level interfaces, you can edit the registry directly and change the path. Beware that the each dot represents a `NULL ASCII` value in the example shown above, so your path should end with that value. An even nicer feature is that this works:

```
H:ibdevfbbuildinterbasejrd>isql \atenea!myshare!g
Database: \atenea!myshare!g
SQL> ^Z
```

but it's not restricted to NetBEUI. Indeed, as noted before, you can use TCP syntax:

```
H:ibdevfbbuildinterbasejrd>isql localhost:!myshare!g
Database: localhost:!myshare!g
SQL> ^Z
```

(Remember that there's no restriction to the name of a `gdb` other than the file name conventions in the platform where the engine resides. In this case, it's simply named "`g`", although an extension helps the database admin.)

There are a couple of drawbacks: first, this hack is tied to Win32. (Furthermore, I don't have a way to test it on XP, but I"ve been informed of success with Windows 2000.) Second, when I read that internal function `expand_share_name()`, I found a possible buffer overrun and closed it. Revisiting the code when I

wrote this article, I found a registry key handle that wasn't closed if the function gives up prematurely for lack of RAM. (I solved this second glitch in Firebird at the time I was finishing this article.)

Hence, I believe the lack of documentation comes from the untested nature of the facility.

See also:
Database toolbar
Communication Diagnostics

Reconnect to Database

This menu item is useful should a database connection have accidentally been disconnected (this may happen sometimes with a remote connection).

The reconnection can be simply made either using the Menu Database / Reconnect Database, or the following icon:

in the Database toolbar. Alternatively the DB Explorer right-click menu may be used.

Should there be any problems reconnecting to the database, go to the Database Registration Info and perform a Test Connect.

Disconnect from a Database

When you have finished working with a database it can be disconnected using the IBExpert menu item Database / Disconnect from Database, or the following icon:

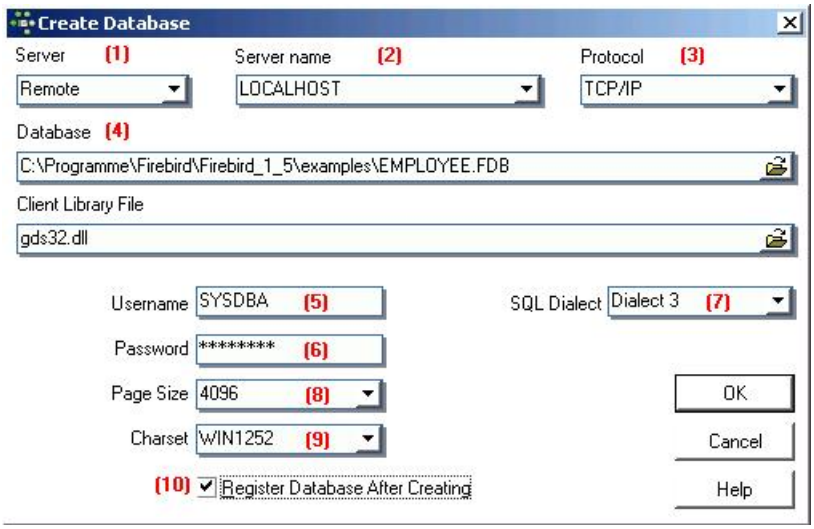
in the Database toolbar. Alternatively the DB Explorer right-click menu may be used, or the key combination [Shift + Ctrl + D].

It is not necessary to disconnect all databases manually when you have finished working with IBExpert. IBExpert does this automatically when it closes down.

See also:
Database toolbar
Exit

Create Database

A new database can be created by simply using the IBExpert menu item Database / Create Database... or using the respective icon in the Database toolbar. The *Create Database* dialog appears:



(1) Server: first the server which is to store the database needs to be specified. This can be local or remote.

- **Remote:** the remote connection needs to be defined by specifying **(2)** Server name and **(3)** Protocol. The pull-down list shows all servers previously connected to/from this workstation/PC.
- **Local:** `LOCALHOST` (own Server). To create a new database on the same machine where IBExpert is in use, you do not need to enter a server name.

We recommend always referencing a server, even if your database is sitting locally on your machine. Going directly using the local specification can cause problems (refer to **(3) Protocol** below), particularly with Windows Vista, so always use the *Remote* and *`LOCALHOST`* options.

The DOS `PING LOCAL HOST` or `PING SRVNAME` command shows the path if unknown (it is not necessary to know which operating system is running or where this server is). By specifying a local server, fields **(2)** and **(3)** are automatically blended out, as they are in this case irrelevant.

(2) Server name: must be known when accessing remotely. The following syntax should be used:

- Windows `SERVER_NAME:C:pathdatabase.gdb`
- Linux `SERVER_NAME:/path/database.gdb`

The standard port for InterBase and Firebird is `3050`. However this is sometimes altered for obvious reasons of security, or when other databases/Firebird versions are already using this port. If a different port is to be used for the InterBase/Firebird connection, the port number needs to be included as part of the server name. For example, if port number `3055` is to be used, the server name is `SERVER/3055`. If you use multiple Firebird versions and have a database, `db1`, sitting locally on `C:\` root using the Firebird version on port 3052 (which has been specified in the `firebird.config`), the database connection path would be:

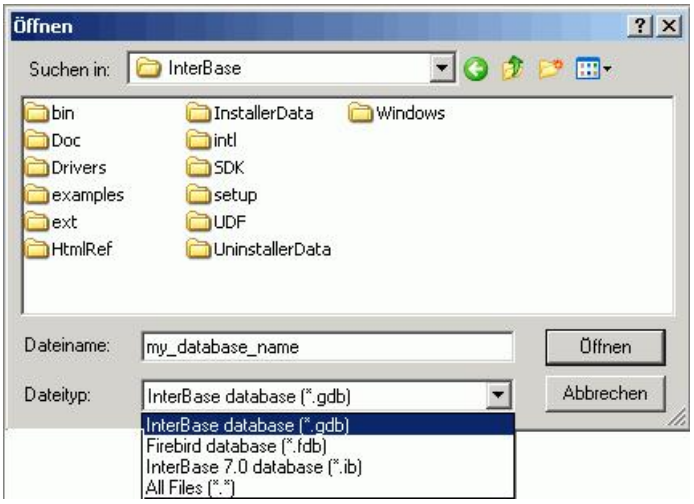
```
localhost3052:C:\db1.fdb
```

(3) Protocol: a pull-down list of three options: TCP/IP, NetBEUI or SPX. As a rule we recommend you always use TCP/IP (worldwide standard).

- SPX used to be used by Novell; now even Novell supports TCP/IP.
- NetBEUI - is not really a network protocol, it simply accesses the line. It is slow as it makes everything available everywhere and anyone can access the information. This is also purely a Windows protocol. *Note:* in DOS the `TRACERT` command lists the protocol route. TCP/IP intelligently takes another direction, if one or part of the lines on the quickest route is blocked or down.

As the local protocol should only be used if really necessary on machines that are isolated and not part of any network, specify the database server connection if possible using *Remote* and *`LOCALHOST`* and selecting one of the above protocols.

(4) Database: by clicking on the folder icon to the right of this field, the path can easily found and specified, the database name entered, and the suffix selected from the pull-down list. The database name must always be specified with the drive and path when creating a database. Please note that the database file for a Windows server must be on a physical drive on the server, because InterBase/Firebird does not support databases on mapped drive letters. The database suffixes do not have to adhere to the forms offered in the list.



(5) User Name: Only those names may be entered when creating a database, which already exist in the server security database `ISC4.GDB`, `security.fdb` or since Firebird 2.0 the new `security2.fdb` (which stores server rights; user rights for the database objects are stored in the database itself). The person creating the database becomes the database owner. Only the database owner and the `SYSDBA` (System Database Administrator) are allowed to perform certain operations upon the database (such as a database shutdown). Therefore if the database owner is defined as the `SYSDBA`, this is the only person entitled to perform these operations. *Note:* when a role with the name `SYSDBA` is created, no other users (not even the `SYSDBA`) can access the database. Therefore ensure the database is created by another user already registered in the security database and not the `SYSDBA`. This way there are at least two users able to perform key administrative tasks.

(6) Password: The passwords are encrypted in the `ISC4.GDB`. If you insist upon using the `SYSDBA` name as the database owner, at least change the standard password (`masterkey`) to ensure at least some degree of security! The `masterkey` password should be changed as soon as possible after creating the database.

InterBase verifies only the first 8 characters of a password, even if a longer word is entered, i.e. in the case of the masterkey password only "`masterke`" is verified. All characters following the 8th are ignored.

(7) SQL Dialect: Here Dialect 1 (up to and including InterBase 5) or 3 (InterBase 6/Firebird) needs to be specified. For more information regarding this subject, please refer to SQL Dialect.

(8) Page size: Specifies the database page size in bytes. For more information regarding this subject, please refer to *Page Size*.

(9) Charset: Here the default character set can be defined for the database. (A default character set can be specified as default for all new databases in the IBExpert Options menu item, Environment Options, under *Default character set*.) This character set is useful, when the database created is to be used for foreign languages as it is applicable for all areas of the database unless overridden by the domain or field definition. If not specified, the parameter defaults to `NONE`, i.e. values are stored exactly as typed. For more information regarding this subject, please refer to Charset/Default Character Set.

(10) Register Database After Creating: This checkbox automatically generates the *Database Registration* dialog so that the database can be registered. Registration is necessary, so that IBExpert recognizes that a database is present. The Register Database dialog however offers many further options. We recommend clicking this checkbox (the default setting), so that the database is registered immediately after creation. If the database is not registered at the time of creation, it cannot be seen in the DB Explorer of the left of the IBExpert screen. This means that the user must know exactly where the new database can be found (i.e. which server, path, name etc.) when registering at a later date.

Tip: IBExpert recommends creating a User Database - please refer to Environment Options / IBExpert User Database for further information.

For those preferring SQL, the syntax is as follows:

```
CREATE {DATABASE | SCHEMA} 'filespec'
[USER 'username' [PASSWORD 'password']]
[PAGE_SIZE ] int]
[LENGTH [ int [PAGE[S]]]
[DEFAULT CHARACTER SET charset]
[secondary_file];
<secondary_file> = FILE 'filespec' [fileinfo] [secondary_file]
<fileinfo> = [LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int }
[fileinfo]
```

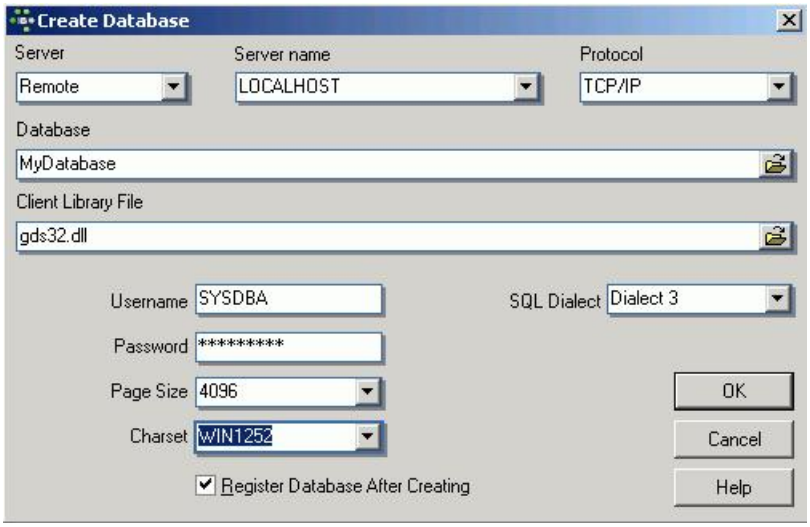
For example:

```
CREATE DATABASE 'C:DATABASEFILESemployee.gdb'
DEFAULT CHARACTER SET ISO8859_1
FILE 'employee2.gdb' STARTING AT PAGE 10001;
```

Charset / Default Character Set

The default character set is the character set defined when creating the database, and applicable for all areas of the database unless overridden by the domain or field definition. It controls not only the available characters that can be stored and displayed, but also the collation order. If not specified, the parameter defaults to `NONE`, i.e. values are stored exactly as typed.

InterBase/Firebird supports multiple character sets for use around the world. If no special character set is specified for individual columns, the database default character set is assumed. The default character set is defined in IBExpert in the *Create Database* dialog:



If a character set is defined as the default character set when creating the database, it is not necessary to define this again for individual columns.

InterBase/Firebird supports more that 20 different character sets directly. The chosen character set is also of importance when importing and exporting data with different character sets. This needs to be taken into consideration when applications are developed with multiple language versions.

The ASCII character set is not synonymous with a non-defined character set. If no character set is defined, Firebird/InterBase chooses the character set `NONE`. The character set `NONE` does not translate characters. Umlauts and accents are not sorted correctly. When the `ASCII` character set is specified, all characters are translated into the ASCII equivalents from the character set under which they were input.

The character set `WIN 1252` is recommended for European countries, as it includes all characters and collation orders of the most important European languages.

Generally this default character set cannot be altered at a later date (only using the command line tools IBExtract and IBEScript). Alternate character sets can however be defined for individual domains and tables, which override the default character set.

For more information about character sets, please refer to: Charset / Character Set, Overview of the main character sets and Declaring character sets in XML and HTML (IANA charset definitions).

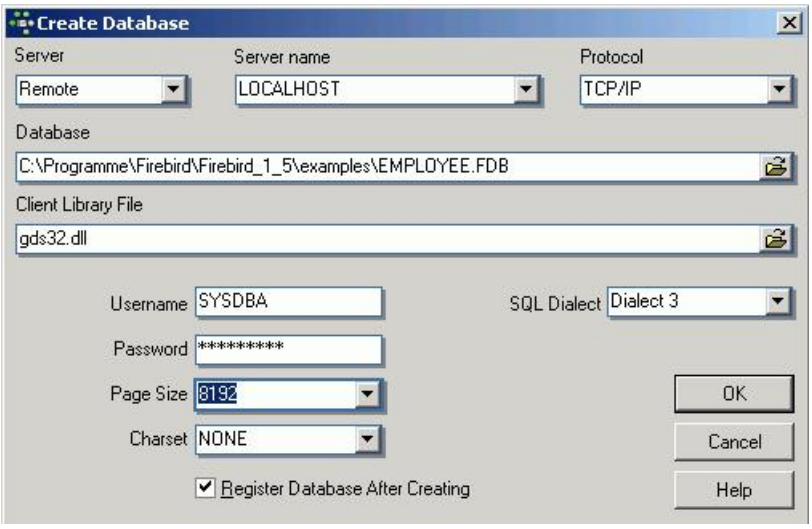
See also:
`SET NAMES`
Character Set

Page size

This is the specification of the database page size in bytes.

Firebird/InterBase databases are saved in blocks. Each of these blocks is called a page. A database page is the smallest adminstrative unit in the database file. Database administration occurs basically by accessing the hard drive block by block. The more data per access fetched by a single database page, the less often it is necessary to load a new page, at least theoretically. Practically, depending upon the operating system and server hardware, access to larger database pages can even influence the performance negatively, as 1024 bytes can be loaded quicker than 8192 bytes.

Page sizes permitted are 1024, 2048, 4096, 8192 and 16384. Up to and including Firebird version 1.5 page sizes up to 8192 should be used. The current largest page size of 16384 should be reserved for Firebird 2.0 and higher.



A large page size has certain advantages in the following situations:

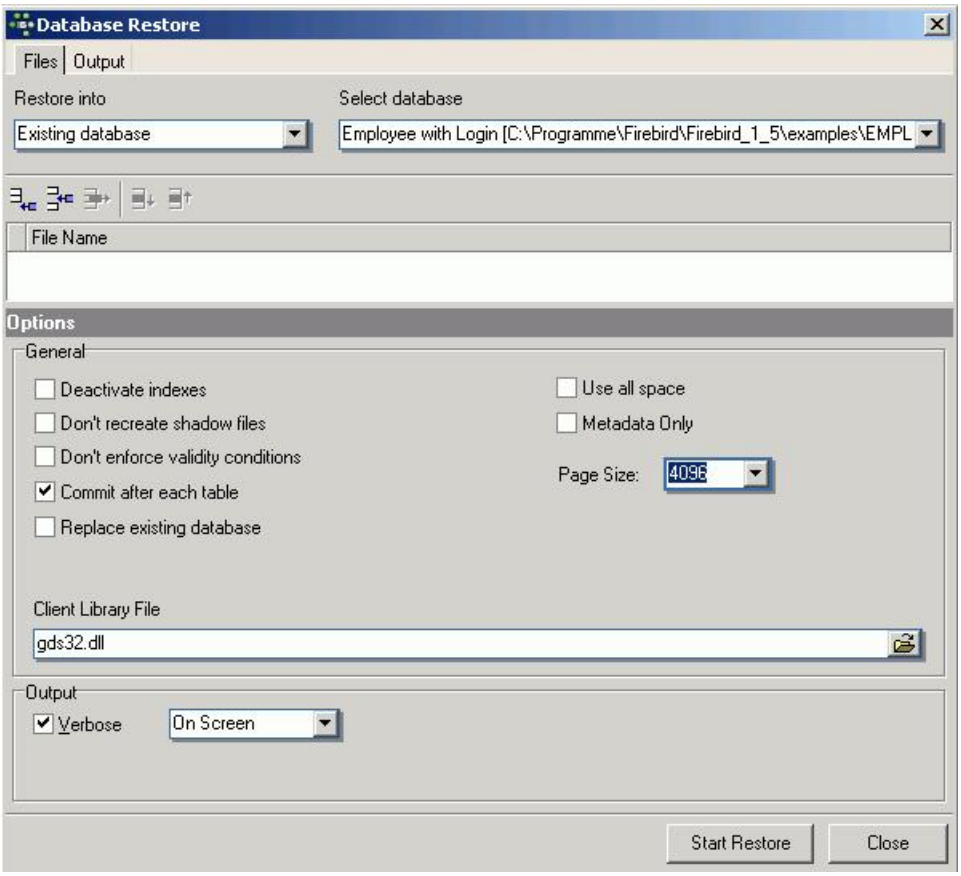
1. Many index-based operations (indices work quicker if the index depth is minimized).
2. Wide records, because with very wide data structures, i.e. with very many and/or very long columns, reading a data set is more efficient. With data sets that do not fit onto one page, several pages have to be read to fetch a single data set. The same applies to writing; ie. fetches across several pages are necessary.
3. Large blob fields, as data is stored and retrieved more efficiently if fewer pages need to be fetched. With larger blobs the writing and reading processes are also more effective, as, for example, 100 accesses are necessary for a 100K blob column with a 1K page size. However with an 8K page size only 13 accesses are required.

A small page size is sufficient if many transactions return only a small number of rows. Slim table structures with small database pages can be accessed more quickly for reading and writing as less memory is required, and more database pages can be held in the cache. However a database with a page size less than 4096 is not recommended on Windows, as this is the Windows block size. Therefore smaller page sizes do not bring any advantages, as Windows will still fetch 4K blocks.

The database page size has a direct influence on the amount of database cache, which influences all of the above points. If a 16 KB page size is specified and the Firebird server's database cache defined in the `firebird.conf` at its maximum of 128,000 pages, a total of 2 GB cache is made available for holding data pages. The same cache specification with a page size of 1 KB only provides 180 MB cache. Please refer to *Memory configuration* for details of cache specification for the Firebird SuperServer and Classic server.

Although you may be wasting a certain amount of space with a large page size, at today's hardware prices this should not be a serious problem, and it can offer more performance advantages.

The only way to subsequently alter a database page size, is to perform a database backup followed by a restore (IBExpert menu item, Services / Restore Database) where the database page size can be redefined.

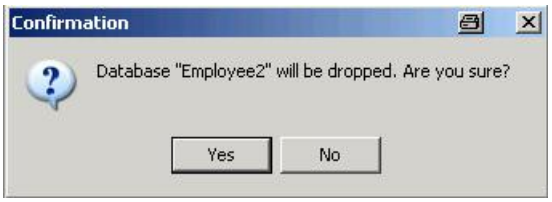


See also:
CREATE statement
Register Database
Database Designer
Memory configuration

Drop Database

Databases can be dropped in IBExpert using the menu item Database / Drop Database. When an InterBase/Firebird database is dropped, all the metadata and data for this database are also deleted, along with all its secondary, shadow and log files ...permanently!

IBExpert asks for confirmation:



before finally dropping the database. Once dropped, it cannot be retrieved, so be extremely careful when using this command.

For those users preferring direct SQL input, the syntax is:

```
DROP DATABASE;
```

A database may only be dropped by its creator or the SYSDBA.

See also:
DROP statement

Recreate Database

This new IBExpert menu item, Recreate Database was introduced in IBExpert version 2004.9.12.1. This drops the database, along with all its contents, and creates it again without the metadata and data content (after confirmation, of course) using the parameters of the database just dropped. The parameters are:

```
server name, protocol, user name, password, page size, SQL dialect, default character set
```

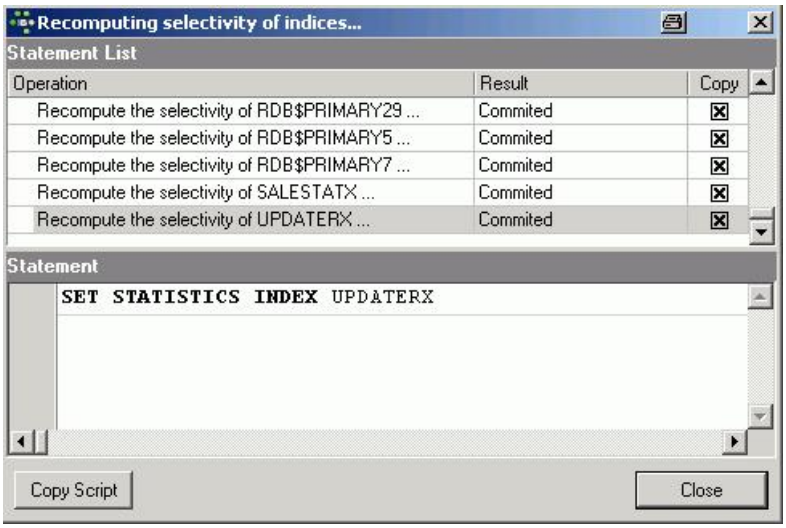
See also:
Drop Database
Create Database

Recompute selectivity of all indices

Indices statistics are used by the InterBase/Firebird Optimizer, to determine which index is the most efficient. All statistics are recalculated only when a database is restored after backing up, or when this is explicitly requested by the developer.

When an index is initially created, its statistical value is 0. Therefore it is extremely important, particularly with new databases where the first data sets are being entered, to regularly explicitly recompute the selectivity, so that the optimizer can recognize the most efficient indices. This is not so important with databases, where little data manipulation occurs, as the selectivity will change very little.

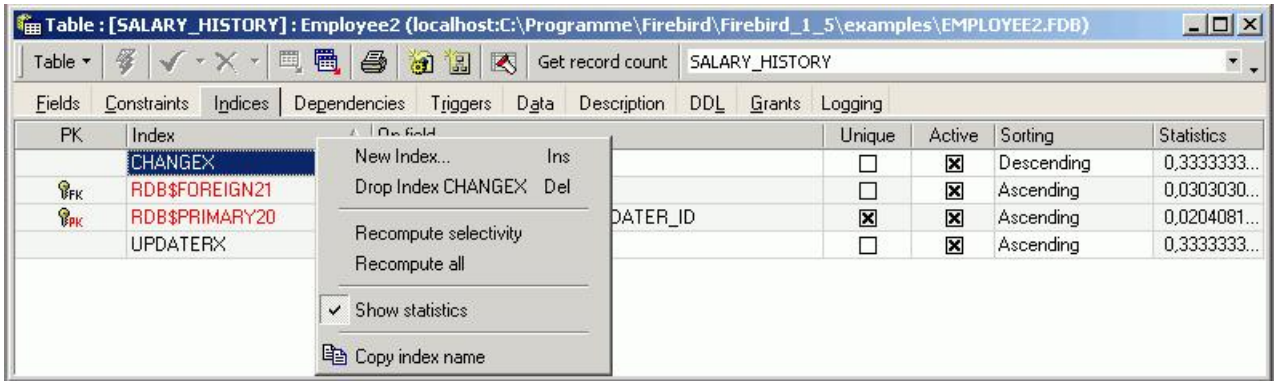
To recompute the selectivity of all indices use the IBExpert menu item *Recompute Selectivity of all Indices*. This can be found in the IBExpert Database menu or using the right mouse button in the DB Explorer.



Individual indices can be recomputed directly in the SQL Editor using the command:

```
SET STATISTICS INDEX <index_name>;
```

Single or multiple indices can also be recomputed directly in the Table Editor / Indices page, using the right-click menu.



The same *Recomputing Selectivity* dialog as above is then displayed.

The new statistical values can be viewed for individual tables in the Table Editor on the Indices page (providing the statistics are blended in using the right-click menu item *ShowStatistics*).

See also:
Index SQL Editor / Plan Analyzer
SQL Editor / Performance Analysis
Database Statistics / Indices
Firebird for the database expert: Episode 1 - Indexes
Firebird 2.0.4 Release Notes: Enhancements to indexing

Recompile all stored procedures and triggers

Stored procedures and triggers use indices internally. The *Recompile* command ensures that the most up-to-date indices are used. Using this command it is also possible to recognize when one procedure or trigger calls another.

This is also useful, for example, when backing up an older InterBase version (e.g. v5) and restoring in a newer version, such as InterBase 6 or Firebird 1.5, as InterBase/Firebird simply copies the data and metadata into the new version when restoring. Unfortunately this means that if a variable name that is a keyword

in the stored procedure is wrong, it is not recognized, as the compiler does not recognize variable names as such. When however procedures and triggers are recompiled, any such problems are discovered.

The menu items, *Recompile all Stored Procedures and Recompile all Triggers* can be found in the IBExpert Database menu or using the right-click menu in the DB Explorer.

See also:
Firebird 2.0.4 Release Notes: Enhancements to indexing

Database security

Please refer to the following subjects, for further information regarding database security:

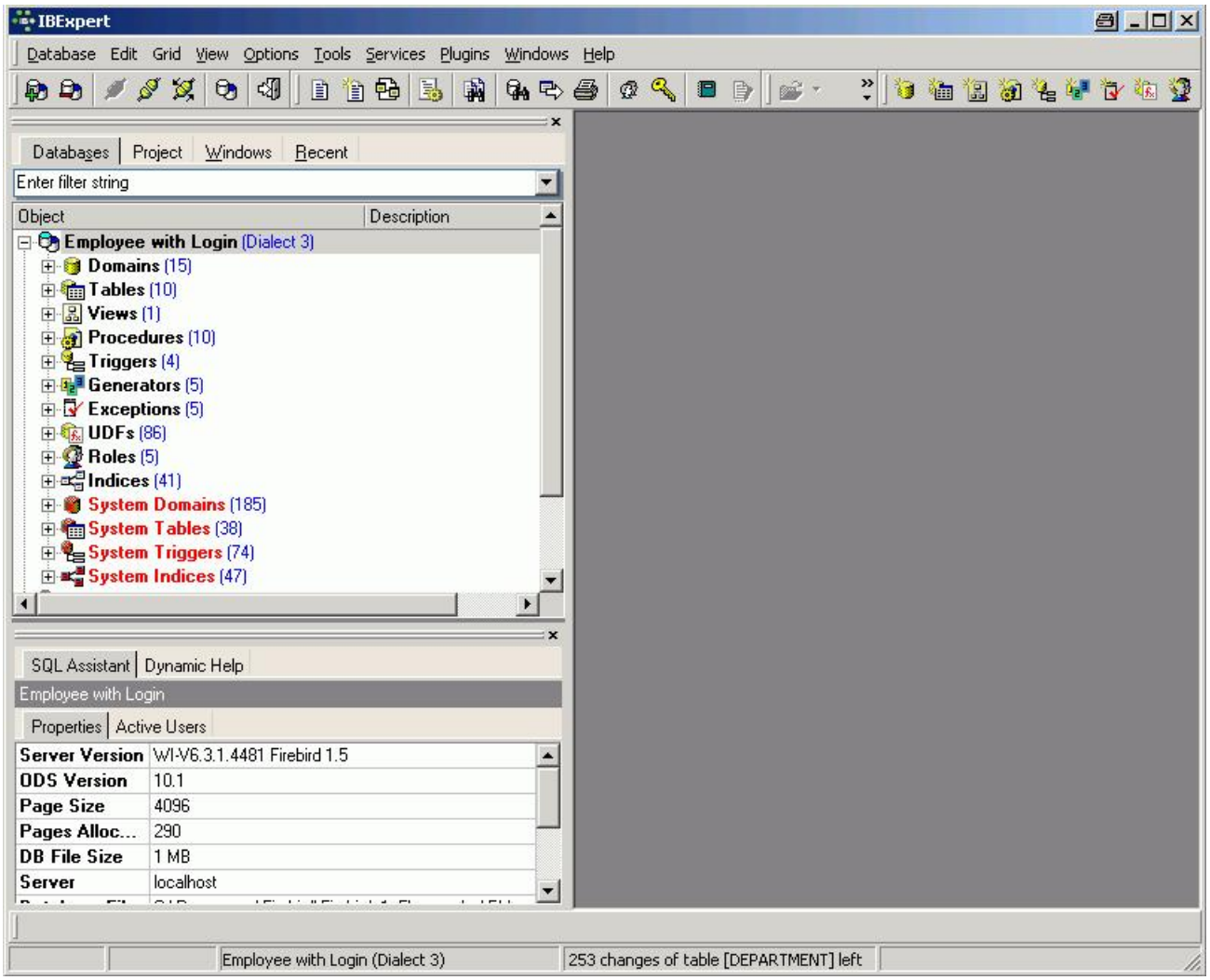
- User Manager
- Grant Manager

Database objects

InterBase/Firebird administrates the database data in database objects. These are the fundamental building blocks of the database and include the following:

- Domains
- Tables
- Generators
- Constraints
- Indices
- Views
- Triggers
- Stored Procedures
- Exceptions
- Blob Filters
- User-defined functions (UDFs)

The database objects can be viewed, created, edited and deleted using the IBExpert DB Explorer.



Alterations to database objects (online operation) are limited to 255 alterations per object (see status bar for more details). At this stage a backup and restore is necessary, in order to perform further alterations. This limitation is due to the fact that InterBase stores each data structure every time a record is inserted.

The IBExpert object editors all contain detailed dialogs for inserting, altering and dropping individual objects. The majority of editors display a number of tabs, comprising multiple input and display pages.

Certain typical windows recur in several object editors:

- **Dependencies:** all objects, which depend on other objects or where other objects are depending on this object, can be viewed on the object editor's Dependencies page.
- **DDL:** the SQL code, resulting from the user input, is displayed.
- **Performance Analysis:** for stored procedures and the SQL Editor, the result set can be started with [F9]. The performance result is displayed on a new page.
- **Description:** shows the description field from the InterBase/Firebird database. Since IBExpert version 2005.09.25 IBExpert will now use the `COMMENT ON` statement (Firebird 2) when updating object descriptions, if it is possible.
- **Grants:** this page allows user rights to be granted for the active object directly in the object editor dialog, without having to leave and start the Grant Manager each time a new object is created. It is even possible to switch to other objects (i.e. views, triggers, procedures and roles), without having to leave the editor.
- **Comparison:** was added in IBExpert version 2006.03.06 and allows you to compare an object with the one in another (comparative) database. The comparative database can be specified in Database Registration Info / Comparative Database.

- **To-Do:** this feature was introduced in IBExpert version 2007.12.01 and can be used to organize your database development. You can add ToDo items for each object in the database.

These pages are explained in more detail in the Table Editor (except Performance Analysis - details under SQL Editor / Performance Analysis).

Domain

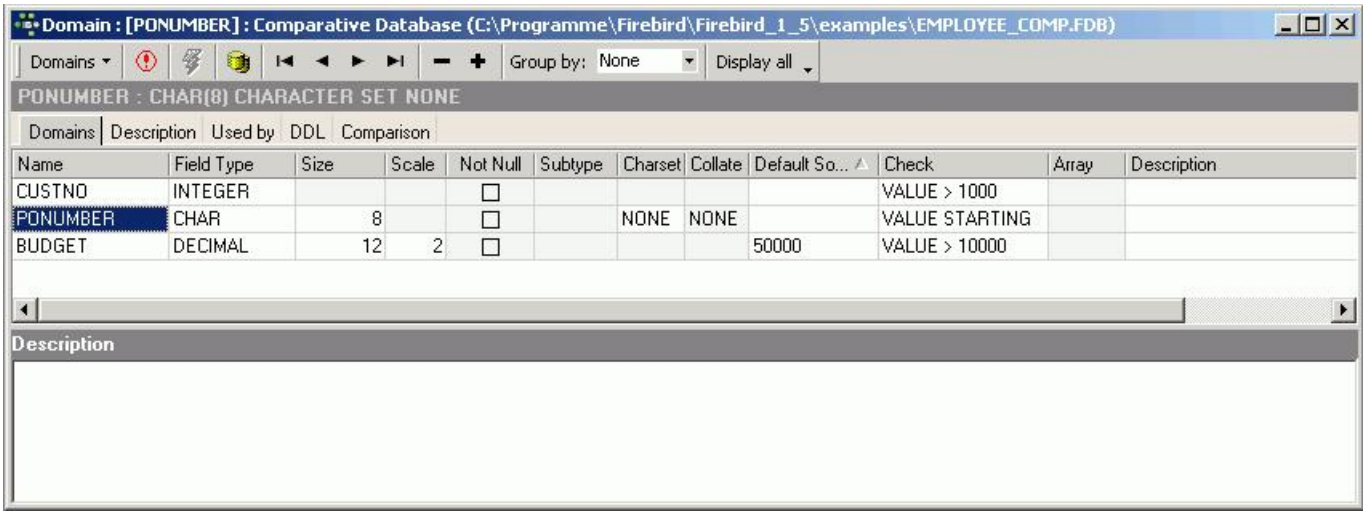
A domain is a user-defined custom datatype global to the database. It is used for defining the format and range of columns, upon which actual column definitions in tables may be based.

This is useful if columns in one or several database tables have the same properties, as it is much simpler to describe such a column type and its behavior as a domain. The columns can then simply be defined by specifying the domain name in the column definition. The column properties (e.g. field length, type, Not Null, constraints, arrays etc.) only need to be defined once in the domain. Domains help you create a uniform structure for your regular fields (e.g. ID, address and currency fields) and add more understanding to your database structure.

Certain attributes specified in the domain can be overwritten in the table field definition, i.e. a column can be based upon a domain; however small changes may still possibly be made for this column.

In addition to the datatype, a number of conditions and checks can be defined.

A domain is a database object and is part of the database's metadata, and can be created, modified and dropped as all other InterBase/Firebird objects in the IBExpert DB Explorer.



When developing a normalized database, the question arises in how far domains are necessary (multiple fields, multiple data etc.).

However, it does make life easier, should column alterations be necessary; e.g. zip code alteration from 4 to 5 digits (as was the case in Germany after the reunion), change of currency (e.g. from DM or Lire to Euro). In such cases, only the domain needs to be altered, and not each relevant column in each table individually throughout the database.

It should also be noted, that if user-defined domains are not explicitly defined and used for table column definitions, InterBase/Firebird generates a new domain for every single table column created! All domains are stored in the system table `RDB$FIELDS`.

Domain integrity

Domain integrity ensures that a column is kept within its allowable limits. This is achieved by keys and constraints.

New domain / Domain Editor

A new domain can be created for a connected database, either by using the menu item Database / New Domain, or using the DB Explorer right-click menu (or key combination [Ctrl + N], when the domain heading of the relevant connected database is highlighted), or the New Domain icon on the New Database Object toolbar.

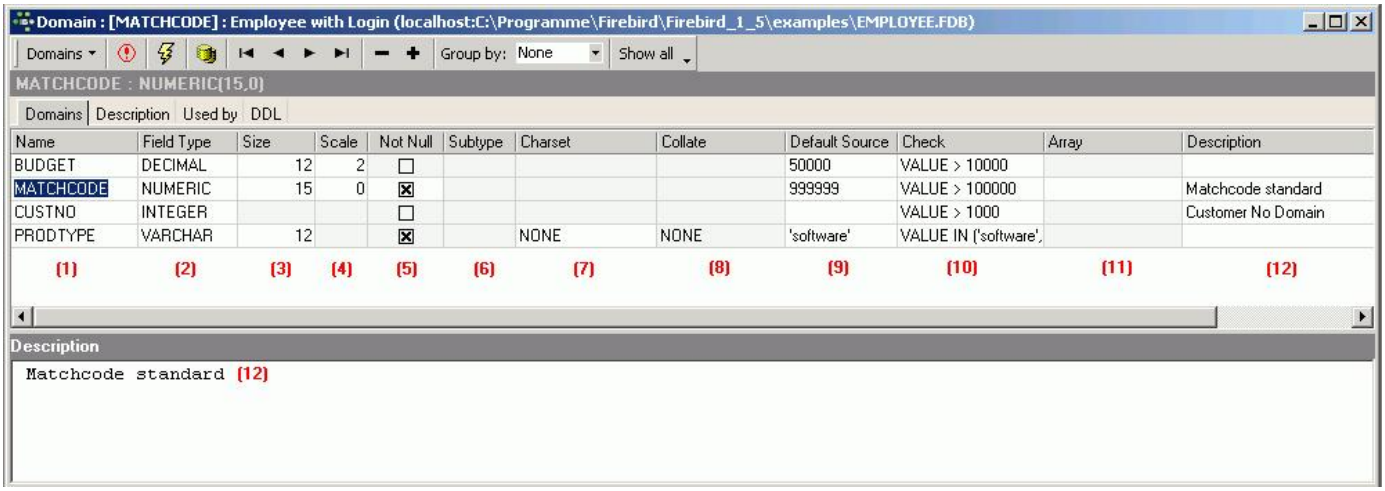
A *New Domain* dialog appears, with its own toolbar, and a pull-down menu (domain button). The toolbar offers the following options:

- Enable direct modifying of system tables
- Compile
- Duplicate the selected domain
- Navigational buttons
- Group by either Type or Charset
- Display all domains

For those users preferring to use the old IBExpert Modal Editor, check the *Use old-style Modal Editor* option in the IBExpert Options menu: Object Editor Options / Domains Editor.

A domain can also be created or selected and edited, when a new field is created or an existing field edited in a table, using IBExpert's Table Editor. (Please refer to *Insert Field* for further information).

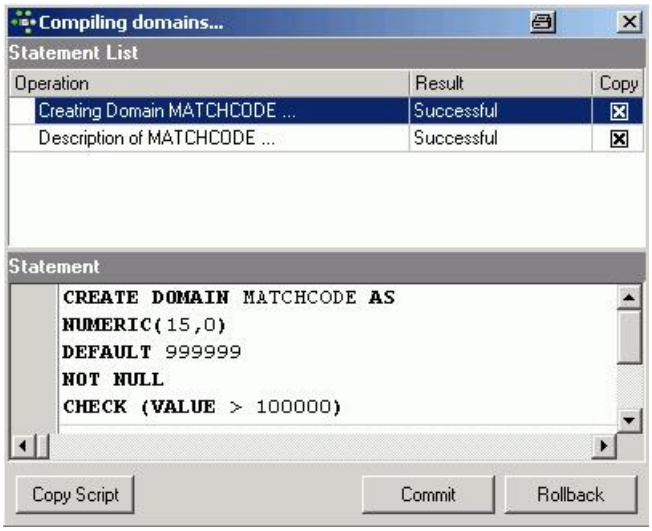
The following illustrates the creation of a new domain using the Domain Editor: initially a domain name is specified **(1)** in the first column on the first page Domains:



(Illustration displays the default Domain Editor.)

- **(2) Field Type:** Here the datatype can be specified.
- **(3) Size:** Specifies the field size.
- **(4) Scale:** Here the number of decimal places can be specified for all numerical fields.
- **(5) Not Null:** This check box can be marked by double-clicking or using the space bar. NOT NULL forces data to be entered in this field (i.e. the field may not be left empty).
- **(6) Subtype:** A subtype should be specified for blob fields.
- **(7) Charset:** A character set may be specified for individual domains. This overrides the database default character set. Although this is seldom used, it may be necessary should, for example, Asian, Russian or Arabic addresses need to be input and collated in a database with a European default character set.
- **(8) Collate:** Determines collation for a character set specified for a domain.
- **(9) Default Source:** Here a default data entry (text or numeric, depending upon the specified datatype) can be specified, e.g. the text NOT KNOWN can be specified as a default source, if an address field cannot be input by the user, because the information is unavailable.
- **(10) Check:** Each data set is examined for validity according to an expression defined in brackets. Certain conditions can be specified (see Check Constraint) causing an automatic database examination during data input, to ensure data consistency in the tables and among each other.
- **(11) Array:** Although arrays contradict all the rules of database normalization, there are certain situations (for example storing measurement data), when they are necessary.
- **(12) Description:** Useful for database documentation. The Description page should be used to describe the domain; the Description field for describing the field.

Several domains can be created simultaneously in the New Domain Editor. After creating the new domain(s), including all necessary parameters, don´t forget to compile (using [Ctrl + F9] or the respective icon):

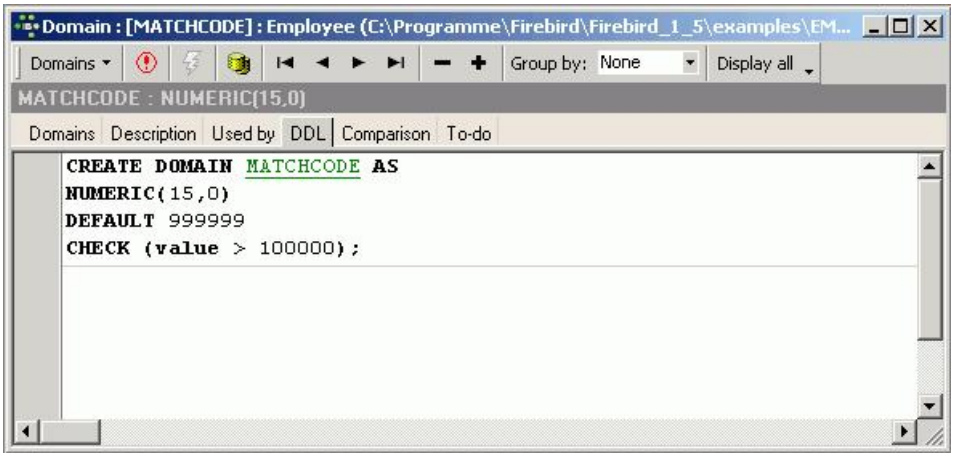


and finally committing, or should amendments be necessary, rolling back.

Tip: by clicking on the column headers (i.e. PK, FK, Field Name etc.), the fields can be sorted into ascending or descending order based upon that column. Double-clicking on the right edge of the column header adjusts the column width to the ideal width.

In addition to the Domains page, there are also Description, Used By, DDL, Comparison and To-Do pages:

- **Description:** this displays the description for the highlighted domain (i.e. the domain, where the cursor is currently standing).
- **Used By:** this displays those database objects which use or depend upon this domain.
- **DDL:** the DDL page displays the SQL statement created by IBExpert to create all specifications made by the user on the Domains page.
- **Comparison:** introduced in IBExpert version 2006.03.06, this feature allows you to compare a domain in the main database with a domain in a comparative database (for further information please refer to Comparison).
- **To-Do:** this feature was introduced in IBExpert version 2007.12.01 and can be used to organize your database development. You can add ToDo items for each object in the database.



Domains can also be created and edited directly from the New Field Editor (please refer to Insert Field).

Domains can, of course, also be created using DDL directly in the SQL Editor, using the following syntax:

```
CREATE DOMAIN domain_name [AS] <data_type>
[DEFAULT {expression | NULL | USER}]
[NOT NULL] [CHECK (<domain_such_expression>)]
[COLLATE collation];
```

For example:

```
CREATE DOMAIN MATCHCODE
  AS INTEGER
     DEFAULT  999999
     NOT NULL
     CHECK (VALUE > 100000);
```

Alter domain

A domain can be altered in the Domain Editor, opened by double-clicking on the domain name in the DB Explorer. Alternatively use the DB Explorer´s right mouse-click menu item Edit Domain or key combination [Ctrl + O].

`CHECK` instructions and default values may be added, altered or deleted. However it is not possible to alter the basic datatype (for example, from `NUMERIC` to `VARCHAR`). Neither is it possible to drop a `NOT NULL` constraint. To alter these the domain has to be dropped and recreated (see Drop Domain/Delete Domain).

Please note that if you want to change the `CHECK` constraint for a domain that already has a constraint defined, the existing constraint must first be dropped and then the new one added. `ADD CHECK` does not replace the current constraint with the new one. It is also important to realize that altering a `CHECK` constraint does not cause existing database rows to be revalidated; `CHECK` constraints are only validated when an `INSERT` or `UPDATE` is performed. One way of overcoming this limitation is to perform an `UPDATE` query using a dummy operation. If existing rows violate the new `CHECK` constraint, the query fails. These rows can then be extracted by performing a `SELECT`.

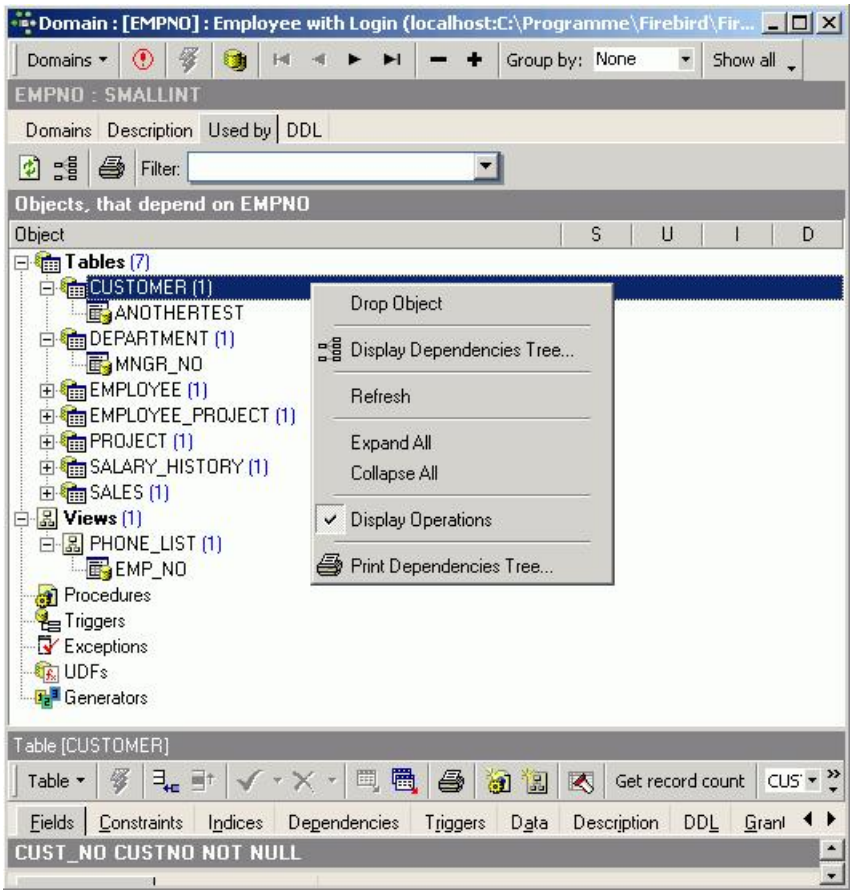
Any changes made apply immediately to all columns using the domain definition, unless, of course, the column´s (field) definition overrides these.

The SQL syntax for this command is:

```
ALTER DOMAIN <domain_name>
SET DEFAULT <default_value> | NULL | USER
  DROP DEFAULT
ADD CHECK <domain_search_condition>
  DROP CONSTRAINT;
```

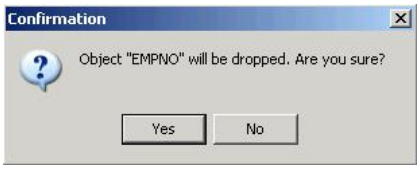
Drop domain/delete domain

A domain may only be dropped if it is not currently being used by any of the database tables. The Domain Editor´s Used By page shows which database objects use this domain. The dependent objects may also be directly dropped here, if wished, using the right-click menu on the selected object, and choosing the menu item Drop Object or [Ctrl + Del].



To drop a domain use the DB Explorer right-click and select the menu item Drop Domain or [Ctrl + Del].

Alternatively, a domain can be dropped directly from the Domain Editor using the pull-down menu Domains or the "-" icon in the Domain Editor toolbar. IBExpert asks for confirmation:



before finally dropping the domain. Once dropped it cannot be retrieved; the domain has to be recreated if a mistake has been made!

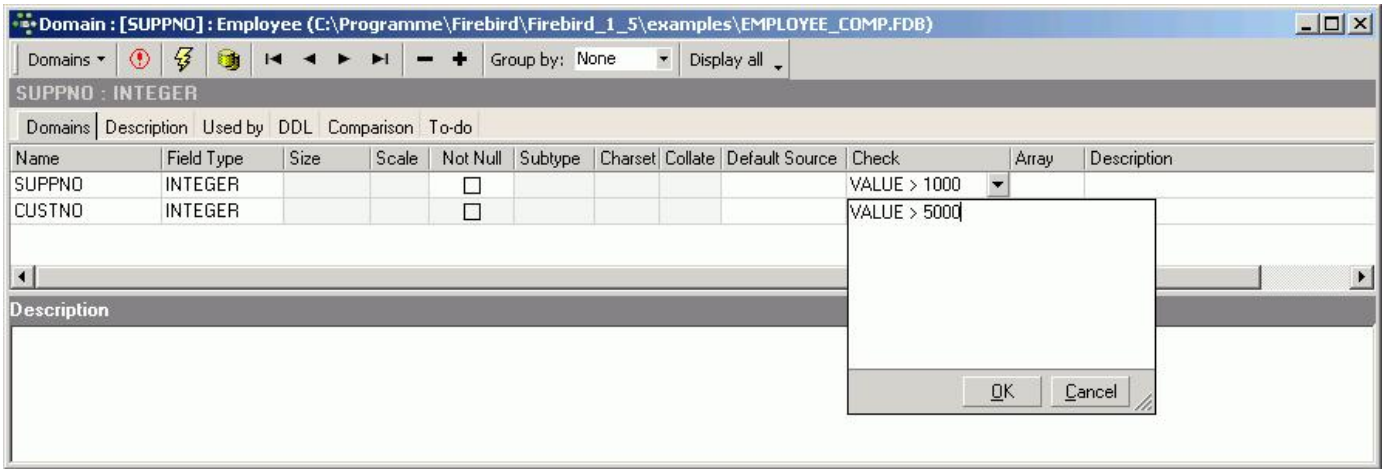
Using SQL the syntax is:

```
DROP DOMAIN <domain_name>;
```

A domain can only be dropped by its creator or the SYSDBA.

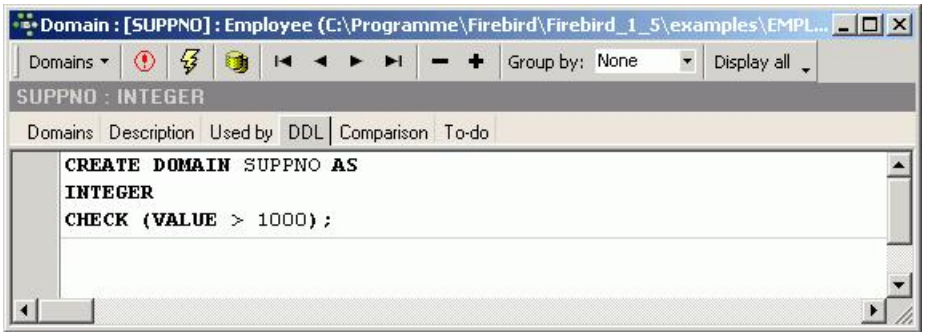
Duplicate domain

It is possible to create a new domain, based on an existing domain, using the Domain Editor´s menu item *Duplicate Domain*, or the



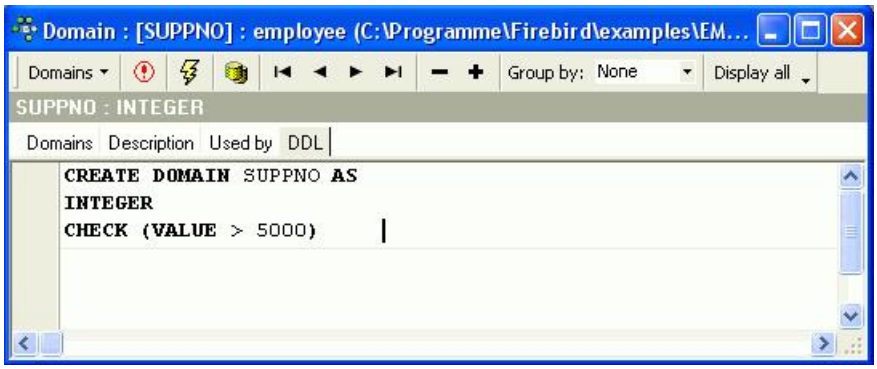
icon in the Domain Editor toolbar.

An exact copy of the selected domain is made, and can then be adapted as wished. For example a new domain, `SUPPNO` could be based on the `CUSTNO` domain in the `EMPLOYEE` database, by duplicating it and then, for example, renaming it and altering the `CHECK VALUE to > 5000`.



This saves time creating several similar domains; all you need to do is copy a domain, perform any minor alterations necessary, compile and finally commit.

The Domain Editor´s *DDL* page displays the actual statement used to create the new domain:



Duplicating domains from one database to another

If you have already created a wide range of domains in one database, and would like to duplicate them in another new database, simply take the following steps in IBExpert:

1. Copy the domain DDL (Data Definition Language) into the SQL Editor and execute it.
2. Drag ´n´ drop the domain from the source database into the Domain Editor of the target database.

See also:
DDL - Data Definition Language
Field

Table

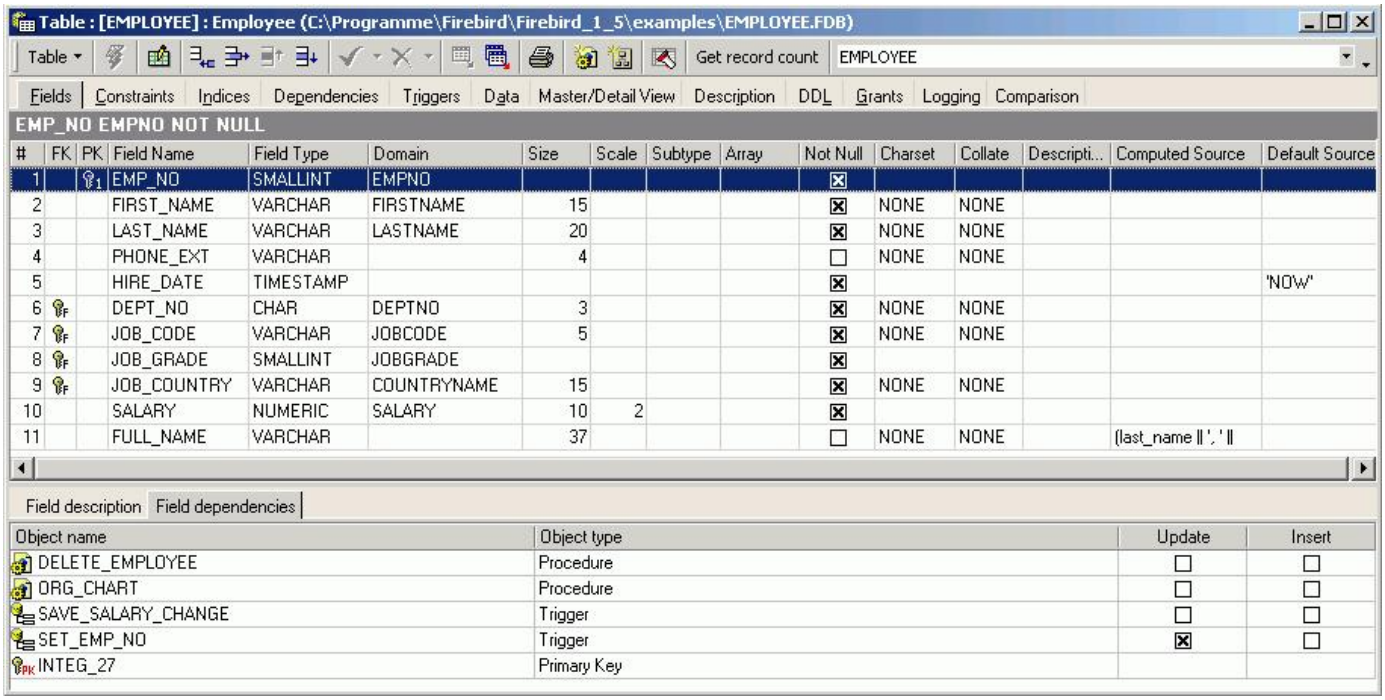
A table is a data storage object consisting of a two-dimensional matrix or grid of columns and rows, theoretically known as a mathematical relation. It is a fundamental element for data storage.

Relational databases store all their data in tables. A table consists of an unordered set of horizontal rows (tuples). Each of these rows contains the same number of vertical columns for the individual singular information types.

The intersection of an individual row and column is a field containing a specific, indivisible atomic piece of information. I.e. columns list the names of individual fields and rows are the data sets containing the input data. Each database column may be assigned a different datatype.

A table is a database object that is part of the database's metadata.

Tables of connected databases can be viewed and manipulated in the IBExpert DB Explorer:



We recommend restricting a table name to no more than 14 characters, so that foreign key names (which are limited to 32 characters up until InterBase 6 and Firebird 1.5; InterBase 7 allows 64 characters) can include both related table names in its name:

Prefix FK plus two separators plus both table names, e.g.

```
FK_Table1_Table2
```

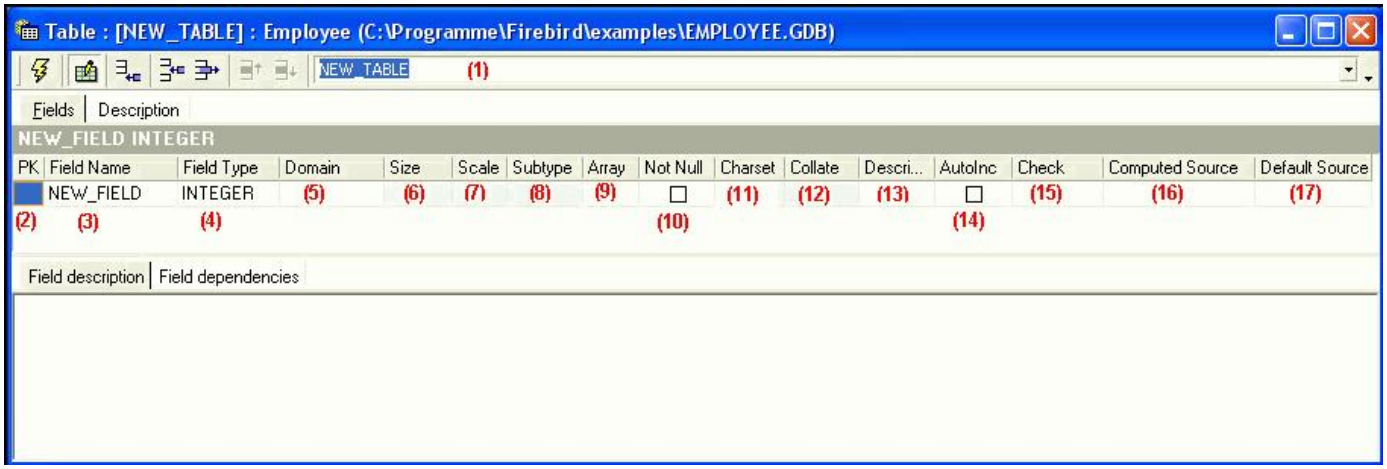
Please note however that this is not an InterBase/Firebird restriction, but purely an IBExpert recommendation to enable a clear and logical naming convention for foreign keys.

New table

A new table can be created in a connected database, either by using the menu item Database / New Table, the respective icon in the New Database Object toolbar, or using the DB Explorer right-click menu (or key combination [Ctrl + N]), when the table heading of the relevant connected database is highlighted. A *NewTable* dialog appears, with its own toolbar (Table Editor toolbar), and a pull-down menu (Table button).

When creating a table it is necessary to define a table name that is unique in the database. At least one column must be specified in order to create the table successfully.

Initially a table name is specified (1) in the upper row:



All data manipulation operations such as SELECT, INSERT, UPDATE and DELETE are carried out using this name.

Fields:

Furthermore, fields can be defined in the Table Editor. At least one field must be defined, so that the table can be committed and registered as an object in the database [Ctrl + F9]. This enables additional table definitions to be made.

An overview of the various input fields is listed below.

Since IBExpert version 2.5.0.61 it is also possible to drag 'n' drop fields from the Database Explorer tree and SQL Assistant into the Table Editor's field list, allowing field definitions to be quickly and easily copied from one table to another.

(2) Primary & Foreign Key: In the first column PK one or more fields can be defined as a primary key (double click). A primary key (PK) serves to uniquely identify a data set, and also acts as an index.

(3) Field Name: Each field should be given a logical name.

(4) Field Type: Here the datatype can be specified.

(5) Domain: Fields can also be based upon domains. If no domain is specified, InterBase/Firebird generates a system domain for the field as specified.

(6) Size: Specifies the field size.

(7) Scale: Here the number of decimal places can be specified here for all numerical fields.

(8) Subtype: A subtype should be specified for blob fields.

(9) Array: Although arrays contradict all the rules of normalization, there are certain situations (for example storing measurement data), when they are necessary. For more information, please refer to arrays.

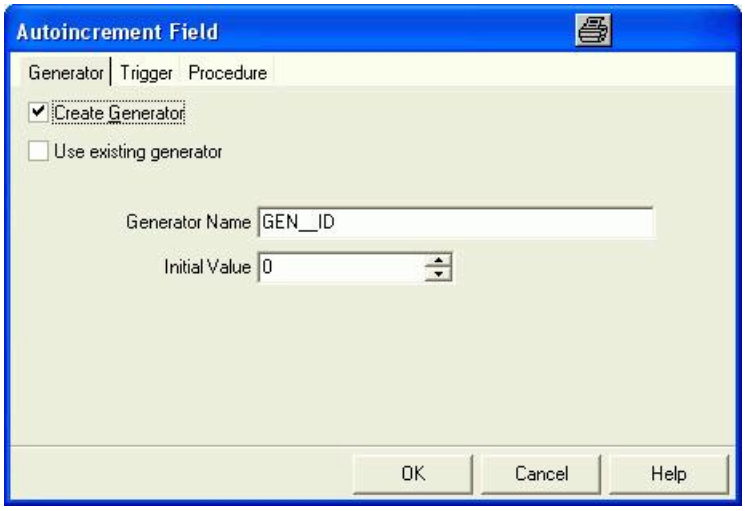
(10) Not Null: This check box can be marked by double-clicking or using the space bar. NOT NULL forces data to be entered in this field (i.e. the field may not be left empty).

(11) Charset: A character set may be specified for individual fields. This overrides the database default character set. Although this is seldom used, it may be necessary should, for example, Asian, Russian or Arabic addresses need to be input and collated in a database with a European default character set.

(12) Collate: This determines the collation for a character set specified for a field.

(13) Description: Useful for database documentation. The *Description* page should be used to describe the table; the *Description* field for describing the field.

(14) Autoinc: Using the space bar or double-click, a new dialog appears, allowing autoincrements (generator, trigger or stored procedure) to be defined.



(15) Check: Each data set is examined according to an expression defined in brackets for validity. Here certain conditions can be specified (see check constraint) causing an automatic database examination during data input, to ensure data consistency in the tables and among each other.

(16) Computed Source: SQL input window for calculations. This can be used for fields containing the results of calculations performed on other fields in the same or other tables in the database.

(17) Default Source: Here a default data entry (text or numeric, depending upon the specified datatype) can be specified, e.g. the text *NOT KNOWN* can be entered as a default source, so that if an address field cannot be input by the user because the information is unavailable, the entry `NOT KNOWN` is automatically entered. It is important to note here, that once a default source has been defined for a field, InterBase/Firebird cannot subsequently alter it (nor subsequently add a default source). The field needs to be dropped, and a new field created.

However, since version 2003.11.6.1 IBExpert has found a way around this. Because the server itself doesn't allow the default value of a field to be altered using `ALTER TABLE` we have implemented a kind of workaround:

First, IBExpert creates the temporary field with the new `DEFAULT` value:

```
ALTER table ADD IBE$$TEMP_COLUMN column_type DEFAULT new_default
```

Secondly, IBExpert copies the `RDB$DEFAULT_SOURCE` and `RDB$DEFAULT_VALUE` values of the newly created temporary field into `RDB$DEFAULT_SOURCE` and `RB$DEFAULT_VALUE` of the field which should be altered:

```
UPDATE RDB$RELATION_FIELDS F1
SET
F1.RDB$DEFAULT_VALUE  = (SELECT F2.RDB$DEFAULT_VALUE
                         FROM RDB$RELATION_FIELDS F2
                         WHERE (F2.RDB$RELATION_NAME = 'table')
                               (F2.RDB$FIELD_NAME ='IBE$$TEMP_COLUMN')),
F1.RDB$DEFAULT_SOURCE = (SELECT F3.RDB$DEFAULT_SOURCE
                         FROM RDB$RELATION_FIELDS F3
                         WHERE (F3.RDB$RELATION_NAME = 'table')
                               (F3.RDB$FIELD_NAME =#IBE$$TEMP_COLUMN'))
WHERE (F1.RDB$RELATION_NAME = 'table')
     (F1.RDB$FIELD_NAME = 'column')
```

After that IBExpert drops the temporary field:

```
ALTER TABLE table DROP IBE$$TEMP_COLUMN
```

Tables can, of course, also be created using DDL directly in the SQL Editor, using the following syntax:

```
CREATE TABLE TABLE_NAME (
COLUMN_NAME1 <COLUMN_DEFINITION>,
COLUMN_NAME2 <COLUMN_DEFINITION>,
...
COLUMN_NAMEn <COLUMN_DEFINITION>;
TABLE_CONSTRAINT1,TABLE_CONSTRAINT2,
...
TABLE_CONSTRAINTn);
```

Once the table has been created do not forget to commit.

Alter table

A table can be altered to change its defined structure. It is even possible to perform multiple changes simultaneously.

Alterations can be made in the Table Editor, opened by double-clicking on the table name in the DB Explorer. Alternatively use the DB Explorer's right mouse-click menu item *Edit Table* or key combination [Ctrl + O].

The following operations may be performed when altering a table:

- Add fields
- Add table level constraints
- Drop fields
- Drop table level constraints
- Modify fields

When dropping fields, it is important to note that the column may not be part of the table's primary key, have a foreign key relationship with another table, contain a unique constraint, be part of a table constraint or part of another column's CHECK constraint.

For further details please refer to Table Editor.

The Constraints page in the Table Editor lists all such fields, so that the developer can quickly ascertain whether constraint alterations/deletions are necessary, before dropping the field in question (or whether, in fact, the field should be dropped at all!).

Using SQL the syntax is:

```
ALTER TABLE <table_name>
ADD <field_name> <field_definition>
ADD CONSTRAINT <constraint_name> <constraint_definition>
DROP CONSTRAINT <constraint_name>
DROP <field_name>;
```

See also:
Firebird 2.0.4 Release Notes: SET/DROP DEFAULT clauses for ALTER TABLE

Drop table/delete table

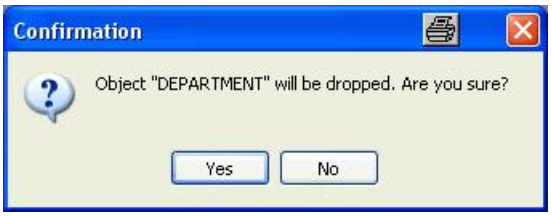
When a table is dropped, all data, metadata and indices in this table are also deleted from the database.

A table can only be dropped, if it is not being used at the time of execution of the DROP command and is not referenced by any other database object, such as in a foreign key relationship, a computed source column or a CHECK constraint for another table, or is a part of the definition of a view or a stored procedure or trigger.

Any existent dependencies can be easily viewed on the Table Editor / Dependencies page. Most database objects can be dropped here directly from the *Dependencies* page or the Dependencies Viewer by right-clicking on the selected object, and choosing the menu item *Drop Object* or [Ctrl + Del].

To drop a table use the DB Explorer, right-click and select the menu item *Drop Table* or [Ctrl + Del].

IBExpert asks for confirmation:



before finally dropping the table. Once dropped, it cannot be retrieved; the table has to be recreated, if a mistake has been made!

Using SQL the syntax is:

```
DROP TABLE <table_name>;
```

Create SIUD procedures

By right-clicking on a table in the DB Explorer, you will find a menu item called *Create SIUD Procedures*. SIUD is the abbreviation for SELECT, INSERT, UPDATE and DELETE.

If you want to prevent database users from directly manipulating data with INSERT, UPDATE and DELETE statements, you can use these procedures, which can be executed.

Please refer to Create Procedure from Table for details.

See also:
SQL Language Reference
Data Definition Language (DDL)
Data Manipulation Language (DML)
INSERTEX
New Database Object toolbar
Table Editor toolbar
Table Editor
Keys
Definitions

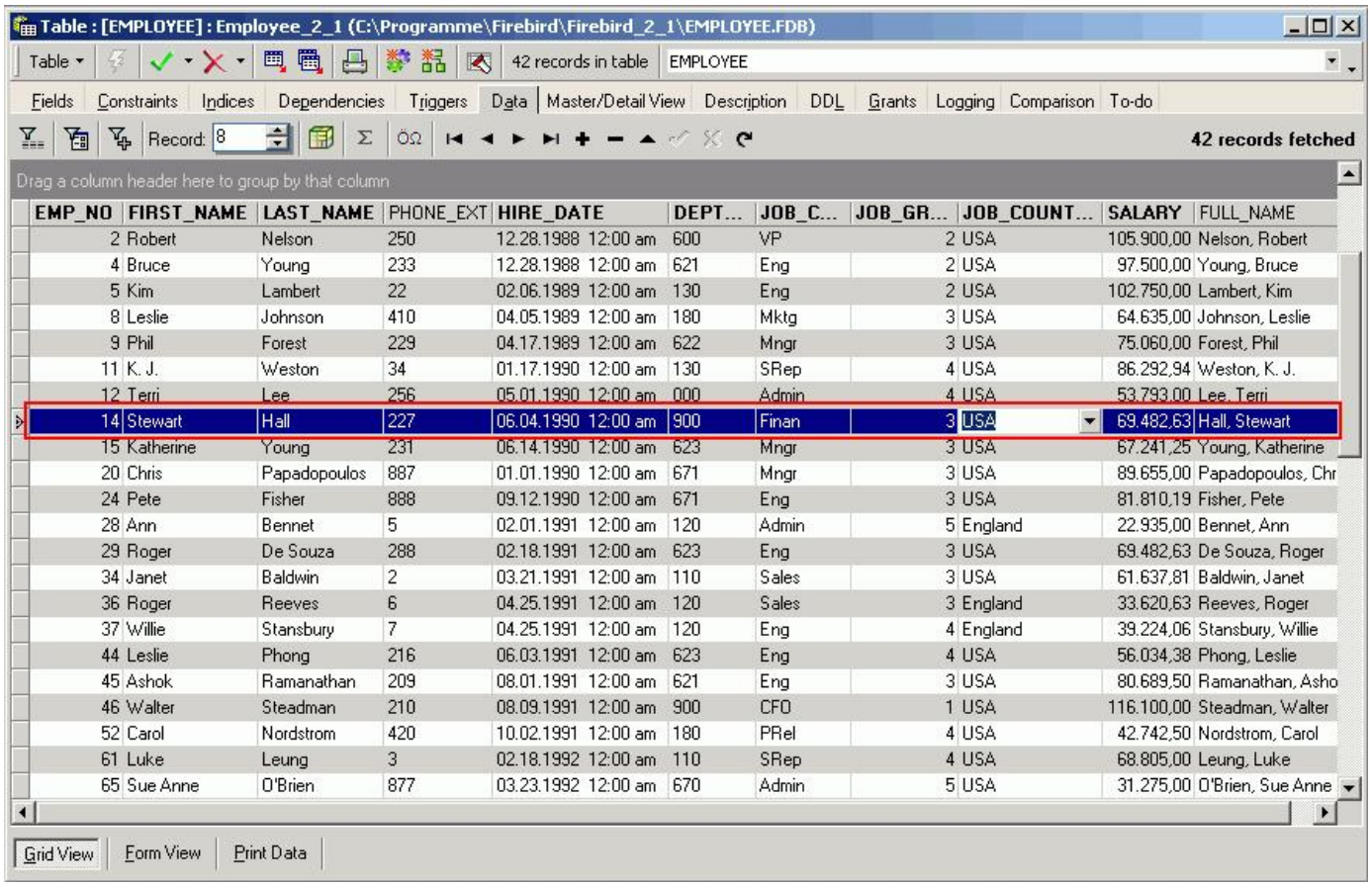
Definitions

Data

Data is the quantity of facts or information input, processed and stored in a computer. Data can consist of one single entry in one field, a data set comprises a series of fields or in fact, any data quantity.

Data set

A data set is one complete data record, which is none other than a table row (which can be viewed on the IBExpert Table Editor / Data page). It encompasses a single set of information, such as, for example, one customer address or one employee record.



EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT...	JOB_C...	JOB_GR...	JOB_COUNT...	SALARY	FULL_NAME
2	Robert	Nelson	250	12.28.1988 12:00 am	600	VP	2	USA	105.900,00	Nelson, Robert
4	Bruce	Young	233	12.28.1988 12:00 am	621	Eng	2	USA	97.500,00	Young, Bruce
5	Kim	Lambert	22	02.06.1989 12:00 am	130	Eng	2	USA	102.750,00	Lambert, Kim
8	Leslie	Johnson	410	04.05.1989 12:00 am	180	Mktg	3	USA	64.635,00	Johnson, Leslie
9	Phil	Forest	229	04.17.1989 12:00 am	622	Mngr	3	USA	75.060,00	Forest, Phil
11	K. J.	Weston	34	01.17.1990 12:00 am	130	SRep	4	USA	86.292,94	Weston, K. J.
12	Terri	Lee	256	05.01.1990 12:00 am	000	Admin	4	USA	53.793,00	Lee, Terri
14	Stewart	Hall	227	06.04.1990 12:00 am	900	Finan	3	USA	69.482,63	Hall, Stewart
15	Katherine	Young	231	06.14.1990 12:00 am	623	Mngr	3	USA	67.241,25	Young, Katherine
20	Chris	Papadopoulos	887	01.01.1990 12:00 am	671	Mngr	3	USA	89.655,00	Papadopoulos, Chr
24	Pete	Fisher	888	09.12.1990 12:00 am	671	Eng	3	USA	81.810,19	Fisher, Pete
28	Ann	Bennet	5	02.01.1991 12:00 am	120	Admin	5	England	22.935,00	Bennet, Ann
29	Roger	De Souza	288	02.18.1991 12:00 am	623	Eng	3	USA	69.482,63	De Souza, Roger
34	Janet	Baldwin	2	03.21.1991 12:00 am	110	Sales	3	USA	61.637,81	Baldwin, Janet
36	Roger	Reeves	6	04.25.1991 12:00 am	120	Sales	3	England	33.620,63	Reeves, Roger
37	Willie	Stansbury	7	04.25.1991 12:00 am	120	Eng	4	England	39.224,06	Stansbury, Willie
44	Leslie	Phong	216	06.03.1991 12:00 am	623	Eng	4	USA	56.034,38	Phong, Leslie
45	Ashok	Ramanathan	209	08.01.1991 12:00 am	621	Eng	3	USA	80.689,50	Ramanathan, Asho
46	Walter	Steadman	210	08.09.1991 12:00 am	900	CFO	1	USA	116.100,00	Steadman, Walter
52	Carol	Nordstrom	420	10.02.1991 12:00 am	180	PRel	4	USA	42.742,50	Nordstrom, Carol
61	Luke	Leung	3	02.18.1992 12:00 am	110	SRep	4	USA	68.805,00	Leung, Luke
65	Sue Anne	O'Brien	877	03.23.1992 12:00 am	670	Admin	5	USA	31.275,00	O'Brien, Sue Anne

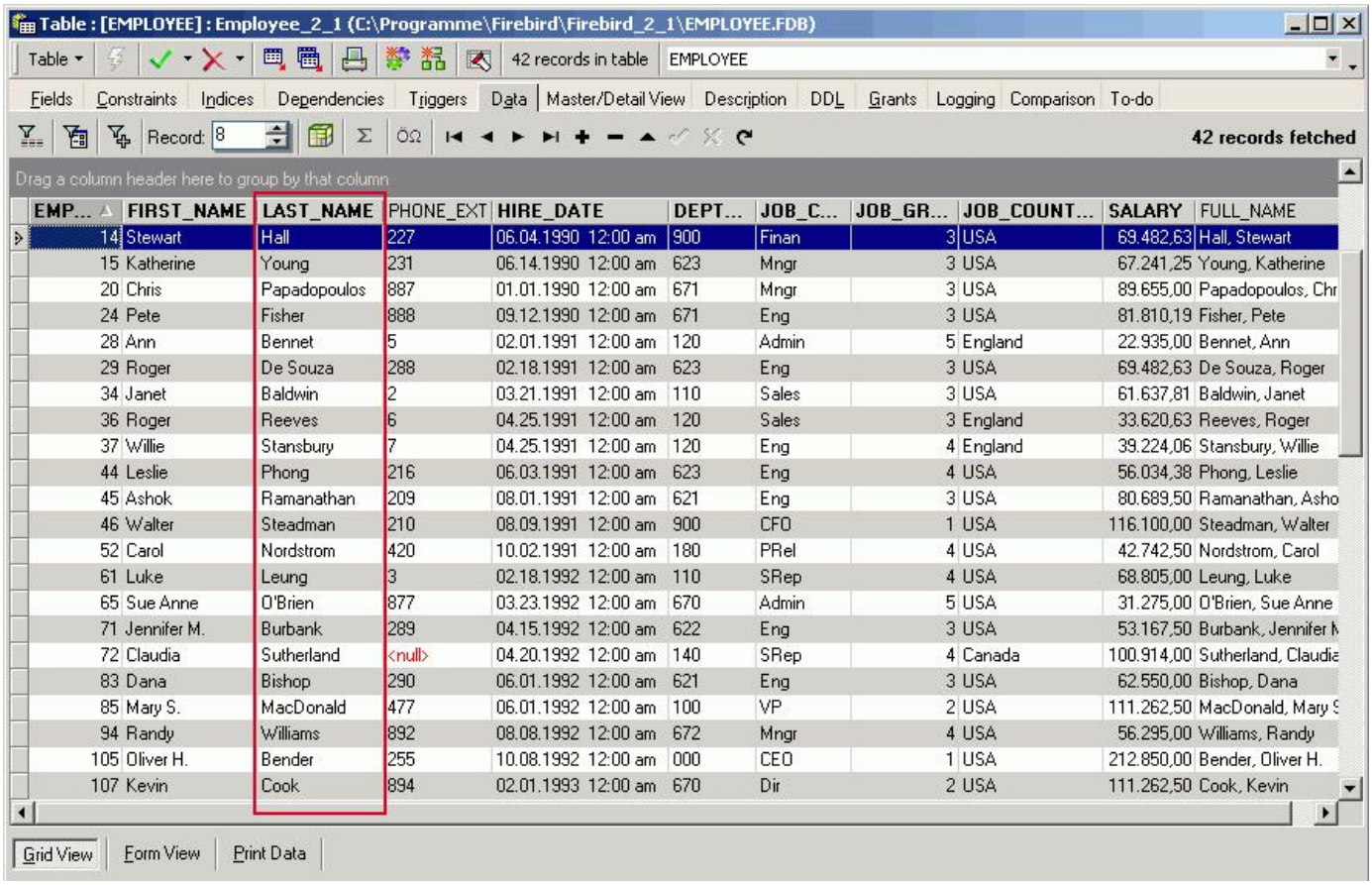
In a relational database the physical sequence of data sets is irrelevant.

Duplicate data sets or records (i.e. double rows) are not allowed in a relational database, as this is, in effect, storage of redundant information (see Database Normalization).

Column

A column is part of a database table, and is also known as an attribute or field. Columns list the names of the individual fields in a table.

A column describes an atomic or indivisible basic piece of information in the database, clearly differentiated from other data, e.g. zip code (and not zip code + city). Each column is assigned a certain datatype, e.g. text, numeric, date or blob. The data can also be assigned properties, such as unique, contain check constraints, autoincrements, computed values, restricted to minimum and maximum values etc. etc.



EMP...	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT...	JOB_C...	JOB_GR...	JOB_COUNT...	SALARY	FULL_NAME
14	Stewart	Hall	227	06.04.1990 12:00 am	900	Finan	3	USA	69.482,63	Hall, Stewart
15	Katherine	Young	231	06.14.1990 12:00 am	623	Mngr	3	USA	67.241,25	Young, Katherine
20	Chris	Papadopoulos	887	01.01.1990 12:00 am	671	Mngr	3	USA	89.655,00	Papadopoulos, Chr
24	Pete	Fisher	888	09.12.1990 12:00 am	671	Eng	3	USA	81.810,19	Fisher, Pete
28	Ann	Bennet	5	02.01.1991 12:00 am	120	Admin	5	England	22.935,00	Bennet, Ann
29	Roger	De Souza	288	02.18.1991 12:00 am	623	Eng	3	USA	69.482,63	De Souza, Roger
34	Janet	Baldwin	2	03.21.1991 12:00 am	110	Sales	3	USA	61.637,81	Baldwin, Janet
36	Roger	Reeves	6	04.25.1991 12:00 am	120	Sales	3	England	33.620,63	Reeves, Roger
37	Willie	Stansbury	7	04.25.1991 12:00 am	120	Eng	4	England	39.224,06	Stansbury, Willie
44	Leslie	Phong	216	06.03.1991 12:00 am	623	Eng	4	USA	56.034,38	Phong, Leslie
45	Ashok	Ramanathan	209	08.01.1991 12:00 am	621	Eng	3	USA	80.689,50	Ramanathan, Asho
46	Walter	Steadman	210	08.09.1991 12:00 am	900	CFO	1	USA	116.100,00	Steadman, Walter
52	Carol	Nordstrom	420	10.02.1991 12:00 am	180	PRel	4	USA	42.742,50	Nordstrom, Carol
61	Luke	Leung	3	02.18.1992 12:00 am	110	SRep	4	USA	68.805,00	Leung, Luke
65	Sue Anne	O'Brien	877	03.23.1992 12:00 am	670	Admin	5	USA	31.275,00	O'Brien, Sue Anne
71	Jennifer M.	Burbank	289	04.15.1992 12:00 am	622	Eng	3	USA	53.167,50	Burbank, Jennifer M
72	Claudia	Sutherland	<null>	04.20.1992 12:00 am	140	SRep	4	Canada	100.914,00	Sutherland, Claudia
83	Dana	Bishop	290	06.01.1992 12:00 am	621	Eng	3	USA	62.550,00	Bishop, Dana
85	Mary S.	MacDonald	477	06.01.1992 12:00 am	100	VP	2	USA	111.262,50	MacDonald, Mary S
94	Randy	Williams	892	08.08.1992 12:00 am	672	Mngr	4	USA	56.295,00	Williams, Randy
105	Oliver H.	Bender	255	10.08.1992 12:00 am	000	CEO	1	USA	212.850,00	Bender, Oliver H.
107	Kevin	Cook	894	02.01.1993 12:00 am	670	Dir	2	USA	111.262,50	Cook, Kevin

Columns are defined under the *Field Definition* in the Create Table dialog or Table Editor, or their definition can be based on domains. They can, of course, also be defined directly in the SQL Editor. Each defined column has the following syntax:

```
ColumnName <data_type>
DEFAULT < Default value > | NULL | USER NOT NULL
CONSTRAINT <constraint name> <constraint def>
COLLATE <collation sequence>;
```

In a relational database the physical sequence of rows and columns is irrelevant.

Row

A row is also called a tuple, record or data set. Each row represents an instance of data, belonging together, composed of different columns. It encompasses a single set of information, such as, for example, one customer address or one employee record.

Table : [EMPLOYEE] : Employee_2_1 (C:\Programme\Firebird\Firebird_2_1\EMPLOYEE.FDB)

Table | 42 records in table | EMPLOYEE

Fields | Constraints | Indices | Dependencies | Triggers | Data | Master/Detail View | Description | DDL | Grants | Logging | Comparison | To-do

Record: 8 | 42 records fetched

Drag a column header here to group by that column

EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT...	JOB_C...	JOB_GR...	JOB_COUNT...	SALARY	FULL_NAME
2	Robert	Nelson	250	12.28.1988 12:00 am	600	VP	2	USA	105.900,00	Nelson, Robert
4	Bruce	Young	233	12.28.1988 12:00 am	621	Eng	2	USA	97.500,00	Young, Bruce
5	Kim	Lambert	22	02.06.1989 12:00 am	130	Eng	2	USA	102.750,00	Lambert, Kim
8	Leslie	Johnson	410	04.05.1989 12:00 am	180	Mktg	3	USA	64.635,00	Johnson, Leslie
9	Phil	Forest	229	04.17.1989 12:00 am	622	Mngr	3	USA	75.060,00	Forest, Phil
11	K. J.	Weston	34	01.17.1990 12:00 am	130	SRep	4	USA	86.292,94	Weston, K. J.
12	Terri	Lee	256	05.01.1990 12:00 am	000	Admin	4	USA	53.793,00	Lee, Terri
14	Stewart	Hall	227	06.04.1990 12:00 am	900	Finan	3	USA	69.482,63	Hall, Stewart
15	Katherine	Young	231	06.14.1990 12:00 am	623	Mngr	3	USA	67.241,25	Young, Katherine
20	Chris	Papadopoulos	887	01.01.1990 12:00 am	671	Mngr	3	USA	89.655,00	Papadopoulos, Chr
24	Pete	Fisher	888	09.12.1990 12:00 am	671	Eng	3	USA	81.810,19	Fisher, Pete
28	Ann	Bennet	5	02.01.1991 12:00 am	120	Admin	5	England	22.935,00	Bennet, Ann
29	Roger	De Souza	288	02.18.1991 12:00 am	623	Eng	3	USA	69.482,63	De Souza, Roger
34	Janet	Baldwin	2	03.21.1991 12:00 am	110	Sales	3	USA	61.637,81	Baldwin, Janet
36	Roger	Reeves	6	04.25.1991 12:00 am	120	Sales	3	England	33.620,63	Reeves, Roger
37	Willie	Stansbury	7	04.25.1991 12:00 am	120	Eng	4	England	39.224,06	Stansbury, Willie
44	Leslie	Phong	216	06.03.1991 12:00 am	623	Eng	4	USA	56.034,38	Phong, Leslie
45	Ashok	Ramanathan	209	08.01.1991 12:00 am	621	Eng	3	USA	80.689,50	Ramanathan, Asho
46	Walter	Steadman	210	08.09.1991 12:00 am	900	CFO	1	USA	116.100,00	Steadman, Walter
52	Carol	Nordstrom	420	10.02.1991 12:00 am	180	PRel	4	USA	42.742,50	Nordstrom, Carol
61	Luke	Leung	3	02.18.1992 12:00 am	110	SRep	4	USA	68.805,00	Leung, Luke
65	Sue Anne	O'Brien	877	03.23.1992 12:00 am	670	Admin	5	USA	31.275,00	O'Brien, Sue Anne

Grid View | Form View | Print Data

In a relational database the physical sequence of rows and columns is irrelevant.

Double rows (i.e. duplicate data sets or records) are not allowed in a relational table, as this is, in effect, storage of redundant information (see Database Normalization).

Constraints

A constraint is a database examination, which ensures data consistency in the tables and among each other.

The constraint determines the range of acceptable values for a column (or columns) or data set in a database or application. This constraint can be executed automatically and so ensures that data contents are kept consistent by testing them as they are input.

A constraint can be specified for each column (or columns) in a table, to guarantee the mechanism described above. Constraints can be domain- or column-based and the specified conditions must be met when new data sets are inserted, or existing data sets are modified. They are used to verify data integrity. If a condition is not met, an exception is raised.

InterBase/Firebird internally generates a trigger for each check condition. Constraints can be defined as follows:

1. **Primary Key/Unique:** Specification of the unique option forces a unique entry in this column (these columns) for each data set (i.e. duplicate field entries are not allowed).

Table : [EMPLOYEE] : Employee_2_1 (C:\Programme\Firebird\Firebird_2_1\EMPLOYEE.FDB)

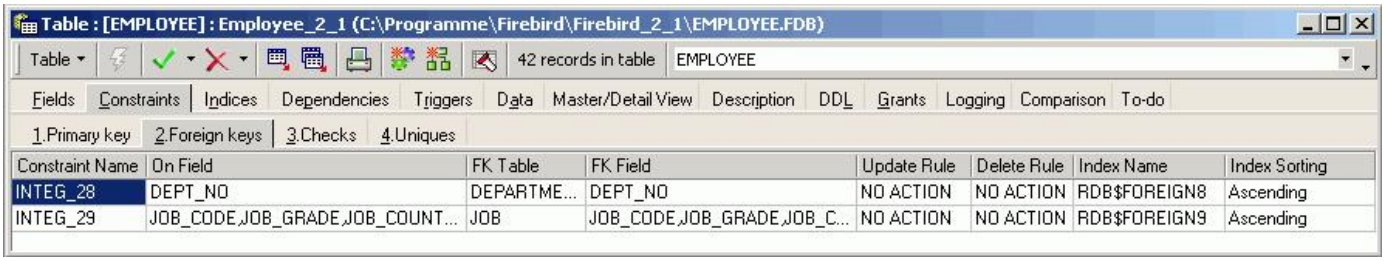
Table | 42 records in table | EMPLOYEE

Fields | Constraints | Indices | Dependencies | Triggers | Data | Master/Detail View | Description | DDL | Grants | Logging | Comparison | To-do

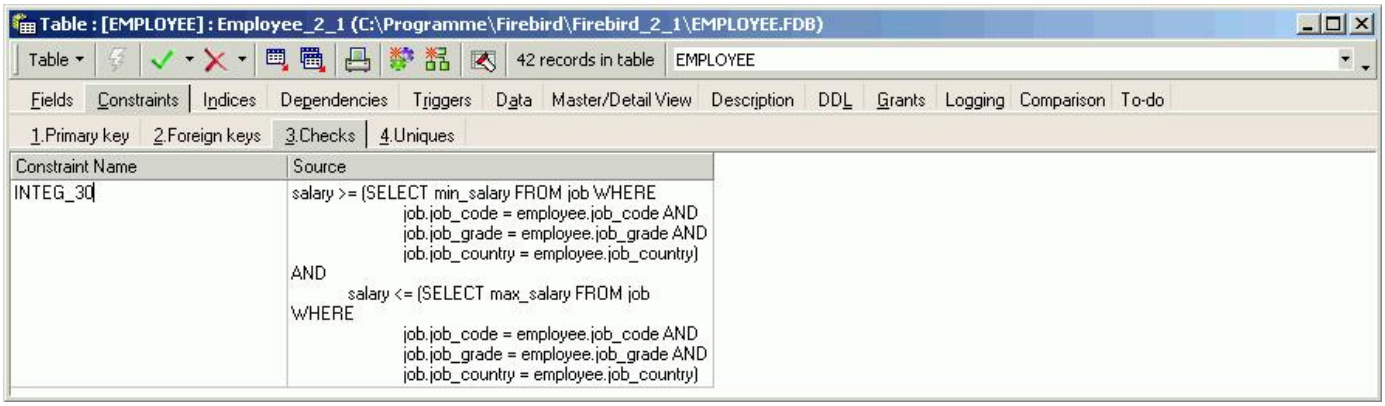
1.Primary key | 2.Foreign keys | 3.Checks | 4.Uniques

Constraint Name	On Field	Index Name	Index Sorting
INTEG_27	EMP_NO	RDB$PRIMARY7	Ascending

2. **Foreign Key:** The foreign key option determines that the column(s) is/are linked by a referential integrity relationship to the primary key of another table (i.e. the input data is only accepted if it already exists in the primary key column(s) in the referenced table).



3. **CHECK:** the check option enables each data set to be examined for validation of an expression specified in brackets. Check constraints in tables are identical to check constraints in domains.



Only one constraint is permitted per column. If the column including a constraint is based on a domain also containing a constraint, both constraints are active.

The specification of the keyword `CONSTRAINT` and the name are optional for all constraints. If no name is specified, InterBase/Firebird generates a name automatically. All constraint names are stored in a system table called `DB$RELATION_CONSTRAINTS`.

It is only necessary to name constraints, if they are to be deactivated at a later date using the `ALTER TABLE DROP` statement.

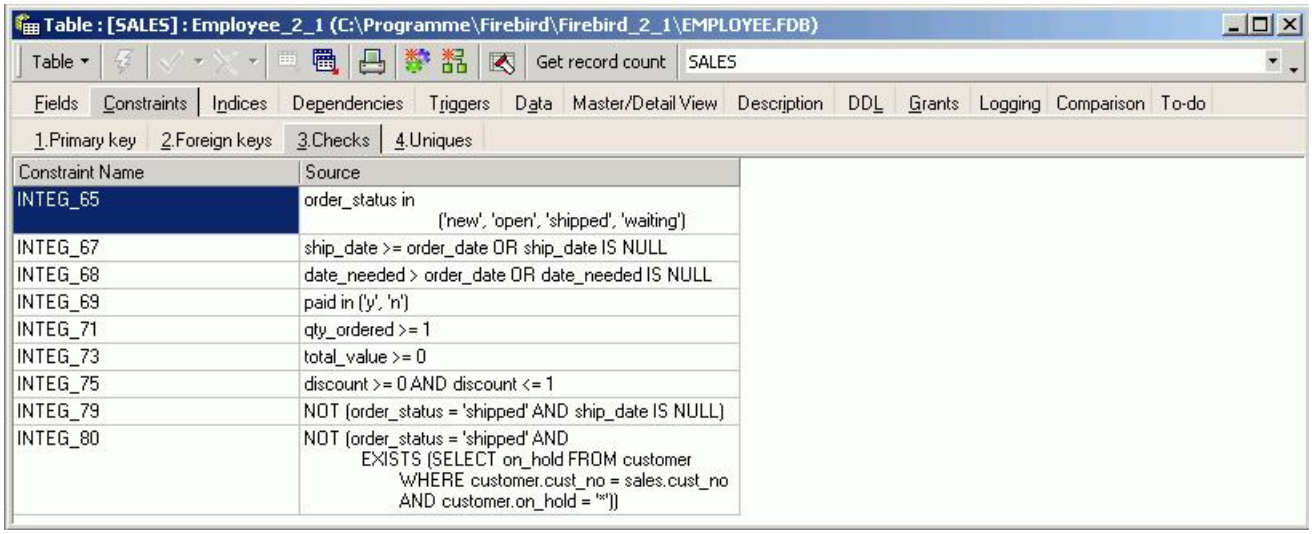
From InterBase 5 onwards, cascading referential integrity is also supported.

Check constraint

A check is a database examination, which ensures data consistency in the tables among each other. It can be executed automatically and so ensures that data contents are kept consistent by testing them before they are stored in the database.

The check constraint option enables each data set to be examined for validation of the expression in brackets following the check constraint. Check constraints in tables are identical to check constraints in domains.

A check constraint can be specified for each column in a table, to guarantee the mechanism described above. It includes an expression that must be true, so that the data set following an insert or update can be written. The field contents must be included in the permissible values, which can be specified in a list. It is also possible to test the value for a minimum and maximum value. Furthermore the value can be compared to values in other columns, in order to test dependencies.

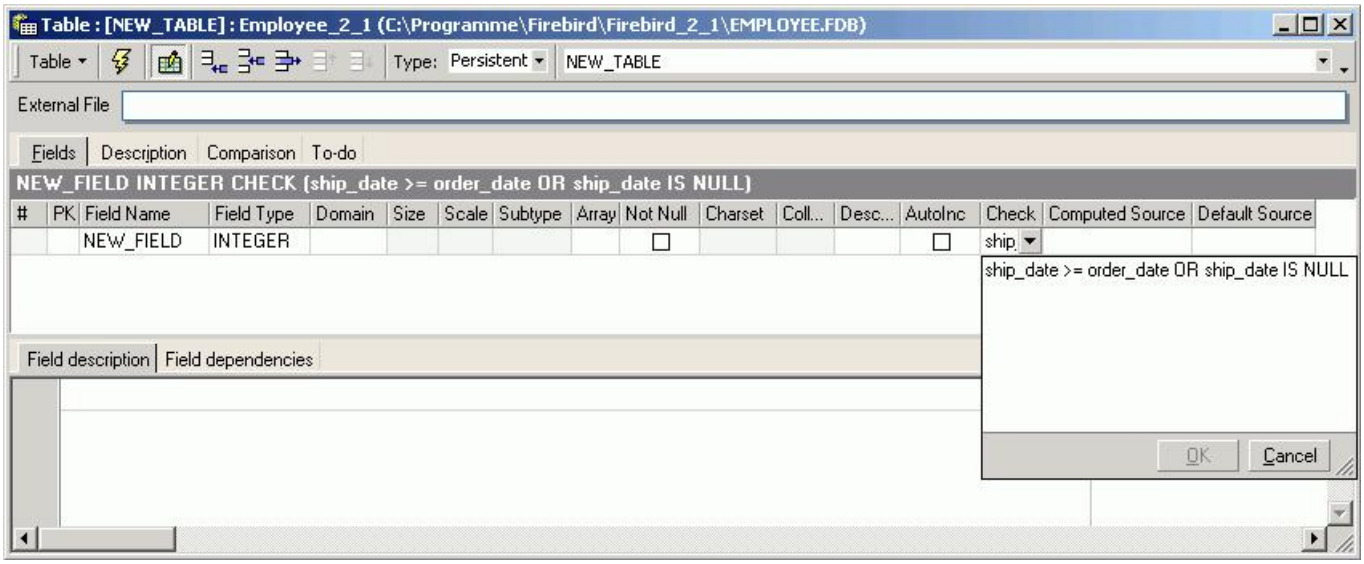


A check constraint can only examine the values in the current data set. When simultaneously inserting or altering multiple data sets, a check constraint can only guarantee one data integrity at a time at data set level.

If other data sets are referenced in the check, these could have been modified by another user at the time of entry, and therefore possibly have become invalid, even though the check constraint's test approved the data set. At the time of a check constraint validation, other data is only read for the check. For this reason, the values for the current operating sequence remain constant, even if another user has modified one of the values already referenced for validation.

A check constraint can be created directly when creating a table. When creating a check constraint, the following criteria should be taken into consideration:

- A check constraint cannot reference a domain.
- A table column can only contain one check constraint.
- A check constraint defined by a domain cannot be overridden by a local check constraint. However additional constraints can be specified.



In a check definition the `VALUE` keyword represents the value of the respective table column. The value examination is generally performed when inserting or updating this table column. The Check Value options permit diverse operations (please refer to Comparison Operators for a full list of possible operators).

Referential integrity declarations and primary key definitions are special check constraint compositions.

Only one constraint is permitted per column. If the column is based on a domain containing a constraint, both check constraints are active.

The specification of the keyword `CONSTRAINT` and the name are optional for all constraints. If no name is specified, InterBase/Firebird generates a name automatically. All constraint names are stored in a system table called `DB$RELATION_CONSTRAINTS`.

It is only necessary to name constraints, if they are to be deactivated at a later date using the `ALTER TABLE DROP` statement.

Please note that if you want to change the `CHECK` constraint for a domain that already has a constraint defined, the existing constraint must first be dropped and then the new one added. `ADD CHECK` does not replace the current constraint with the new one. It is also important to realize that altering a `CHECK` constraint does not cause existing database rows to be revalidated; `CHECK` constraints are only validated when an `INSERT` or `UPDATE` is performed. One way of overcoming this limitation is to perform an `UPDATE` query using a dummy operation. If existing rows violate the new `CHECK` constraint, the query fails. These rows can then be extracted by performing a `SELECT`.

Index/indices

An index can be compared to a book index enabling rapid search capabilities.

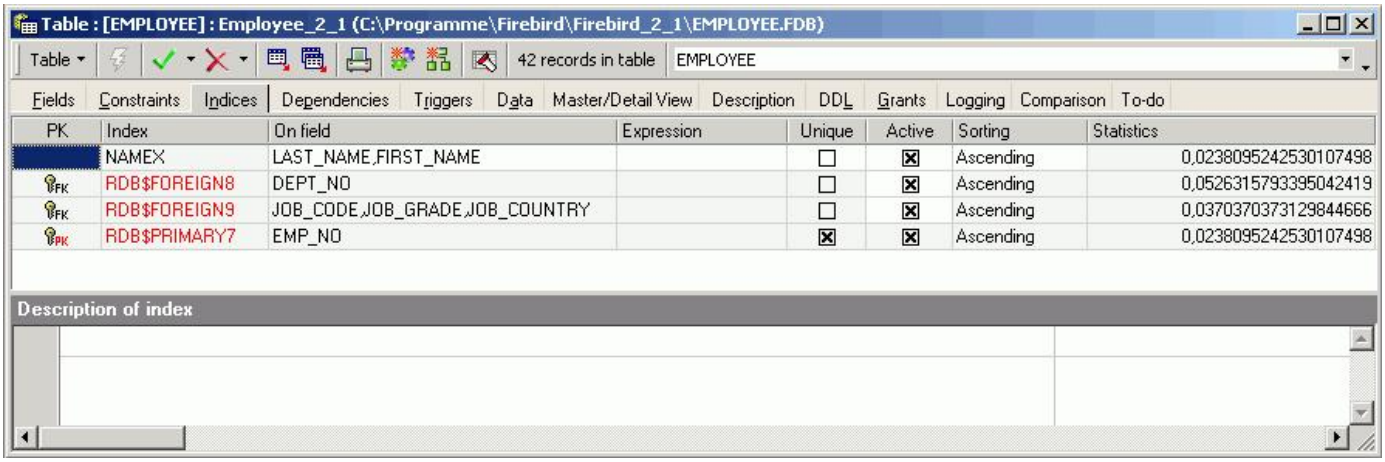
Indices are a sorted list of pointers into tables, to speed data access. They can be best described as an alphabetical directory with internal pointers, where what can be found. If the indexed field is unique there is only one pointer.

An index can be ascending or descending, and can also be defined as unique if wished.

Indices should not be confused with keys. In the relational model, a key is used to organize data logically, so that specific rows can be identified. An index, however, is part of the table's physical structure on-disk, and is used to increase the performance of tables during queries. Indices are therefore not a part of the relational model. In spite of this indices are extremely important for relational database systems.

For columns defined with a primary key or a foreign key in a table, InterBase/Firebird automatically generates a corresponding ascending index and enforces the uniqueness constraint demanded by the relational model.

An index can be defined in the IBExpert Table Editor (started from the DB Explorer):



Up to and including Firebird 1.5 up to 64 indices can be defined for each table. Since Firebird 2.0 this number has risen to 255.

Indices are updated every time a new data set is inserted, or rather, the index-referenced field is updated. InterBase/Firebird writes an additional second mini version of the data set in each index table.

An index has a sequence e.g. when an ascending index is assigned to a field (default), and a descending `select` on this field is requested, InterBase/Firebird does not sort using the ascending index. For this a second descending index needs to be specified for the same field.

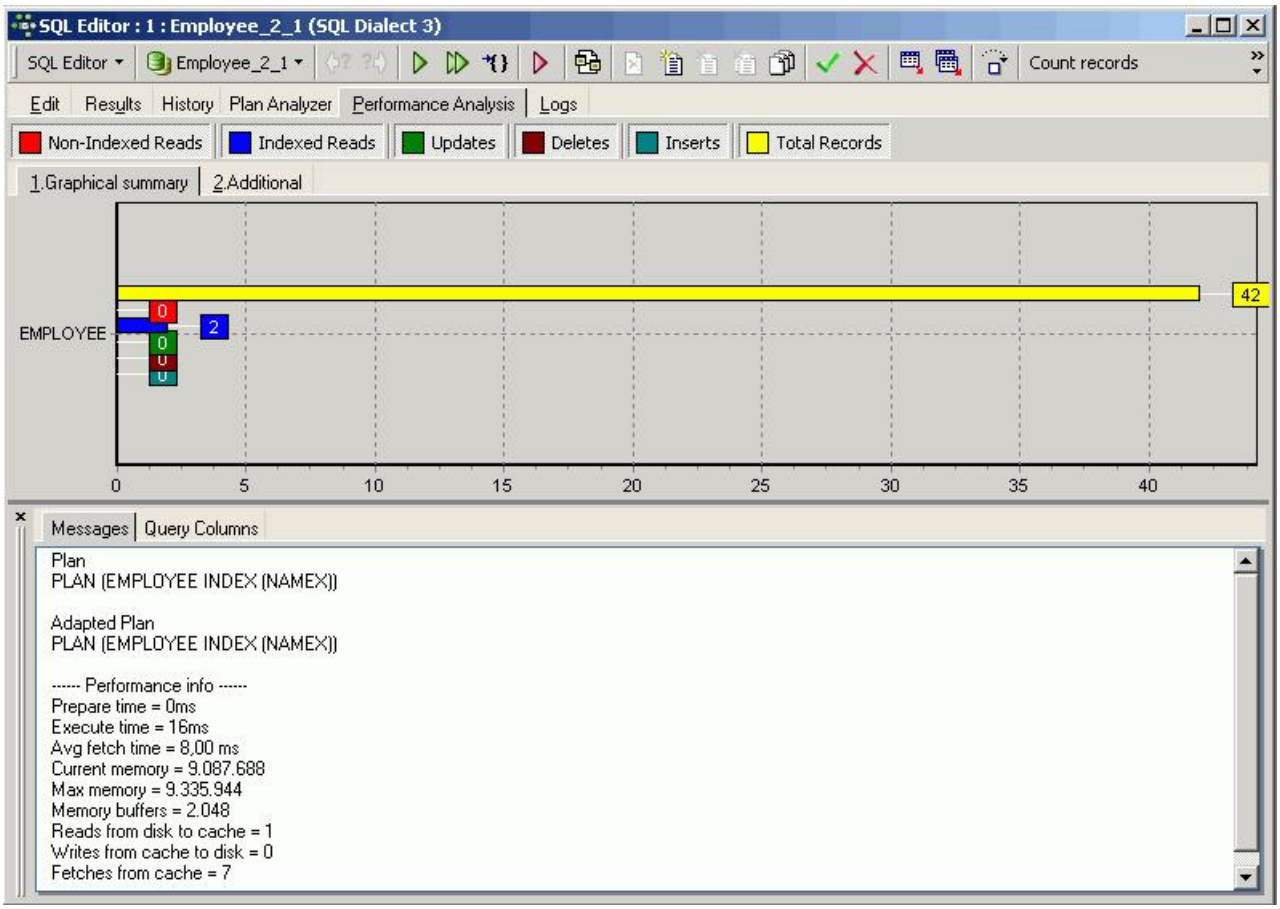
An index can be named as wished; consecutive numbers can even be used, as it is extremely rare that an index is named in SQL.

An index on two fields simultaneously only makes sense when both fields are to be sorted using `ORDER BY`, and this should only be used on relatively small quantities of results.

InterBase/Firebird decides automatically which index it uses to carry out `SELECT` requests. On the Table Editor / Indices page under *Statistics*, it can be seen that the index with the lowest value has a higher uniqueness, and is therefore preferred by InterBase/Firebird instead of other indices with a lower level of uniqueness. This is known as selectivity.

An index should only be used on fields which are really used frequently as sorting criteria (e.g. fields such as `STREET` and `MALE/FEMALE` are generally unimportant) or in a `WHERE` condition. If a field is often used as a sorting criterion, a descending index should also be considered, e.g. in particular on `DATE` or `TIMESTAMP` fields. Care should also be taken that indexed `CHAR` fields are not larger than approximately 80 characters in length (with Firebird 1.5 the limit is somewhat higher).

Indices can always be set after the database is actually in use, based on the performance requirements. For further details and examples please refer to Performance Analysis.



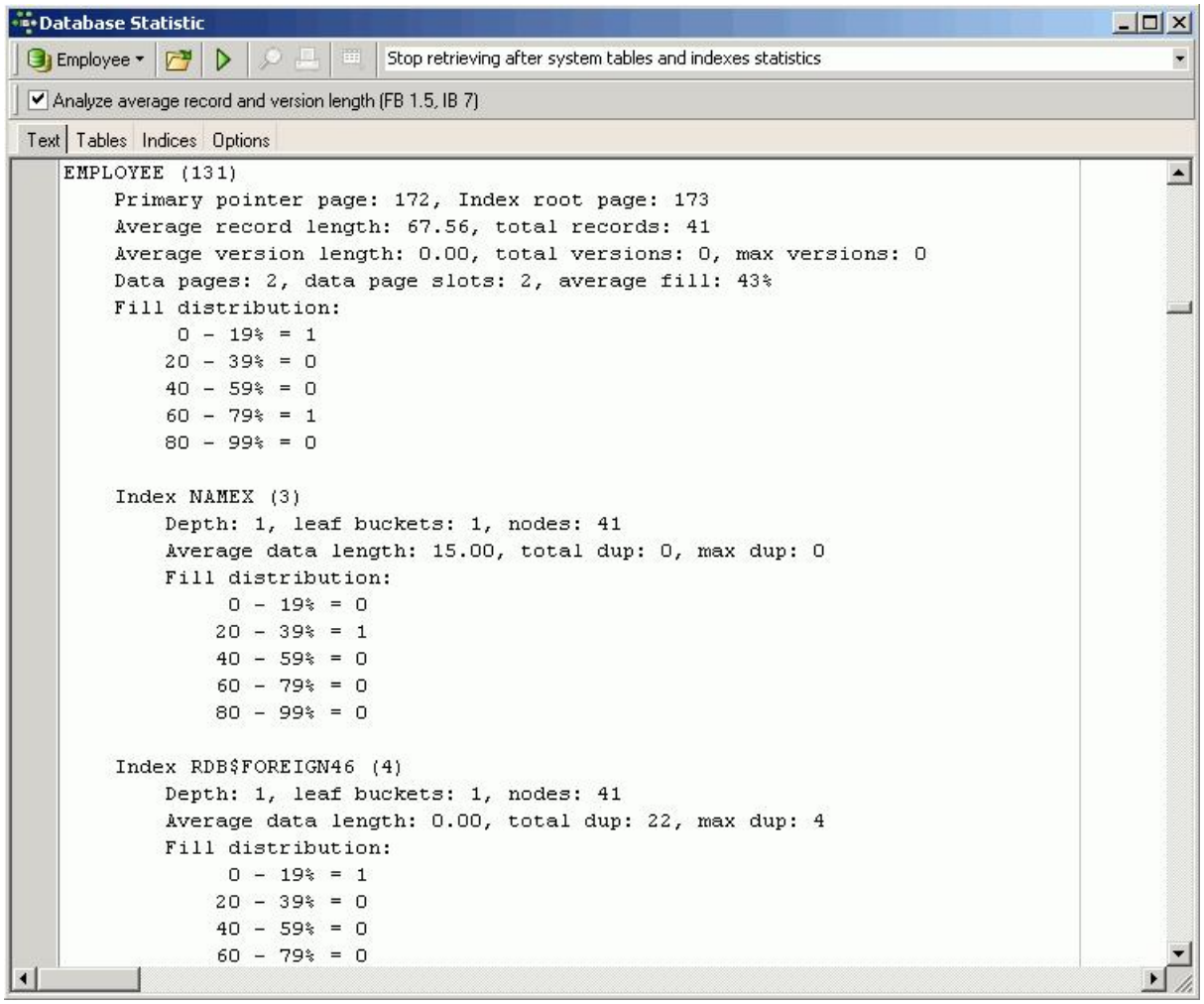
Using the IBExpert menu Services / Database Statistics the index statistics can be viewed.

Index statistics and index selectivity

When a query is sent to the server, the Optimizer does not intuitively know how to process it. It needs further information to help it decide how to go about executing the query. For this it uses indices, and to decide which index is the best to use first, it relies on the index selectivity. The selectivity of an index is the best clue that the query plan has whether it should use a certain index or not. And when more than one index is available, it helps the Firebird server decide, which index to use first.

So the first thing the Optimizer does when it receives a query, is to prepare the execution. It makes decisions regarding indices based solely upon their selectivity.

If you have an index on a field with only two distinct values (e.g. `yes` or `no`) in it, it will have a selectivity of 0.5. If your indexed field has 10 values, it will have a selectivity of 0.1. The higher the number of different values, the lower the selectivity number and the more suitable it is to be used as an index. Your benchmark is always your ID - the primary key, because that will always have complete unique values in it, and therefore the lowest selectivity.

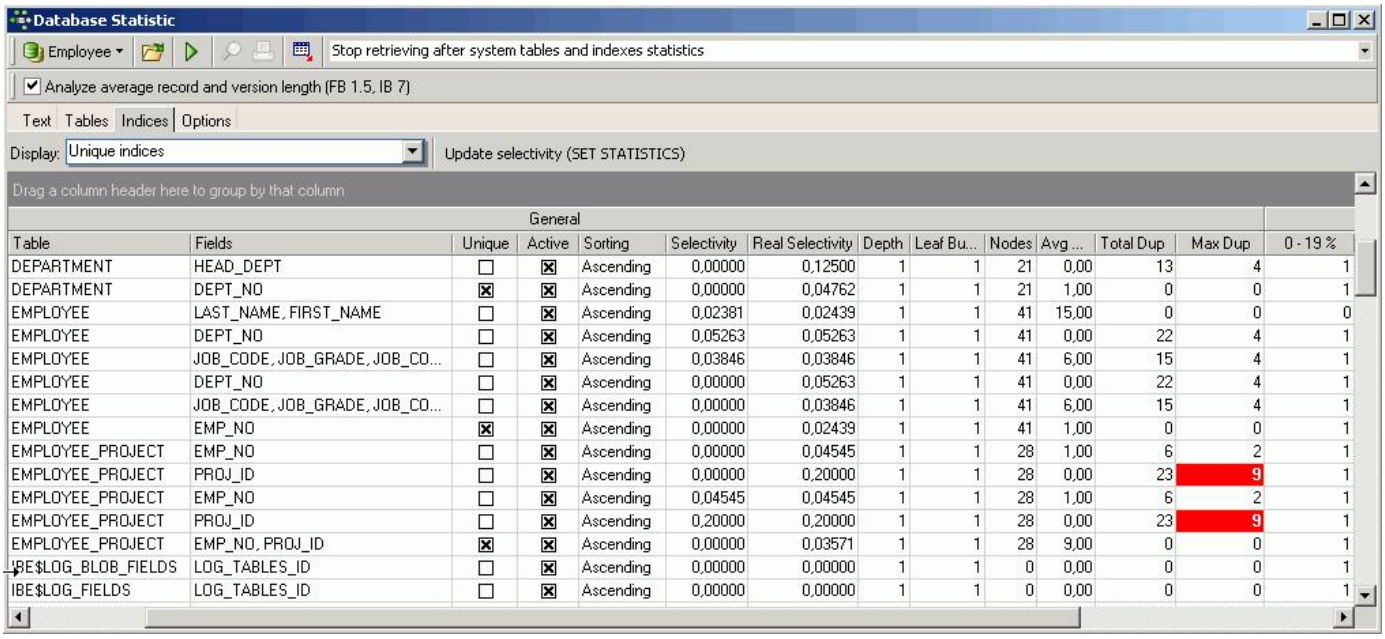


The selectivity is only computed at the time of creation, or when the IBExpert menu item *Recompute Selectivity* or *Recompute All* is used (found directly in the IBExpert Services menu item, Database Statistics dialog, in the Database menu, or in the right-click DB Explorer menu). Alternatively the

```
SET STATISTIC INDEX {INDEX_NAME}
```

command can be used in the SQL Editor to recompute individual indices. To automate regular recalculation of all indices, please refer to the next chapter, *Automating maintenance operations*.

This is automatically performed during a database backup and restore, as it is not the index, but its definition that is saved, and so the index is therefore reconstructed when the database is restored.



				General									
Table	Fields	Unique	Active	Sorting	Selectivity	Real Selectivity	Depth	Leaf Bu...	Nodes	Avg ...	Total Dup	Max Dup	0 - 19 %
DEPARTMENT	HEAD_DEPT	☐	☒	Ascending	0,00000	0,12500	1	1	21	0,00	13	4	1
DEPARTMENT	DEPT_NO	☒	☒	Ascending	0,00000	0,04762	1	1	21	1,00	0	0	1
EMPLOYEE	LAST_NAME, FIRST_NAME	☐	☒	Ascending	0,02381	0,02439	1	1	41	15,00	0	0	0
EMPLOYEE	DEPT_NO	☐	☒	Ascending	0,05263	0,05263	1	1	41	0,00	22	4	1
EMPLOYEE	JOB_CODE, JOB_GRADE, JOB_CO...	☐	☒	Ascending	0,03846	0,03846	1	1	41	6,00	15	4	1
EMPLOYEE	DEPT_NO	☐	☒	Ascending	0,00000	0,05263	1	1	41	0,00	22	4	1
EMPLOYEE	JOB_CODE, JOB_GRADE, JOB_CO...	☐	☒	Ascending	0,00000	0,03846	1	1	41	6,00	15	4	1
EMPLOYEE	EMP_NO	☒	☒	Ascending	0,00000	0,02439	1	1	41	1,00	0	0	1
EMPLOYEE_PROJECT	EMP_NO	☐	☒	Ascending	0,00000	0,04545	1	1	28	1,00	6	2	1
EMPLOYEE_PROJECT	PROJ_ID	☐	☒	Ascending	0,00000	0,20000	1	1	28	0,00	23	9	1
EMPLOYEE_PROJECT	EMP_NO	☐	☒	Ascending	0,04545	0,04545	1	1	28	1,00	6	2	1
EMPLOYEE_PROJECT	PROJ_ID	☐	☒	Ascending	0,20000	0,20000	1	1	28	0,00	23	9	1
EMPLOYEE_PROJECT	EMP_NO, PROJ_ID	☒	☒	Ascending	0,00000	0,03571	1	1	28	9,00	0	0	1
IBE$LOG_BLOB_FIELDS	LOG_TABLES_ID	☐	☒	Ascending	0,00000	0,00000	1	1	0	0,00	0	0	1
IBE$LOG_FIELDS	LOG_TABLES_ID	☐	☒	Ascending	0,00000	0,00000	1	1	0	0,00	0	0	1

The SQL plan used by the InterBase/Firebird Optimizer merely shows how the server plans to execute the query.

If the developer wishes to override InterBase/Firebird's automatic index selection, and determine the index search sequence himself, this must be specified in SQL.

For example, an index is created in the `EMPLOYEE` database:

```
CREATE INDEX EMPLOYEE_IDX1 ON EMPLOYEE(PHONE_EXT);
```

Then:

```
SELECT * FROM EMPLOYEE
   WHERE EMPLOYEE.PHONE_EXT='250'
   PLAN (EMPLOYEE INDEX (EMPLOYEE_IDX1));
```

Each index needs to be named and entered individually.

To eliminate an index from the plan `+0` can be added in the query to the field where you wish the index to be ignored, thus denying the optimizer the ability to use that index for that particular query. This is much more powerful and flexible than deleting the index altogether, which prevents any use of it by the Optimizer in the future.

Indices should be prudently defined in a data structure, as not every index automatically leads to an acceleration in query performance. If in a table, for example, a column comprises data only with the value 0 or 1, an index could even slow performance down. A complex index structure can however have a huge influence upon insertion and alteration processes in the long run.

Please also refer to the Firebird 2.0.4 Release Notes chapter, *Enhancements to indexing* for improvements and new features in Firebird 2.0, and to the following subjects for further general information regarding indices.

See also:
Index

- SQL Editor / Plan Analyzer
- SQL Editor / Performance Analysis
- IBExpert Table Editor / Indices
- Firebird 2.0.4 Release Notes: Enhancements to Indexing
- Recompute selectivity of all indices
- Firebird for the database expert: Episode 1 - Indexes
- Recreating Indices 1
- Recreating Indices 2

Automating maintenance operations

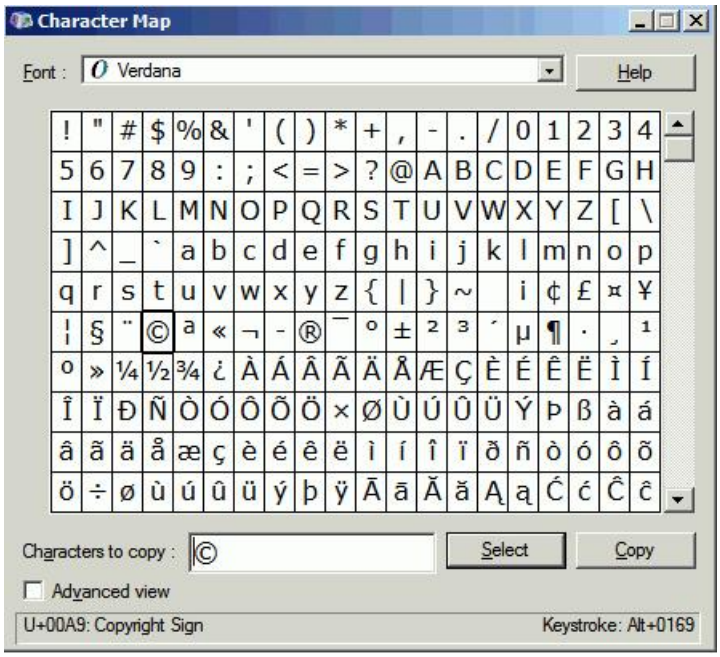
To calculate statistics for all indices, without any user intervention, simply run the following:

```
CREATE PROCEDURE REINDEX
AS
declare variable SQL VARCHAR(200);
BEGIN
  FOR
    select trim(rdb$index_name) from rdb$indices
    INTO :SQL
  DO
  BEGIN
    execute statement 'SET STATISTICS INDEX '||:sql;
  END
END
```

This should be executed regularly, particularly with databases undergoing a lot of manipulation (INSERTS, UPDATES, DELETES).

Ascending index

An ascending index searches according to an ascending letter or numeric sequence, depending upon the defined character set (or, if no character set has been specified for the indexed field, the default character set).



Descending index

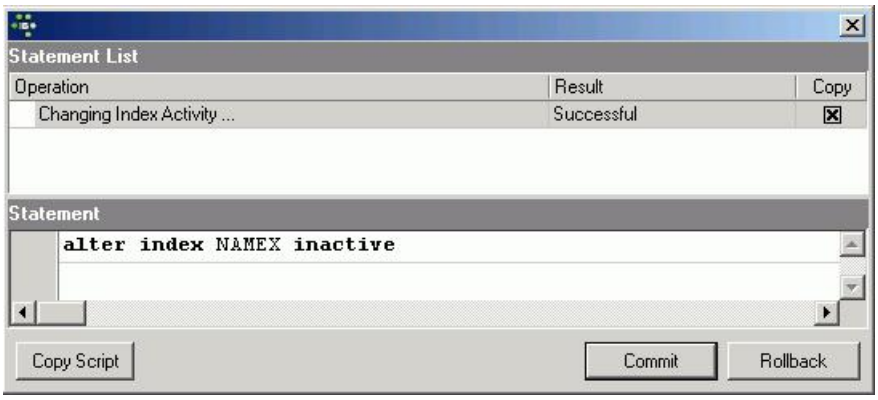
A descending index searches according to a descending letter or numeric sequence, depending upon the defined character set (or, if no character set has been specified for the indexed field, default character set).

Alter index

Once an index has been defined it is not possible to alter the following: indexed columns, sort direction or uniqueness constraints. The only way to change any of this information is to drop the index and then to recreate it (see Drop Index).

However the status of an index may be altered to active or inactive. An index should be deactivated when, for example, a large number of data sets are to be added, as an active index would recompute the index each time a data set is input. By deactivating the index, and then reactivating after all the data has been input, the index is only recomputed once.

This can be done simply and directly on the Table Editor / Indices page, by checking or unchecking the relevant boxes in the *Status* column, then compiling, using the respective Editor icon or [Ctrl + F9], and finally committing.



The SQL syntax is:

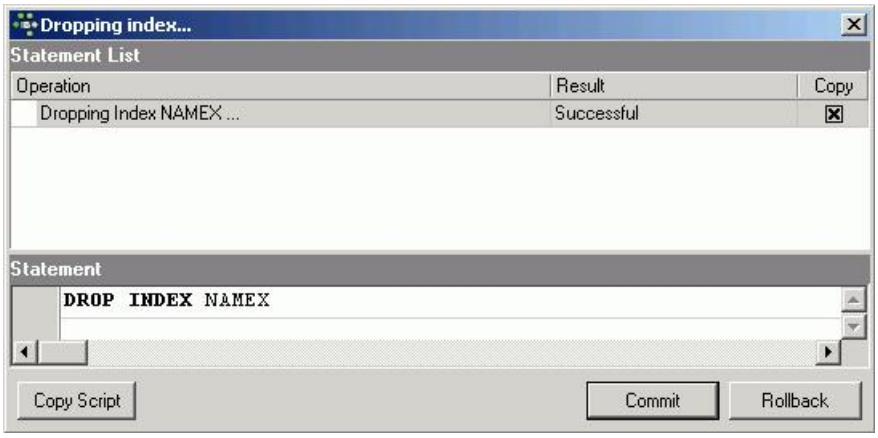
```
ALTER INDEX <index_name> ACTIVE | INACTIVE
```

An index can only be altered by the database creator or by the SYSDBA.

Drop index/Delete index

Only user-defined indices can be dropped. As the only alterations permitted on indices are activation and deactivation, indices often need to be dropped and then subsequently recreated, in order to alter certain index information such as indexed columns, sort direction or uniqueness constraints.

Indices can be dropped simply in IBExpert using the Table Editor / Indices page. Mark the index to be dropped and then right-click and select the menu item *Drop Index <INDEXNAME>* or use the [DEL] key:



Finally commit or roll back.

Using SQL the syntax is:

```
DROP INDEX Index_Name
```

`DROP INDEX` cannot be used for system-generated indices on primary or foreign keys, or on columns with a uniqueness constraint in the table definition.

An index can only be dropped by the database creator or by the SYSDBA.

See also:
Indices
Indexed reads/non-indexed reads
Database Statistics / Indices
Recompute selectivity of all indices
Firebird 2.0.4. Release Notes: Enhancements to indexing
Firebird for the database expert: Episode 1 - Indexes
Recreating Indices 1
Recreating Indices 2

Keys

In the relational model, key is used to organize data logically, so that a specific row can be uniquely identified. A key should not be confused with an index. An index is part of the table's physical structure on-disk. It is used to speed data access when queries are performed. Indices are therefore not a part of the relational model.

InterBase/Firebird automatically generates an index for primary and foreign key columns. On primary key columns, the index actually enforces the unique constraint required by the relational model. Links between tables usually occur on primary and foreign keys, so having an index on these columns ensures maximum performance.

Primary key

A primary key is a column (= simple key) or group of columns (= composite key/compound key) used to uniquely define a data set/row in the table. A primary key should always be defined at the time of defining a new table for each table. If you have a database that does not contain primary keys in all tables, and need to add these subsequently, please refer to *Adding primary keys to existing tables* below.

Relational theory states that a primary key should be designated for every table. It must be unique, and therefore cannot be `NULL`. It provides automatic protection against storing multiple values. In fact, without a primary key it is impossible to delete just one of two identical data sets. Each table can have only one designated primary key, although it can have other columns that are defined as `UNIQUE` and `NOT NULL`.

A primary key column is nothing other than a unique constraint complemented by a system index and the check constraint `NOT NULL`. Primary keys are always the preferred index of the InterBase/Firebird Optimizer.

When a data set is created or changed, Firebird/InterBase immediately checks the validity of the primary key. If the number already exists, a key violation results, and the storage process is immediately cancelled. Unfortunately InterBase/Firebird allows tables to be created without a primary key, which is a mistake. Data tables should always be keyed.

Existing primary keys and their system names can be viewed on the IBExpert Table Editor / Constraints page.

It is wise to keep the primary key as short as possible to minimize the amount of disk space required, and to improve performance. IBExpert recommends the use of an autoincrement generator ID number used as an internal primary key for all tables. For example, a simple `BIGINT` datatype generator not influenced in any way by any actual data. They do not need to be visible to the user as they are merely a tool to help the database work more efficiently and increase database integrity. One generator can be used as a source for all primary keys in a database, as the numbers do not need to be consecutive but merely unique. Each time a new data set is inserted, the generator automatically generates an ID number, regardless of the table name, for example, new `customer_id = 1`, new `order_id = 2`, new `orderline_id = 3`, new `orderline_id = 4`, new `customer_id = 5`, etc. A further advantage of such a single autoincrement generator primary key is that the database is perfectly prepared for replication; two or more servers can be connected and their data easily swapped, as the primary keys can be simply defined on both servers, e.g. server 1's generator should start at the value 1000000000 and server 2's at 2000000000 thus avoiding any conflict.

Although this method is unfortunately seldom used in the real world, it should be. Each primary key will only ever appear once in the database, which can be quite important in an OO (object-oriented) framework where there are so many objects floating around. They and you both need some unique identifier for the system to tell you what is behind the number, product, order etc.

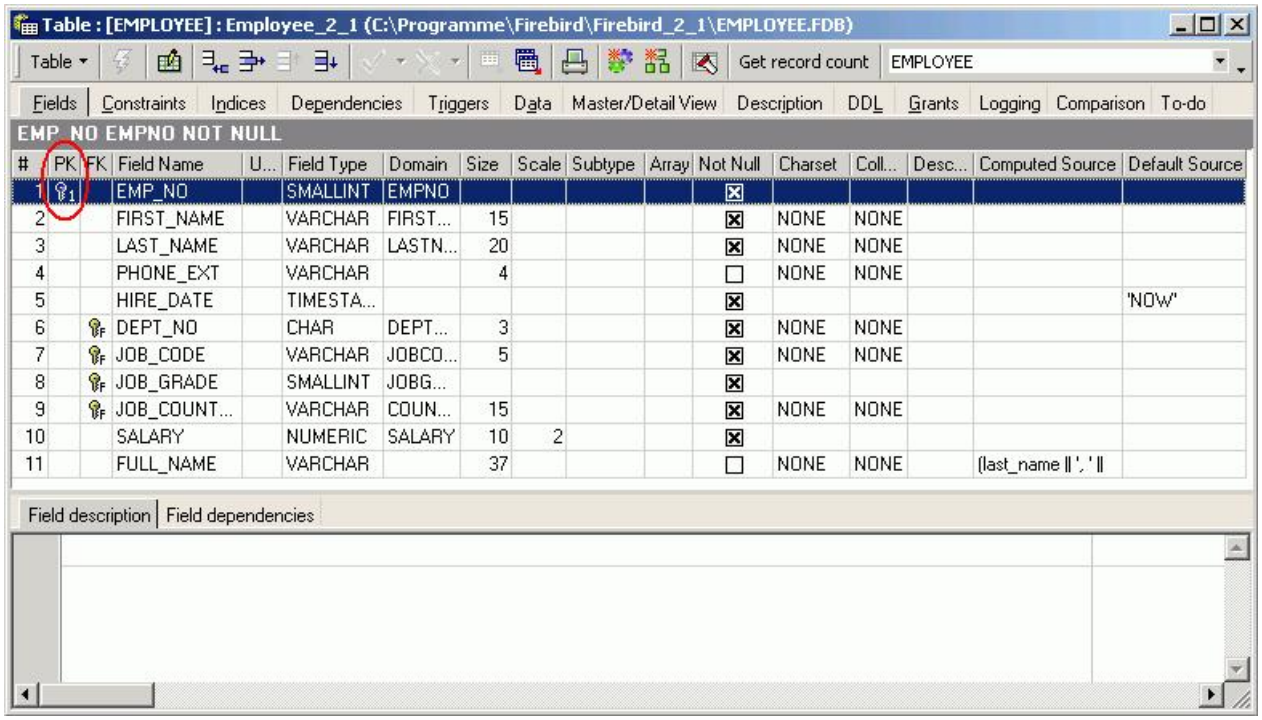
Composite keys are not recommended, as these always slow performance and the sequence of the fields concerned must be identical in all referenced tables.

Adding primary keys to existing tables

This article was written by Melvin Cox, and provides a method of defining primary keys on existing tables using IBExpert:

Here is a viable workaround for those of us who do not wish to spend an eternity exporting data, dropping and recreating multiple tables, and finally import the data back into those tables. Working with a Firebird 1.5 database (dialect 1) created via ODBC export from a Microsoft Access database, I have successfully defined primary keys on tables by taking the following steps:

1. Bring up the table within the IBExpert interface's Table Editor window (double-click on the respective table in the DB Explorer or use [Ctrl. + O]). The *Fields* page should be active.



2. Double click in the `NOT NULL` box corresponding to the field that you wish to designate as the primary key. This will call up the Edit Field dialog.
3. Check the `NOT NULL` option and select an existing or create a new domain.

4. Press OK and then, after checking the script produced by IBExpert, the *Commit* button. The field is now set to `NOT NULL`.
5. Bring up the SQL Editor: Tools / SQL Editor (or press [F12]).
6. Enter the following command:

```
ALTER TABLE table_name ADD PRIMARY KEY (field_name);
```

For example, to define a primary key on the `EVENTS` table enter:

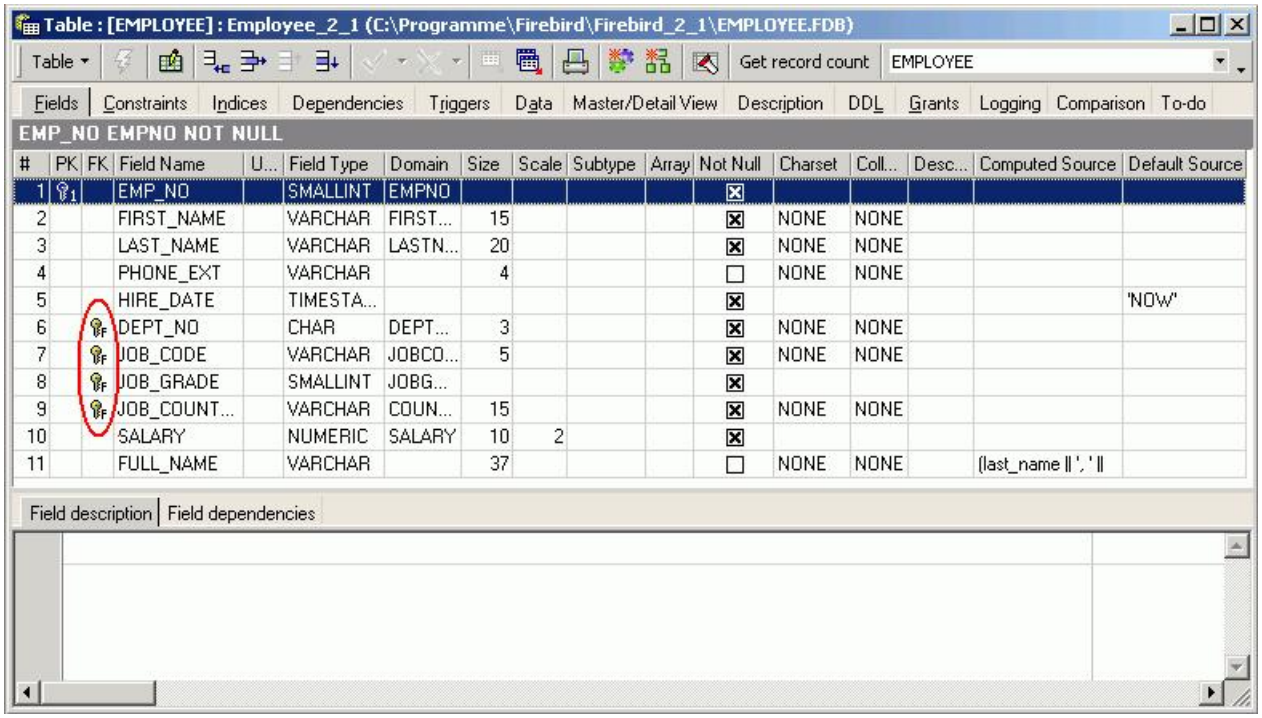
```
ALTER TABLE events ADD PRIMARY KEY (event_id);
```

7. Press the *Execute* Button or [F9].
8. Close the SQL Editor. This will call up the *Active Transaction Found* dialog. Select `Commit`.
9. Close the Table Editor window.
10. Reopen the Table Editor window [Ctrl. + O]. The newly defined primary key will now be visible.

Foreign key

A foreign key is composed of one or more columns that reference a primary key. *Reference* means here that when a value is entered in a foreign key, Firebird/InterBase checks that the value also exists in the referenced primary key. This is used to maintain domain integrity.

A foreign key is vital for defining relationships in the database. It can be specified in the IBExpert Table Editor (started from the DB Explorer) on the Constraints page.



Foreign keys are used mainly for so-called reference tables. In a table storing, for example, employees, it needs to be determined which department each employee belongs to. Possible entries for the department number of each `EMPLOYEE` data set are contained in the `DEPARTMENT` table. As the `EMPLOYEE` table refers to the `DEPT_NO` as the primary key for the `DEPARTMENT` table, there is a foreign key relationship between the `EMPLOYEE` table and the `DEPARTMENT` table. Foreign key relationships are automatically checked in Firebird/InterBase, and data sets with a non-existent department number cannot be saved.

When a primary key:foreign key relationship links to a single row in another table, what is known as a virtual row is created. The columns in that second table provide additional description about the primary key of the first table. This is also know as a 1:1 relationship.

A foreign key can also point to itself. Firebird enables you to reference recursive data and even represent tree structures in this way.

Foreign keys and their system names can be defined and viewed on the IBExpert Table Editor / Constraints page.

A primary key does not have to reference a foreign key. However a unique index is insufficient; a unique constraint needs to be defined (this definition also causes a unique index to be automatically generated).

When defining a foreign key, it is necessary to specify update and delete rules. Please refer to *Referential integrity* and *Cascading referential integrity* for further information.

SQL syntax:

```
ALTER TABLE MASTER
ADD CONSTRAINT UNQ_MASTER UNIQUE (FIELD_FOR_FK);
```

Foreign key names are limited to 32 characters up until InterBase 6 and Firebird 1.5; InterBase 7 allows 64 characters. IBExpert therefore recommends limiting table names to 14 characters, so that the foreign key name can include both related table names: prefix `FK` plus two separators plus both table names, e.g. `FK_Table1_Table2`.

Please note however that this is not an InterBase/Firebird restriction, but purely an IBExpert recommendation to enable a clear and logical naming convention for foreign keys.

Note: if data has already been input in a table which is to subsequently be assigned a foreign key, this will not be allowed by InterBase/Firebird, as it violates the principle of referential integrity. It is however possible to filter and delete the old data (where no reference to a primary key has been made) using a `SELECT` statement and committing. It is important to then disconnect and reconnect the database in IBExpert, for this to work.

New to Firebird 2.0: Creating foreign key constraints no longer requires exclusive access - Now it is possible to create foreign key constraints without needing to get an exclusive lock on the whole database.

Candidate key

Any column or group of columns which can uniquely identify a data set, and can therefore be considered for use as a primary key. It is always `NOT NULL` (i.e. must not be left undefined), and unique.

Simple key

A simple key is composed of one column only, i.e. a single column is designated as a table's primary key.

Composite key/compound key

A composite key consists of two or more columns , designated together as a table's primary key. Multiple-column primary keys can be defined only as table-level constraints:

Single-column primary keys can be defined at either the column or the table level (but not both). For example, the following code states that the table's primary key consists of three columns, `JOB_CODE`, `JOB_GRADE`, and `JOB_COUNTRY`. Neither of these columns is required to be unique by itself, but their combined value must be unique (and `NOT NULL`).

```
CREATE TABLE
COLUMN_defs ...
PRIMARY KEY (JOB_CODE,JOB_GRADE,JOB_COUNTRY);
```

Unfortunately such keys have two huge disadvantages: firstly they slow the database performance considerably, as InterBase/Firebird needs to check all contents of all columns designated in such a composite key; secondly the sequence of the fields concerned must be identical in all referenced tables.

Basically composite keys should be avoided! It is much preferable to use an internal ID key (so-called artificial key) as the primary key for each table.

Unique

Unique fields are unequivocal, unambiguous, one-of-a-kind (i.e. there is no duplicate information allowed in the data sets of a unique field). Such fields must therefore also be `NOT NULL`.

Unique fields are given a unique index. Each unique field is a candidate key.

Artficial key/surrogate key/alias key

An artificial or alias or surrogate key is created by the database designer/developer if there is no candidate key, i.e. no logical, simple field to be the primary key. An artificial key is a short ID number used to uniquely identify a record.

Such an internal primary key ID is recommended for all tables. They should always be invisible to the user, to prevent any potential external influence regarding their appearance and composition.

It is always wise to keep the primary key as short as possible to minimize the amount of disk space required, and to improve performance; therefore artificial keys should also be as short as possible. An ideal solution for the generation of an artificial key is the use of an autoincrement generator ID number.

IBExpert recommends this solution be used as an internal primary key for all tables.

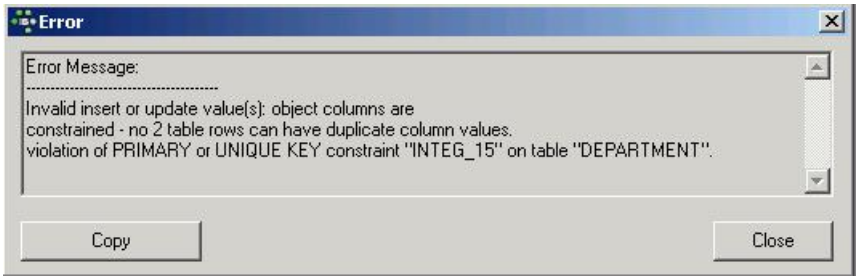
Usually such an artificial/alias/surrogate key is just an autoincrement integer field so that each record has it's own unique integer identifier. For example:

```
CREATE TABLE CUSTOMERS (
   CUSTOMER_ID INTEGER NOT NULL,
   FIRST_NAME VARCHAR(20),
   MIDDLE.NAME VARCHAR(20),
   LAST_NAME VARCHAR(20);
...);
```

In this case `CUSTOMER_ID` the artificial or surrogate key.

Key violation

When a data set is created or changed, InterBase/Firebird immediately checks the validity of the primary key. If the number already exists, or the field has been left blank, a key violation results, and the storage process is immediately cancelled.



InterBase/Firebird immediately sends an error message referring to the violation of a unique or primary key constraint.

Referential integrity

The relationship between a foreign key and its referenced primary key is the mechanism for maintaining data consistency and integrity. Referential integrity ensures data integrity between tables connected by foreign keys. A foreign key is one or more columns that reference a primary key, i.e. when a value is entered in the foreign key, InterBase/Firebird checks that this value also exists in the referenced primary key, so maintaining referential integrity.

Referential integrity can occur in the following three cases:

1. In the master table a data set is deleted. For example, the deletion of a customer, for whom there are still existing orders could lead to order data sets without a valid customer number. This could falsify analyses and lists, as the internal relationships no longer appear. The prevention of data set deletion in the master table, when data sets still exist in the detail table, is called prohibited deletion. The relay of deletions to all detail tables is called cascading deletion.

2. The primary key is changed in the master table. For example a customer is given a new customer number, so that all orders relating to this customer need to also relate to the new customer number. This is known as a cascading update.

3. A new data set is created, and the foreign key does not exist in the master table. For example an order is input with a customer number not yet allocated in the master table. A possible solution could be the automatic generation of a new customer. This is called a cascading insert.

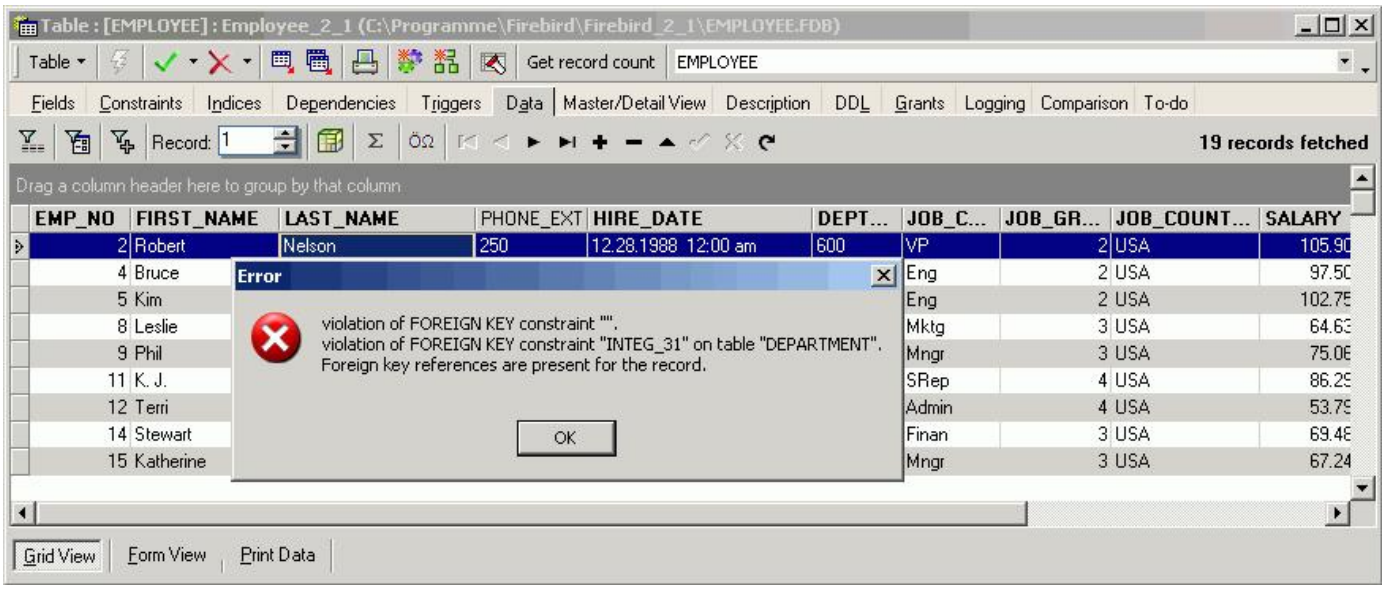
Referential integrity is supported natively in InterBase/Firebird, i.e. all foreign key basic relationships are automatically taken into consideration during data alterations. Since Version 5, InterBase supports declarative referential integrity with cascading deletes and updates. In older versions, this could be implemented with triggers.

Cascading referential integrity

Since InterBase v5/Firebird, cascading referential integrity is also supported.

When a foreign key relationship is specified, the user can define which action should be taken following changes to, or deletion of its referenced primary key. `ON UPDATE` defines what happens when the primary key changes and `ON DELETE` specifies the action to be taken when the referenced primary key is deleted. In both cases the following options are available:

1. **`NO ACTION`:** throws an exception if there is a existing relationship somewhere in another table:



1. **`CASCADE`:** the foreign key column is set to the new primary key value. A very handy function when it comes to updating, as all referenced foreign key fields are automatically updated. When deleting the `CASCADE` option also deletes the foreign key row when the primary key is deleted. Be extremely careful when using `CASCADE ON DELETE`; when you delete a customer, you delete his orders, order lines, address, everything where there is a defined key relationship. It is safer to write a procedure that ensures just those data sets necessary are deleted in the right order.
2. **`SET NULL`:** if the foreign key value is allowed to be `NULL`, when a primary key value is deleted, it will set the relevant foreign key fields referencing this primary key value also to `NULL`.
3. **`SET DEFAULT`:** the foreign key column is set to its default value when a primary key field is deleted.

Table Editor

The Table Editor can be used to analyze existing tables and their specifications, or to add new fields, specifications etc, in fact, perform all sorts of table alterations. It can be started directly from the DB Explorer by simply double-clicking on the relevant table in the IBExpert DB Explorer, or using the DB Explorer right-click menu *Edit Table ...* (key combination [Ctrl + O]).

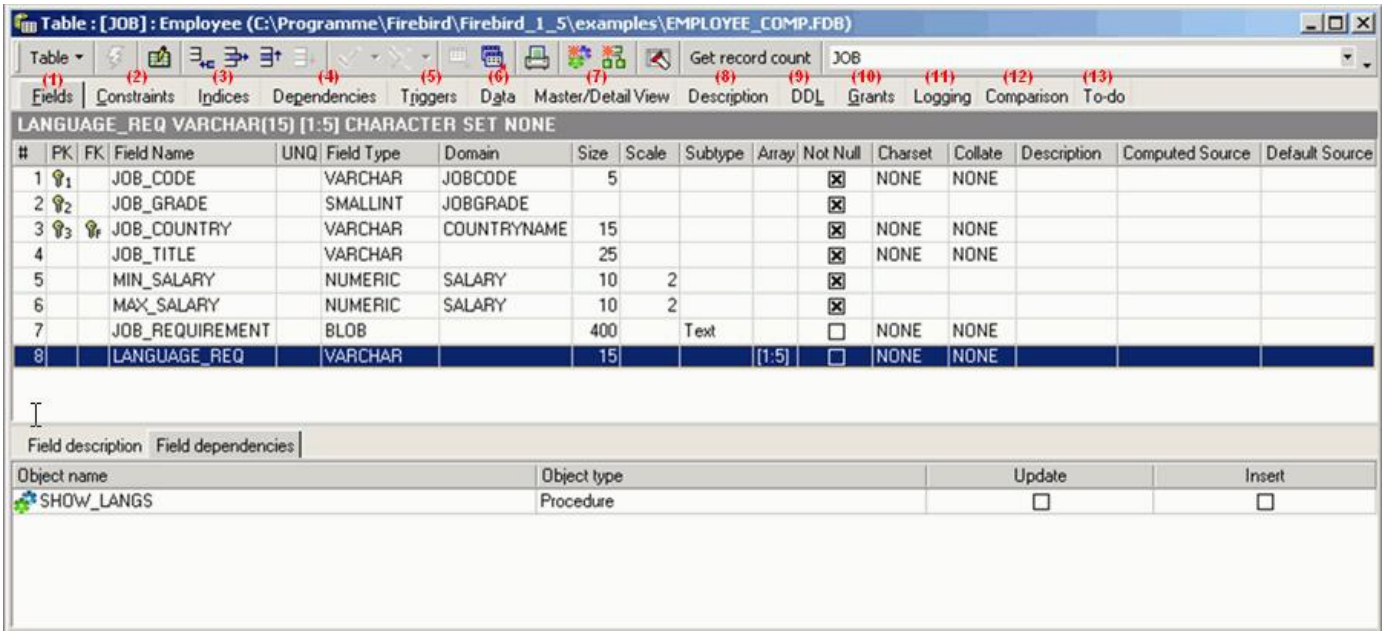
The Table Editor comprises a number of pages, opened by clicking the corresponding tab heading, each displaying those properties already specified, and allowing certain specifications to be added, altered or deleted.

Note: the IBExpert status bar shows how many remaining changes may be made to the table before a backup and restore is necessary. (A total of 255 changes may be made to a database object before InterBase/Firebird demands a backup and restore).

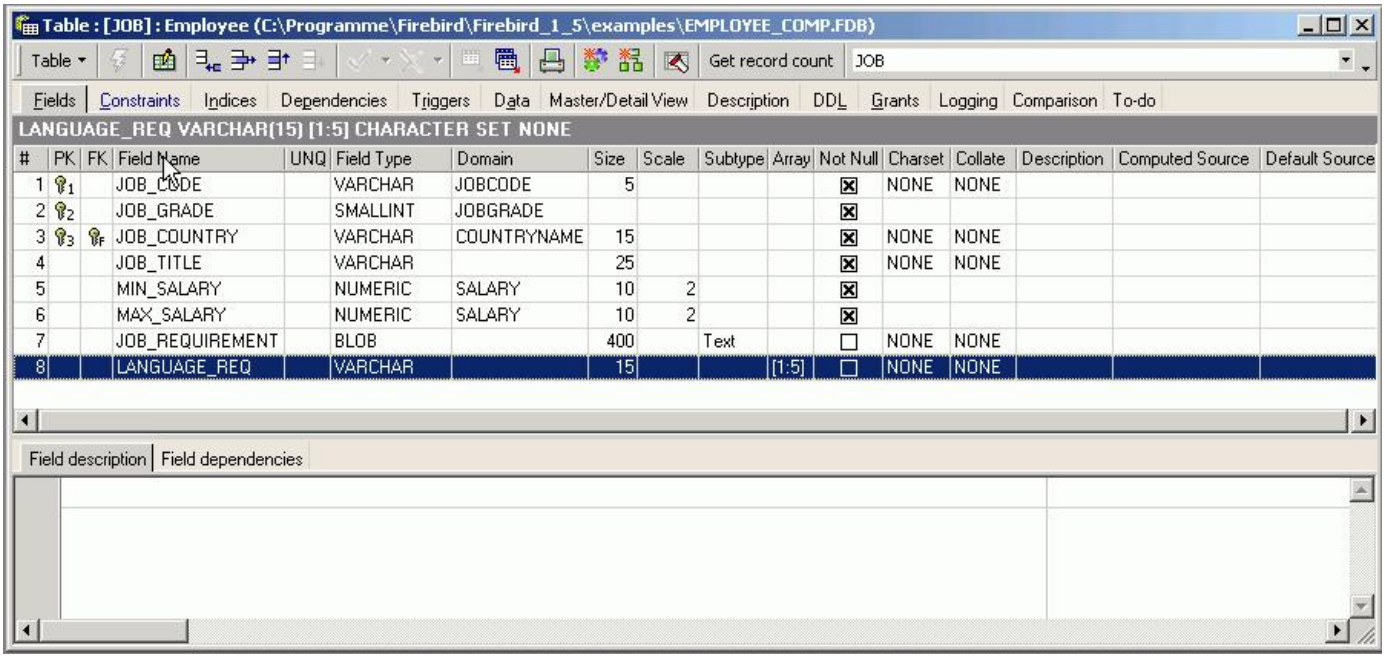
The *Get Record Count button* at the right of the Table Editor toolbar, displays the number of records in the table. To the right of this the table name is displayed. By clicking on the drop-down list, all tables for the connected database can be viewed and selected.

Alternatively for those competent in SQL - the SQL Editor [F12] can be used directly for making table alterations using SQL code.

Support for the InterBase 7.5 temporary tables feature was added in IBExpert version 2004.12.12.1.



(1) Fields



The many possible field specifications are listed on the *Fields* page. The individual columns are explained in detail under New Table. Fields can be amended by simply overwriting the existing specification where allowed. Please note that it is not always possible to alter certain fields once data has been entered, e.g. a field cannot be altered to `NOT NULL`, if data has already been entered which does not conform to the `NOT NULL` property (i.e. the field has been left undefined). Similarly a primary key cannot be specified following data entries with duplicate values.

Since IBExpert version 2005.08.08 the `NOT NULL` checkbox is now checked when a field itself has not a `NOT NULL` flag and is based on a `NOT NULL` domain.

New in IBExpert version 2.5.0.61: It is possible to drag 'n' drop fields from the Database Explorer tree and SQL Assistant into the Table Editor's field list, allowing you to quickly and easily copy field definitions from one table to another.

The contents of text blob fields can even be read in the IBExpert Table Editor; simply hold the mouse over the text field, and the full text appears.

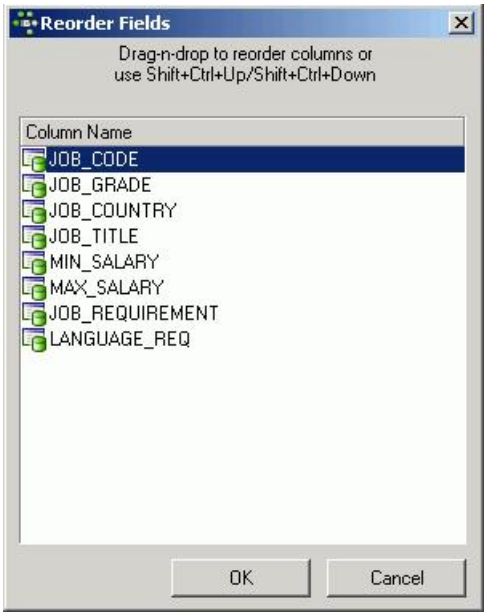
Tip: as with all IBExpert dialogs, the fields can be sorted into ascending or descending order based upon the column where the mouse is, simply by clicking on the column headers (i.e. PK, FK, Field Name etc.).By double-clicking on the right edge of the column header, the column width can be adjusted to the ideal width.

Since IBExpert version 2003.11.6.1 the new Grid menu offers a number of options when working in the Table Editor's Field? and Data pages.

Table Editor right-click menu

The Table Editor *Fields* page has its own context-sensitive menu using the right mouse button:

This can be used to add a New Field, or edit or drop an existing highlighted field. Fields can also be reordered using drag 'n' drop:



or key combinations [Shift + Ctrl + Up] and [Shift + Ctrl + Down] in the *Reorder Fields* window, or directly on the *Fields* page in Table Editor using the field navigator icons in the Navigation toolbar or previously mentioned key combinations.

A field list can also be copied to clipboard, and the pop-up *Description Editor* blended in or out.

New fields can be added using the

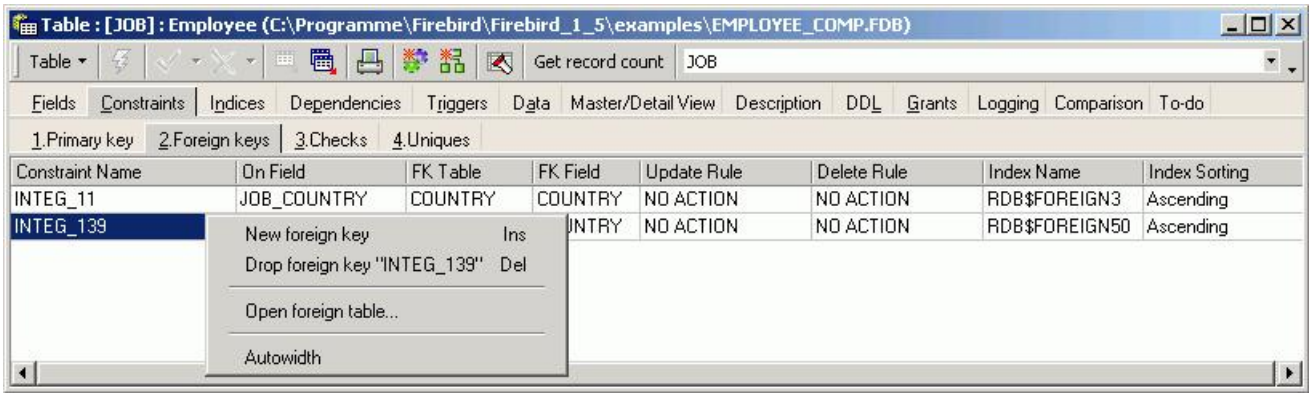
icon (or [Ins] key), to open the *Adding New Field Editor* (please refer to *Insert Field* for details).

Important! Do not forget to commit the transaction following creation, alteration or deletion of a field on the *Fields* page, otherwise the field alterations will not be displayed on the Data page, or any other Table Editor page for that matter.

In the lower part of the Table Editor the individual *Field Descriptions* and *Field Dependencies* can be viewed. Since IBExpert version 2003.11.6.1, the field dependencies list also includes indices, primary and foreign keys. This new version also enables you to alter the default value of a field.

(2) Constraints

Constraints are used to ensure data integrity. Constraints give out database the extra integrity it needs. Each constraint has its own context-sensitive right mouse button menu, and a new toolbar is displayed offering the most common operations as shortcuts.



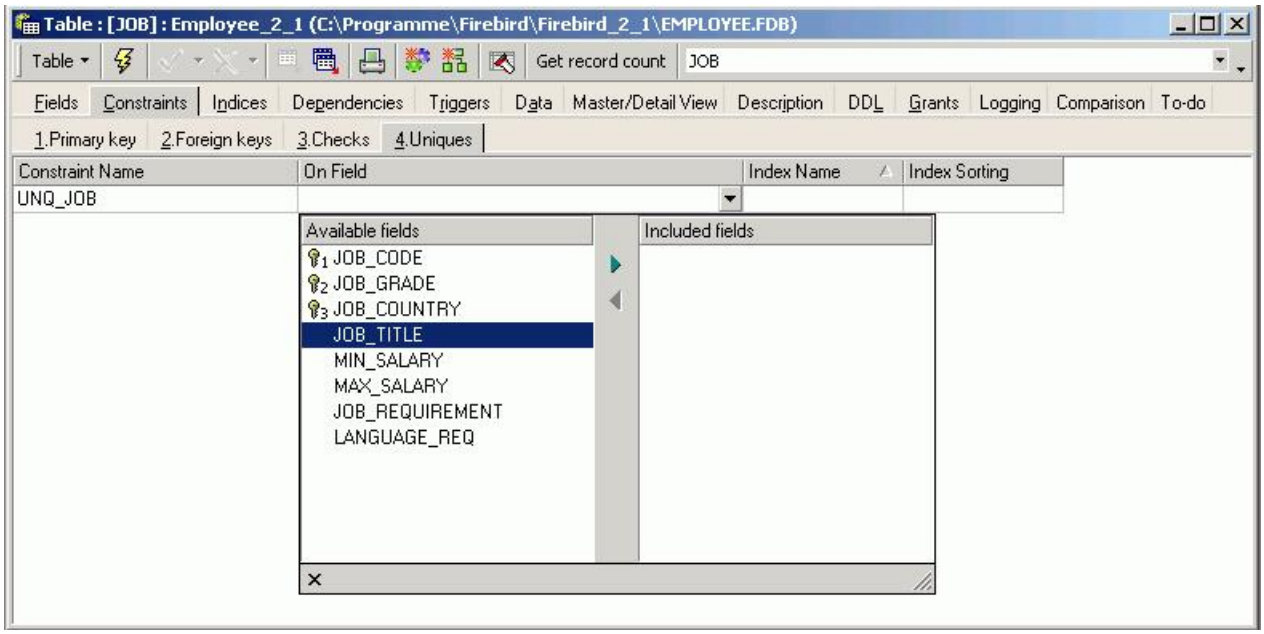
The right-click menu for the *Foreign Key* page offers, for example, *New Foreign Key* [Ins], *Drop Foreign Key* [Del], *Open foreign table ...* and *Autowidth*. *Autowidth* automatically adjusts the column widths to fit into the visible dialog width.

The following can be viewed, added or edited in the Table Editor under the *Constraints* tab:

- Primary keys: A primary key can officially only be defined at the time of defining a new table. There is however a workaround in IBExpert, should you ever find yourself in the situation, where you need to add a primary keys to existing tables (please refer to *Adding primary keys to existing tables*).
- Foreign keys: A foreign key is a link to another table and stores the primary key of another table. When defining a foreign key relationship, it is necessary to specify what should happen to the foreign key, if the primary key is updated or deleted. Please refer to *Referential integrity* and *Cascading referential integrity* for further information.
- Checks: Further conditions can be specified by the user (check constraint). *Checks* allows you to add a simple piece of logic to that every time you change that table, it's checked for validity. It's a way to be able to associate values on the same row. It is possible to define field constraints, e.g. the

value in the PRICE field must be larger than 0 and smaller than 10,000. It is also possible to define table constraints in this way (e.g. delivery date > order date).

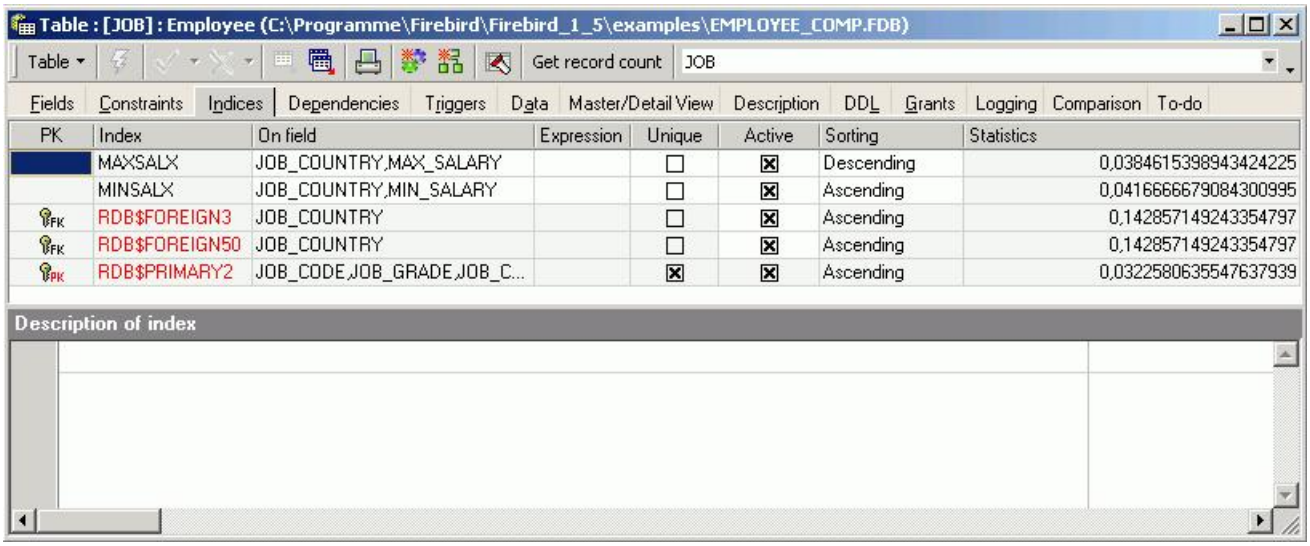
- Uniques: All fields defined as unique are also candidate keys. To define a field as unique in IBExpert, right-click on the *Constraints / Unique* page, and specify *New unique constraint*. Either accept or alter the default name UNQ_TABLENAME, and then click the drop-down list in the *On Field* column to select the field(s) you wish to specify as unique.



New to IBExpert version 2003.11.6.1 is the added support for the Firebird feature - user-defined constraint index names. And since IBExpert version 2005.01.12.1 the maximum constraint name length was expanded from 27 to 31.

(3) Indices

Indices already defined for the table can be viewed on the *Indices* page.



Information displayed includes key status, index name, upon which field the index has been set, whether it is unique, the status (i.e. whether *active* or *inactive*), which sorting order (*Ascending* or *Descending*) and the *Statistics* (displayed in older versions under the column heading *Selectivity*). Since IBExpert version 2007.05.03 it is also possible to define an index description. Those indices beginning with RDB$, are InterBase/Firebird system indices.

Indices can be added or deleted using the right-click menu or [Ins] or [Del]. However, instead of deleting indices, we recommend deactivating them (simply uncheck the *Deactivate* box by double-clicking) - you never know when you may need them again at a future date. System indices cannot be deleted.

Further options offered in the right mouse button menu are:

- Recompute Selectivity
- Recompute All
- Show Statistics (blends the selectivity statistics in and out)
- Copy index name

Expression indexes are also possible since Firebird 2.0. Arbitrary expressions applied to values in a row in dynamic DDL can now be indexed, allowing indexed access paths to be available for search predicates that are based on expressions.

Syntax

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]] INDEX <index name>
ON <table name>
COMPUTED BY ( <value expression> )
```

Example

```
CREATE INDEX IDX1 ON T1
   COMPUTED BY ( UPPER(COL1 COLLATE PXW_CYRL) );
COMMIT;
/**/
SELECT * FROM T1
   WHERE UPPER(COL1 COLLATE PXW_CYRL) = 'ÔÛÂÀ'
-- PLAN (T1 INDEX (IDX1))
```

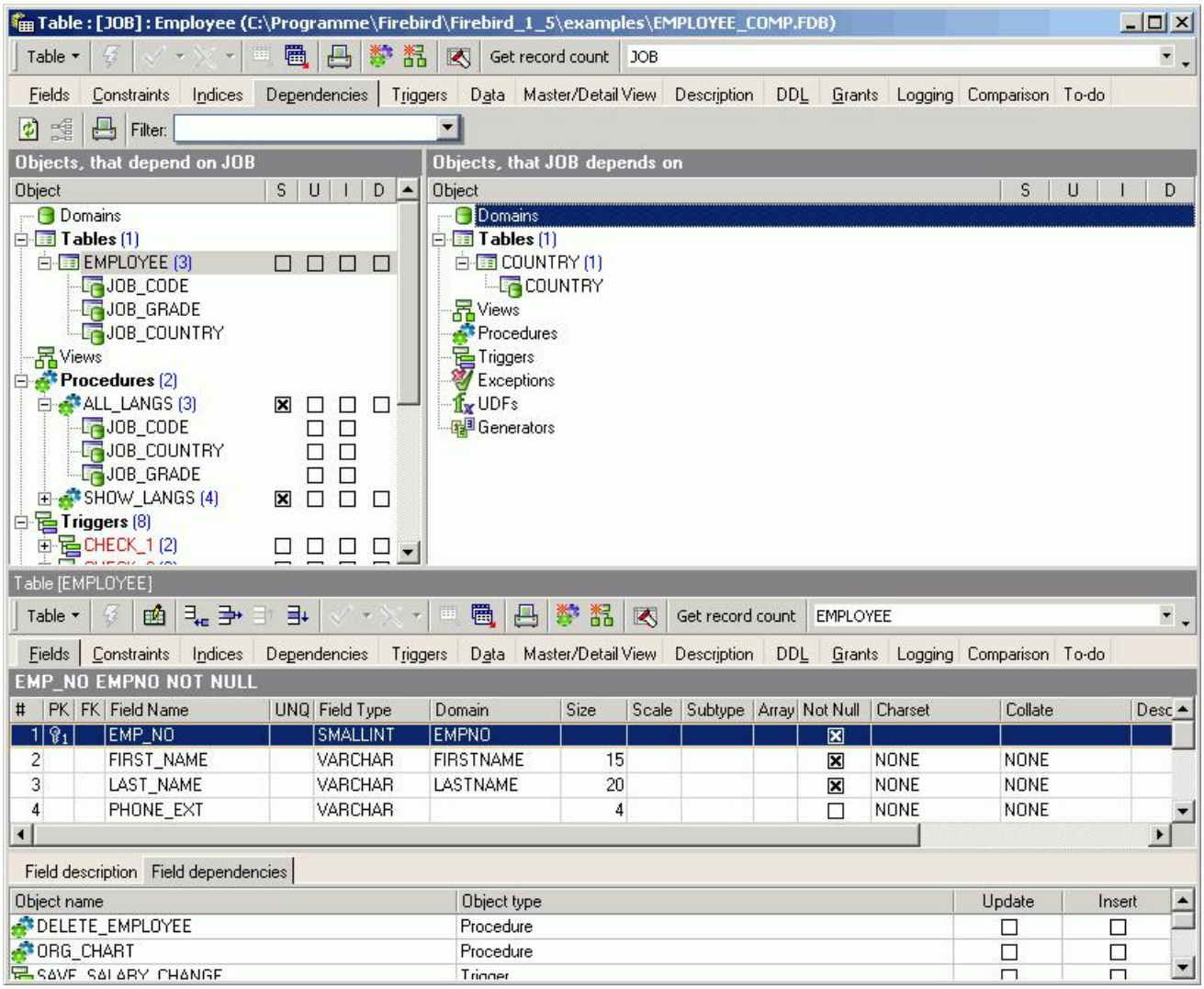
Please refer to the *Firebird 2.0.4. Release Notes* chapter, *Enhancements to indexing* for further index improvements in Firebird 2.0.

Although it is possible to set an index on multiple columns, this is not recommended, as an index on two fields simultaneously only makes sense when both fields are to be sorted using `ORDER BY`, and this should only be used on relatively small quantities of results as they can actually worsen performance rather than improve it.

See also:
Index
Alter index
Drop index
Firebird 2.0.4 Release Notes: Enhancements to indexing

(4) Dependencies

Here the dependencies between database objects can be viewed.



This summary can, for example, be useful if a database table should need to be deleted or table structures altered, or for assigning user rights to foreign key referenced tables. It displays both those objects that are dependent upon the table (left side), and those objects that the table depends upon (right side).

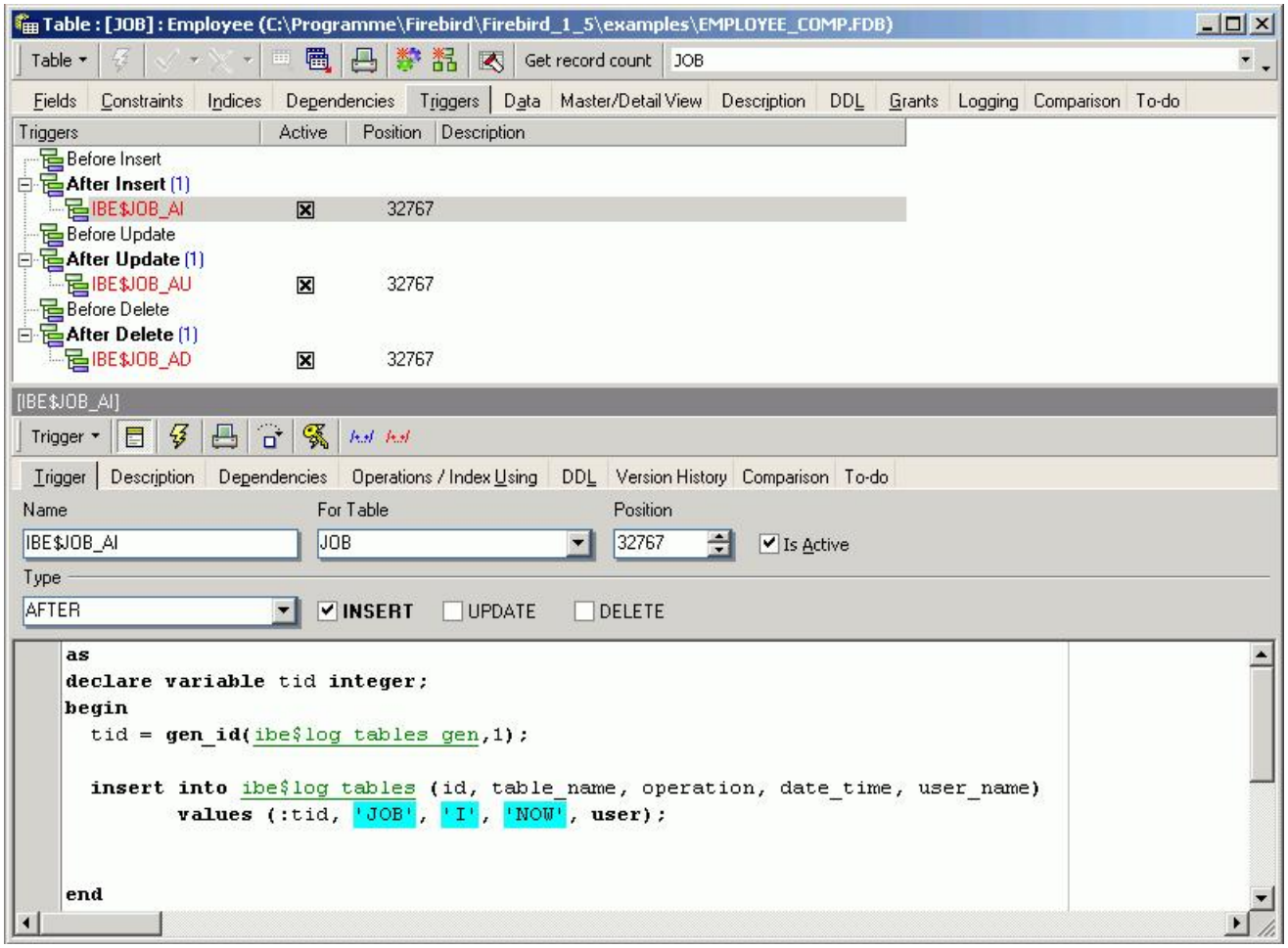
The object tree can be expanded or collapsed by using the mouse or [+] or [-] keys, or using the context-sensitive right-click menu items *Expand All* or *Collapse All*.

It even shows the actions (when blended in using the right mouse button menu item *ShowActions*) - `S` (=`SELECT`), `U` (=`UPDATE`), `I` (=`INSERT`) or `D` (=`DROP`).

The object code can be viewed and edited in the Table Editor lower panel, provided the *Inplace Objects' Editors* option has been checked in the IBExpert Options menu item Environment Options / Tools. If this option is not checked, then the code may only be viewed in the lower panel, and the object editor must be opened by double-clicking on the respective object name, in order to make any changes to it. This also applies to all triggers listed on the Triggers page.

(5) Triggers

Triggers are SQL scripts, which are executed automatically in the database when certain events occur.



Similar to dependencies, the triggers are listed in a tree structure according to the following events:

```
BEFORE INSERT
 AFTER INSERT
BEFORE UPDATE
 AFTER UPDATE
BEFORE DELETE
 AFTER DELETE
```

The object tree can be expanded or collapsed by using the mouse or [+] or [-] keys (or using the right-click menu).

When a trigger is highlighted, the right mouse button menu offers options to create a new trigger, edit or drop the highlighted trigger, or set the marked trigger to inactive/active.

IBExpert version 2007.12.01 introduced the option to set more than one trigger simultaneously as *active/inactive*.

The trigger code can be viewed and edited in the Table Editor lower panel, provided the *Inplace Objects' Editors* option has been checked in the IBExpert Options menu item Environment Options / Tools. If this option is not checked, then the code may only be viewed in the lower panel, and the Trigger Editor must be opened by double-clicking on the respective trigger name, in order to make any changes to the trigger.

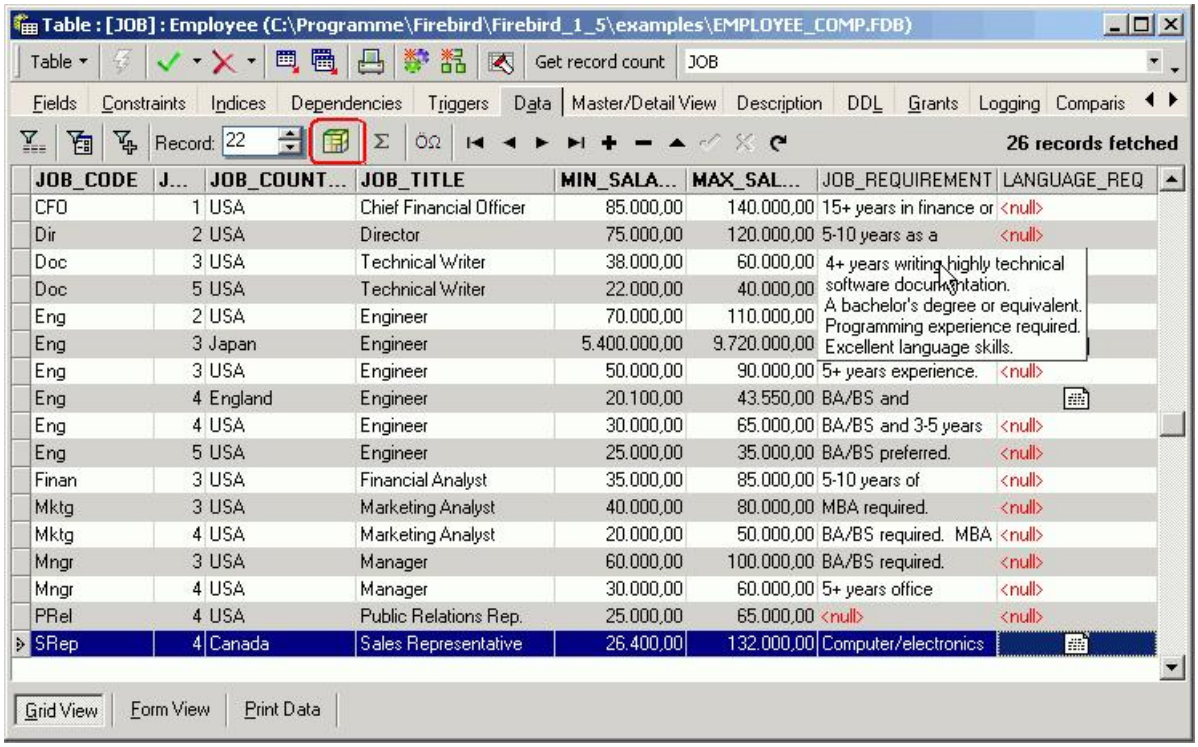
This also applies to all objects listed on the Dependencies page.

(6) Data Grid

Here the data in the database table can be manipulated (i.e. inserted, altered or deleted) directly.

There are three modes of view:

1. Grid View - all data is displayed in a grid (or table form).



The data sets can be sorted according to any field in either ascending or descending order by simply clicking on the column header. New data sets can also be added, altered and deleted here. And all operations, as with any operations performed anywhere in IBExpert, may be monitored by the SQL Monitor (started from the IBExpert Tools menu), particularly useful, should problems be encountered with SIUD operations.

The contents of blob and memo fields can be read by simply holding the cursor over the respective field. IBExpert displays them as a Blob value; it is also possible to view and edit them in the Blob Editor (HEX format).

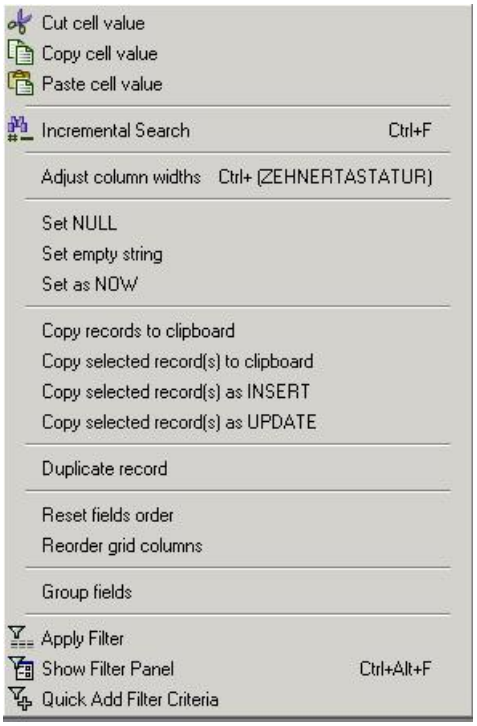
A new feature in IBExpert version 2004.10.30.1 is the OLAP and data warehouse tool, Data Analysis, opened using the *Data Analysis* icon (highlighted in red in the above illustration).

There are many options to be found under Options / Environment Options / 6. Grid, which allow the user to customize this grid. Under the IBExpert menu item Register Database or Database Registration Info there are additional options, for example, *Trim Char Fields in Grids*.

Since IBExpert version 2003.11.6.1 the new Grid menu offers a number of options when working in the Table Editor's Field and Data pages.

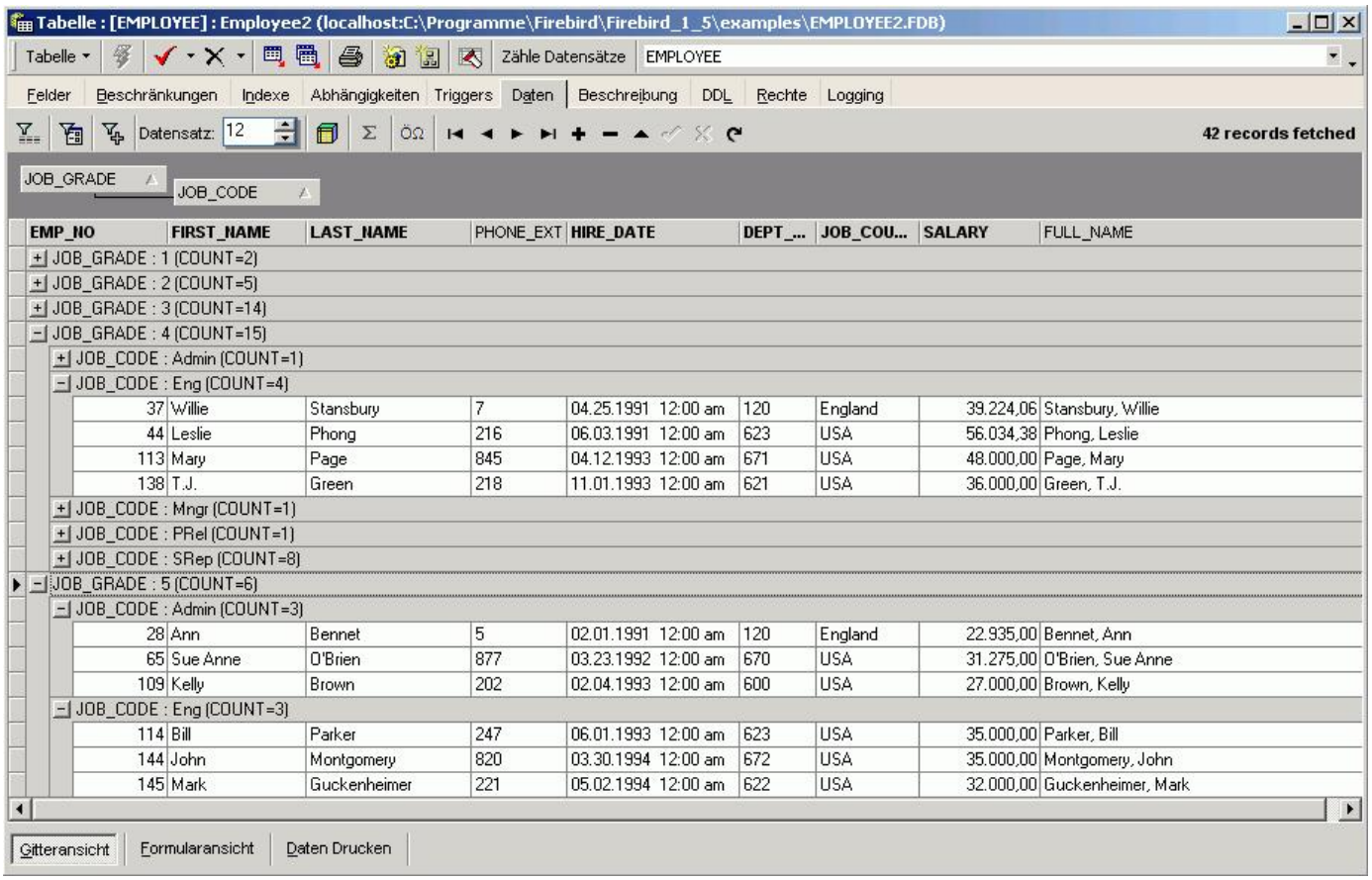
The *Data* page *Grid View* also has its own context-sensitive menu, opened by right-clicking with the mouse.

This includes the following options:



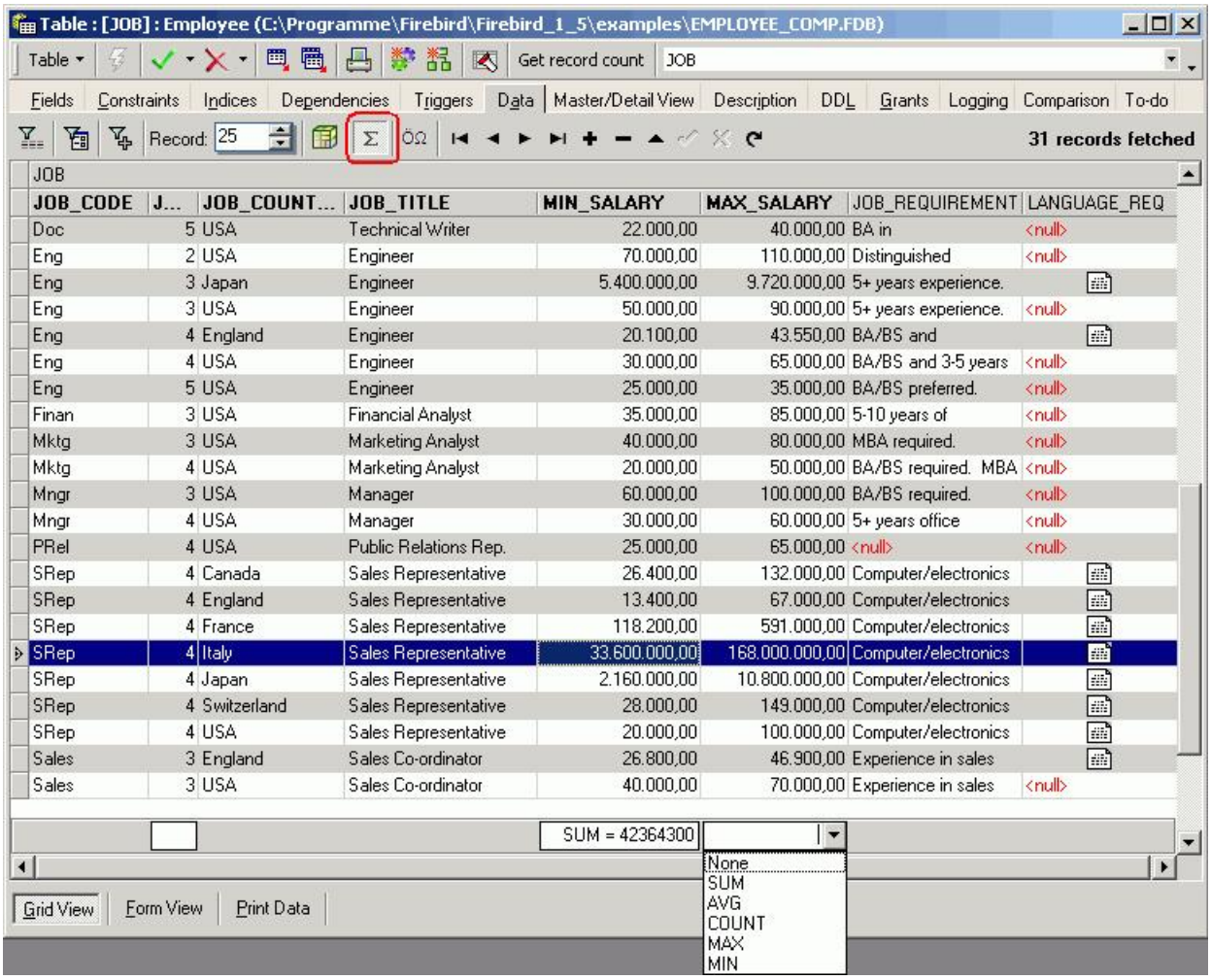
- **Cut, Copy and Paste** functions.
- **Incremental Search** [Ctrl + F] allows a quick search for individual entries by simply marking the desired column header, clicking the right mouse button menu item *Incremental Search* [Ctrl + F] and then typing the relevant digits/letters, until the required dataset(s) is/are found.

- **Adjust Columns widths** (or [Ctrl + "+" NUMBLOCK] adjusts all column widths in the grid view to the ideal width.
- **SET commands:** set field as NULL, empty or NOW.
- **Copying operations:** copies all or one or more selected records to clipboard, as INSERT or as UPDATE. Multiple records may only be selected if the *AllowMultiselect* option has been checked in the Options menu: Environment Options / Grid.
- **Duplicate record** option.
- **Reset fields order:** returns the field order to the original (not available in SQL Editor / Results).
- **Reorder grid columns:** simply using drag 'n' drop.
- **Group/Ungroup Fields:** offers an alternative visual option, allowing grid columns to be grouped, which is sometimes useful, for example, if you need to execute a complex query with joins of many tables. The *Grouping* feature is displayed as a dark gray bar labelled *Drag a column header here* to group by that column, displayed directly above the column headers over the grid. Should this not be visible, go to the IBExpert Options menu item Environment Options / Grid, and ensure that the *Allowrecords grouping* option is checked. The column header simply needs to be dragged and dropped onto the gray bar, to group by that column. A reorganized data view appears, where the group contents can be revealed or hidden, by clicking on the + or - buttons (see illustration below).
- **Filter options:** these can also be found in the data page toolbar (see below).



Both the *Grid* and *Form Views* offer a Navigation toolbar, allowing the data to be moved, inserted, altered and deleted. Furthermore data can be filtered using the Filter Panel toolbar. (Please refer to Filter Panel for further information.)

Since IBExpert version 2004.8.5.1 there is the added option to calculate aggregate functions (COUNT, SUM, MIN, MAX, AVG) on numeric and datetime columns. Simply click *Showsummary footer* button on the toolbar of the data view to display the summary footer:

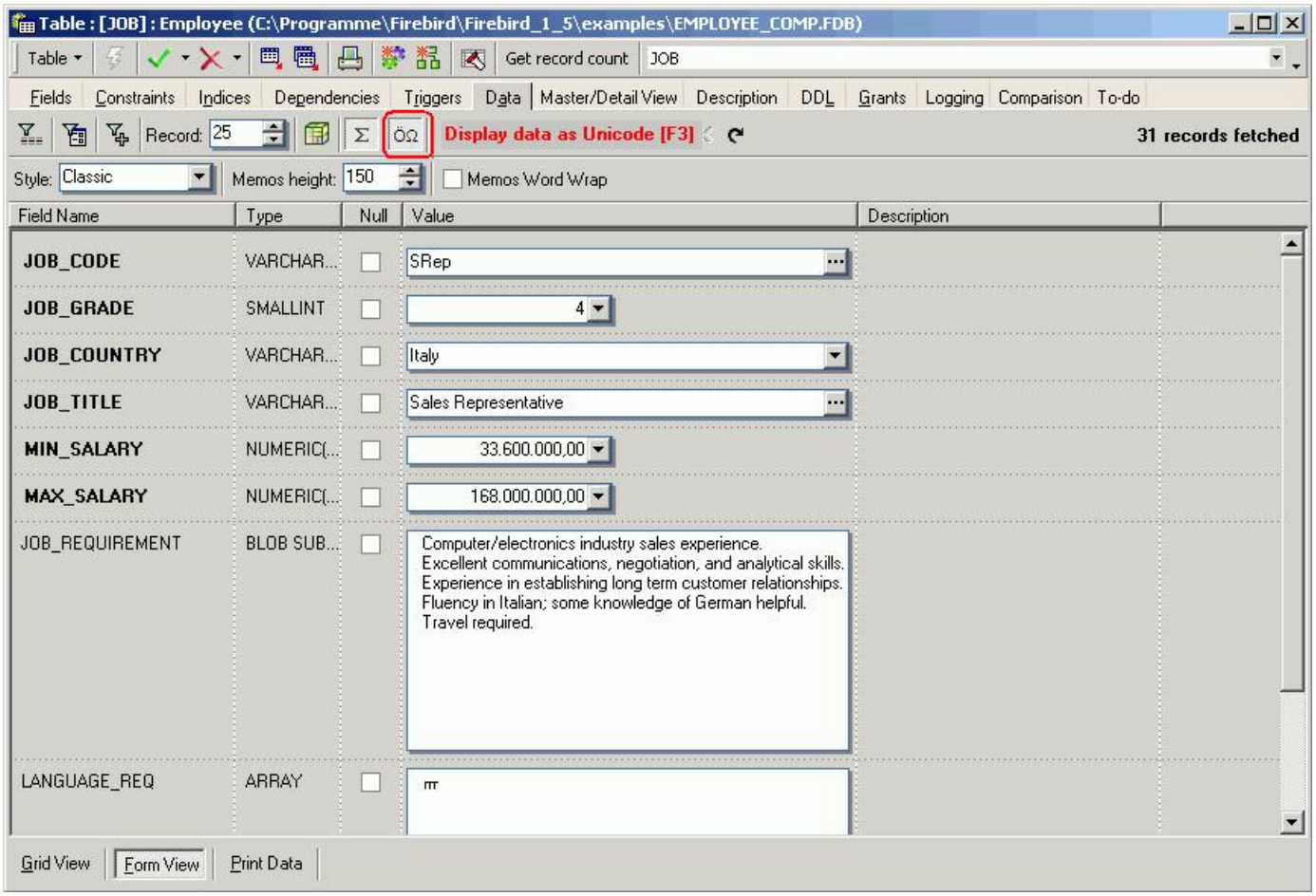


It is then possible to select an aggregate function for each numeric/datetime column separately.

IMPORTANT: this feature performs all calculations on the client side, so do not use this function on huge datasets with millions of records because IBExpert will fetch all records from the server before calculating.

Since IBExpert version 2004.8.26.1 it is possible to display data as Unicode. Simply click the relevant icon in the Navigation toolbar or use [F3]. It is not possible to edit the data directly in the grid. To edit data in Unicode, use the *Form View* or modal editor connected with string cell.

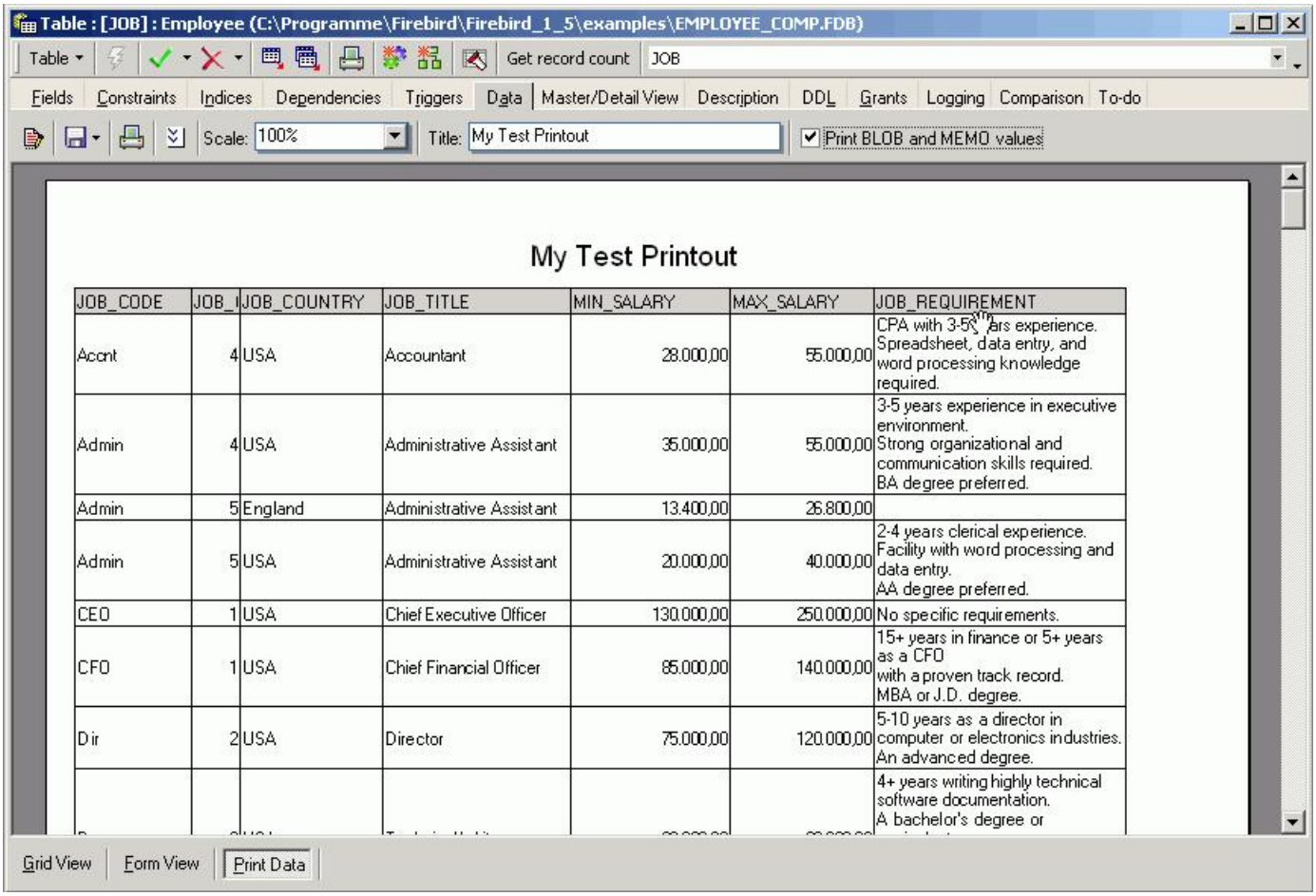
2. Form View - one data set is displayed at a time in a form.



New to version 2004.8.26.1: The *Form View* has been completely redesigned. It now also displays field descriptions. It is also possible to select alternative layouts (*classic* or *compact*), the *compact* alternative for those who prefer a more condensed and faster interface. Visual options now also include specification of *Memo Height* and *Memo Word Wrap*.

3. Print Data - displays data in WYSIWYG mode (the status bar showing which page number is currently visible and how many pages the data covers altogether). The data can be either saved to file or printed.

The *Print Data* view also has its own right-click menu, enabling size adjustments (2 pages, whole page, page width, and scaling from 10% to 200%), this being also available as a pull-down list of options in the *Print Preview* toolbar. Further toolbar options include saving the information to file, printing directly, and specifying the page set up. There is even a check option to specify whether BLOB and MEMO values should be printed or not.



JOB_CODE	JOB_I	JOB_COUNTRY	JOB_TITLE	MIN_SALARY	MAX_SALARY	JOB_REQUIREMENT
Acont	4	USA	Accountant	28.000,00	55.000,00	CPA with 3-5 years experience. Spreadsheet, data entry, and word processing knowledge required.
Admin	4	USA	Administrative Assistant	35.000,00	55.000,00	3-5 years experience in executive environment. Strong organizational and communication skills required. BA degree preferred.
Admin	5	England	Administrative Assistant	13.400,00	26.800,00	
Admin	5	USA	Administrative Assistant	20.000,00	40.000,00	2-4 years clerical experience. Facility with word processing and data entry. AA degree preferred.
CEO	1	USA	Chief Executive Officer	130.000,00	250.000,00	No specific requirements.
CFO	1	USA	Chief Financial Officer	85.000,00	140.000,00	15+ years in finance or 5+ years as a CFO with a proven track record. MBA or J.D. degree.
Dir	2	USA	Director	75.000,00	120.000,00	5-10 years as a director in computer or electronics industries. An advanced degree.
						4+ years writing highly technical software documentation. A bachelor's degree or

IBExpert also offers a Test Data Generator (IBExpert Tools menu), should test data be required for comparing query times etc.

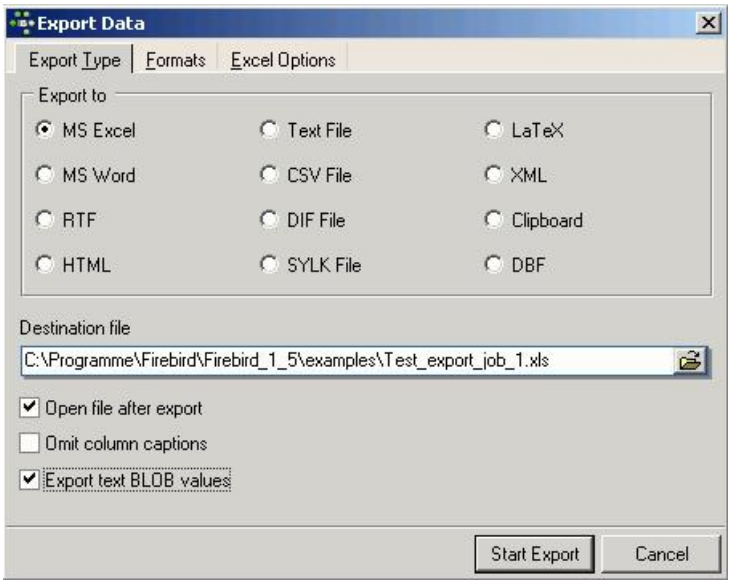
Note that when deleting data, the InterBase/Firebird database becomes larger, as the data is merely flagged as deleted, due to the rollback option, which is available until the drop commands are committed.

Export Data

Data can be exported from the Data page in the Table Editor and from the Results page in the SQL Editor, by simply clicking the

icon or using the key combination [Ctrl + E] to open the *Data Export* window.

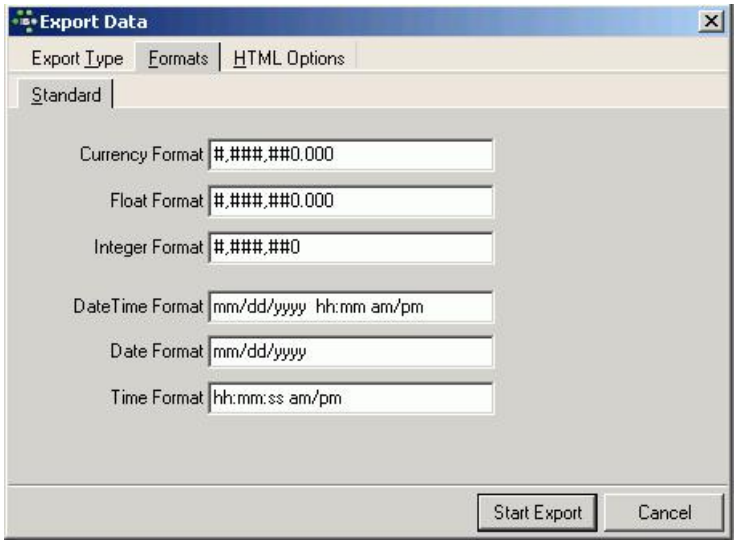
The first page in the *Export Data* dialog, *Export Type*, offers a wide range of formats, including Excel, MS Word, RTF, HTML, Text, CSV, DIF, SYLK, LaTex, SML, Clipboard and DBF, which can be simply and quickly specified per mouse click (or using the directional keys).



The destination file name must also be specified, and check options allow you to specify whether the resulting export file should be opened following the data export or not, and - for certain export formats - whether column headings should be omitted or not, and whether text BLOB values should also be exported.

Should you encounter problems when exporting text BLOB values, please check that the *Show text blobs as memo* option is checked on the *Grid* page found unter the IBExpert menu item Options / Environment Options.

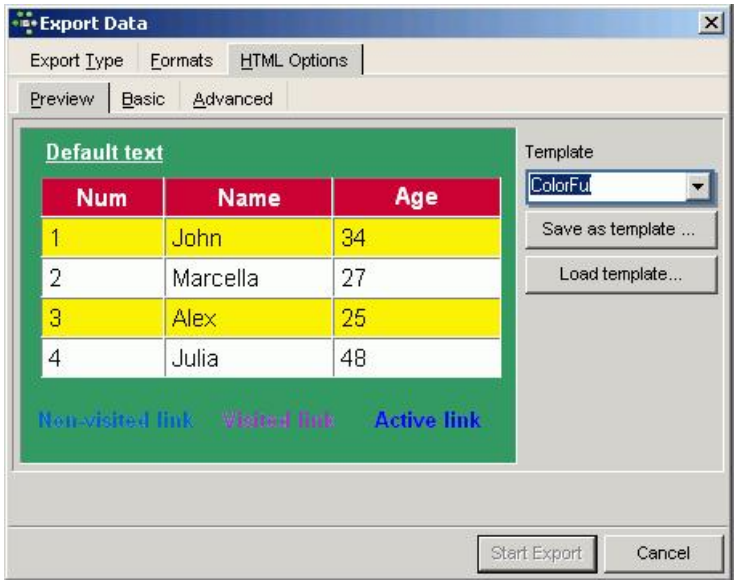
Depending on the format, further options can be specified on the second or third pages, *Formats* and *Options*, specific to the export type. The *Formats* page is available for all export types, with the exception of XML.



Here it is possible to specify a range of numerical formats, including `currency`, `float`, `integer`, `date`, `time` or `date and time`. Please note that not all of these options may be altered for all export types (for example when exporting to DBF it is only possible to specify the formats for date/time and time).

Depending upon which format has been specified, additional options may be offered on the third page, for example:

- Excel - specification of page header and footer.
- HTML - template selection and preview, title, header and footer text as well as a wide range of advanced options.
- CSV - Quote String check option, and user specification of CSV separator.
- XML - Encoding format may be selected from a pull-down list. There are also check options to export `String`, `Memo` and `DateTime` fields as text.
- DBF - check options to export strings to DOS, long strings to Memo, and to extract `DateTime` as `Date`.



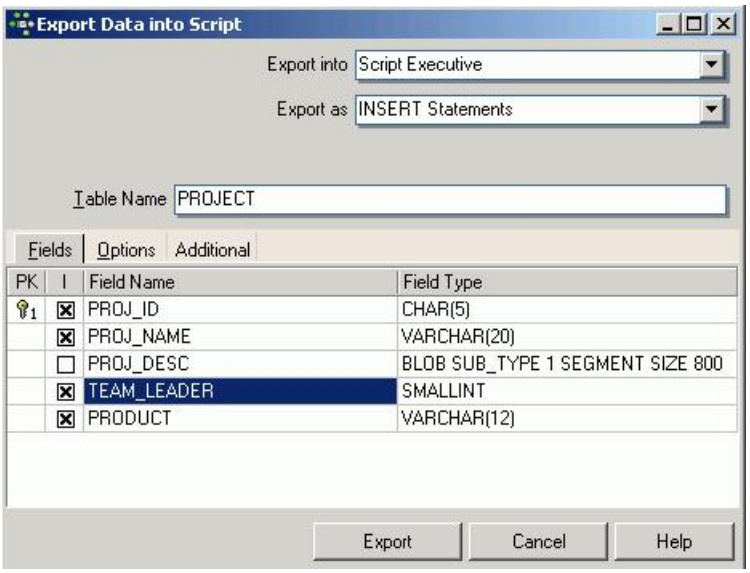
The export is then finally started using the *Start Export* button in the bottom right-hand corner. Following a successful export, a message appears informing of the total number of records exported.

Using the right-hand icon in the or Table Editor toolbar (*Export data into script*) the data can be exported into an insert SQL script (without the blob fields).

Export Data into Script

The *Export Data into Script* dialog can be started using the

on the Data page in the Table Editor or the Results page in the SQL Editor.

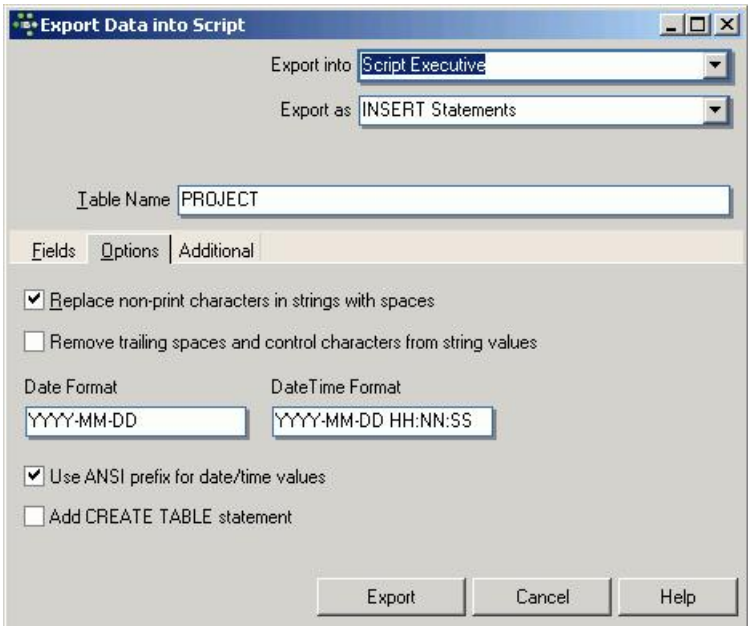


The following options may be selected before starting the export:

- Export into: *File*, *Clipboard* or *Script Executive*.
- Export as: INSERT statements, UPDATE statements or since version 2003.12.18.1 there is also the added possibility to export data as a set of EXECUTE PROCEDURE statements.

Specify the file name if exporting to file and the table name from which the data is to be exported. The Fields page allows the table fields to be selected.

The *Options* page:



offers a number of options including replacement of non-print characters in strings with spaces, removal of trailing spaces and control characters from string values, date and time specification and whether the CREATE TABLE statement should be added into the script.

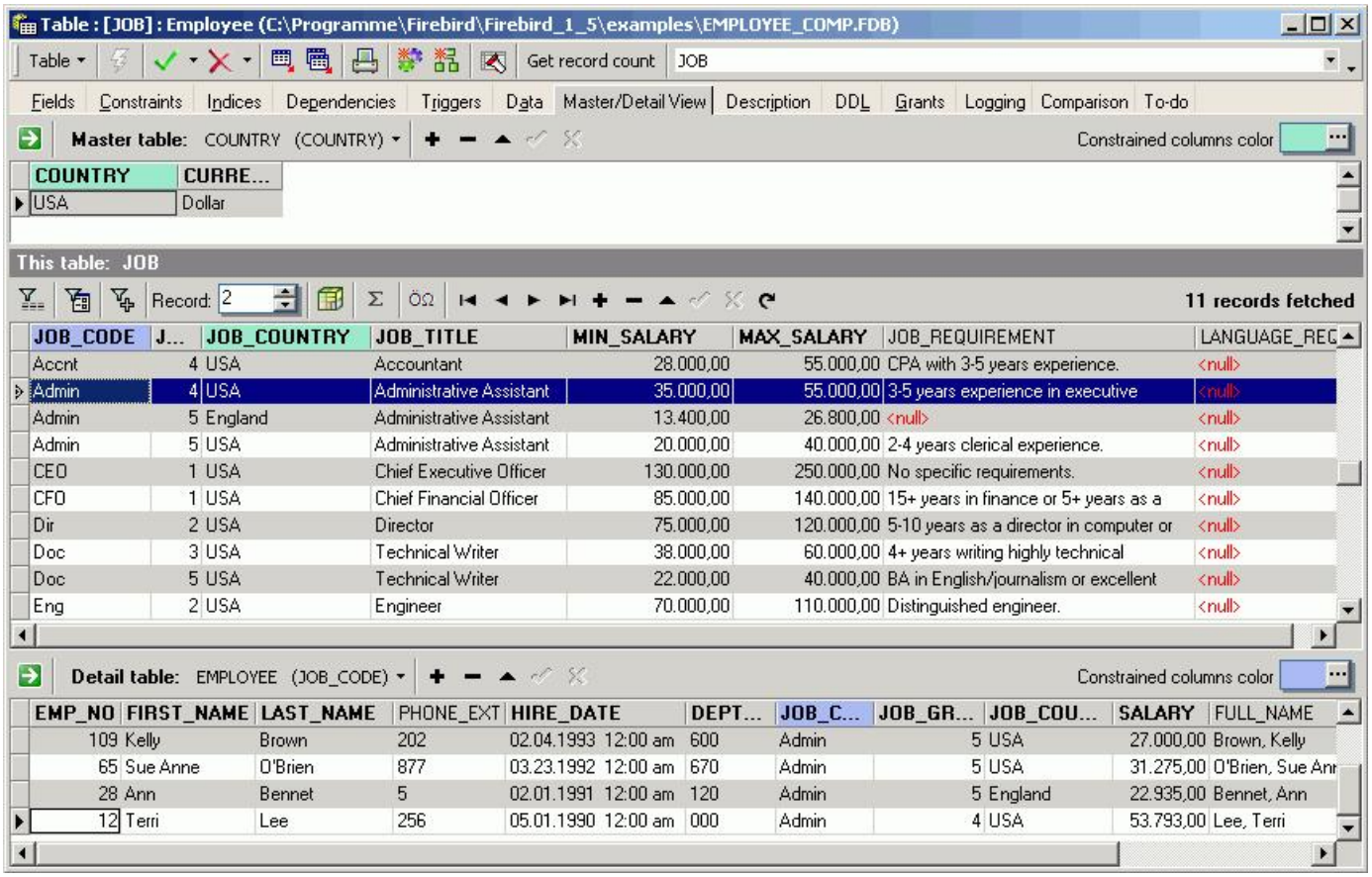
The *Additional* page allows additional definitions for query to be made, for example, ORDER BY or WHERE clauses.

After completing all specifications as wished, simply click the Export button to perform the data export.

Please note that since IBExpert version 2007.09.25 IBExpert can work with scripts larger than 2 GB. With older IBExpert versions, should the script exceed 2 GB, you will need to split it into two or more smaller ones. This can be done using the IBExpert Tools menu item Extract Metadata, where it is possible to specify the option *separate files* and even the maximum file size limit.

(7) Master/Detail View

The *Master/Detail View* was added to the object editors in IBExpert version 2006.06.05. This allows you to view data of tables that reference (or are referenced by) the current table by a foreign key.



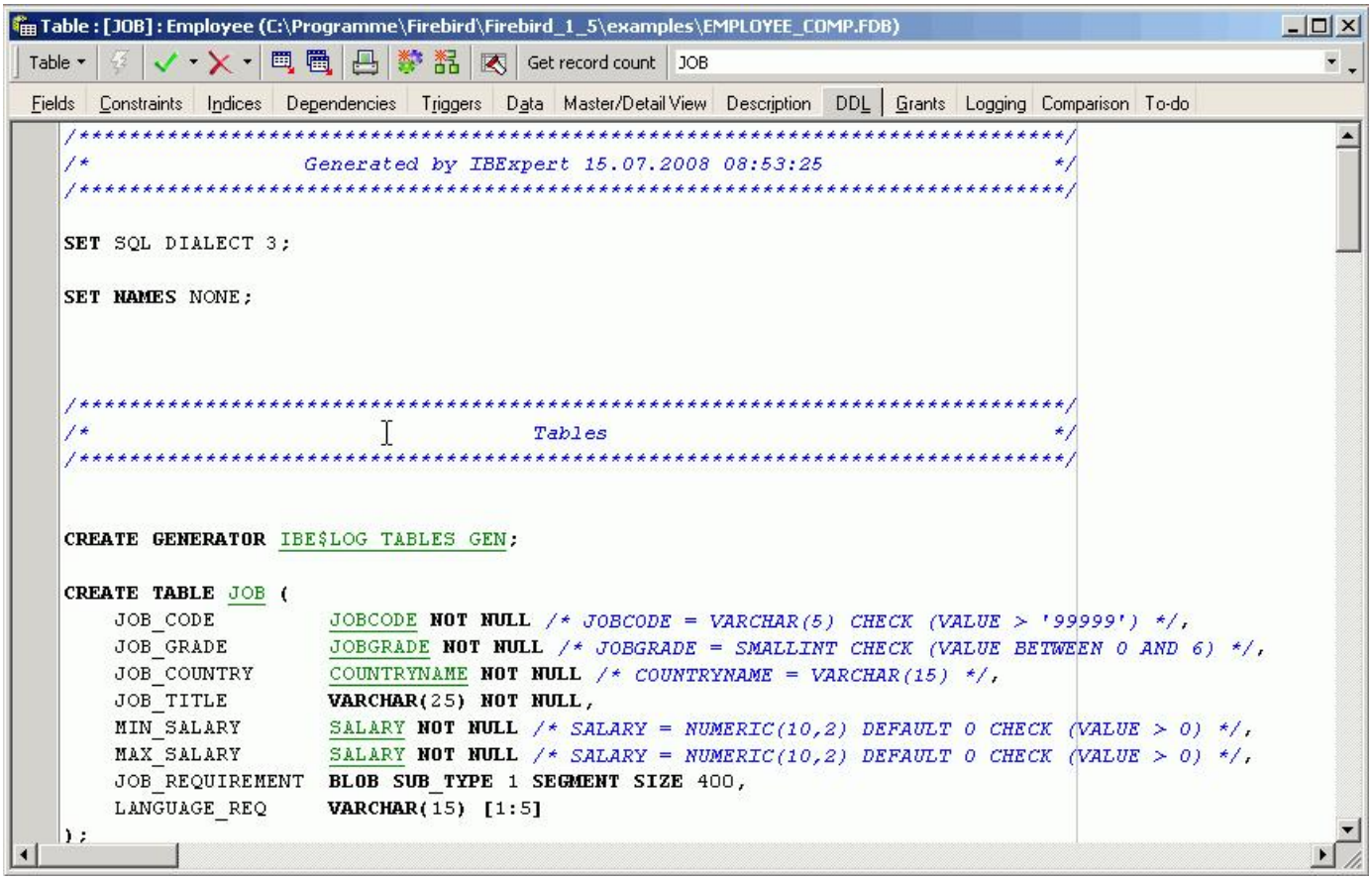
Since IBExpert version 2006.08.12 it is possible to edit Master/Detail data. Use the *Commit* and *Rollback* toolbar buttons to commit or rollback any changes.

(8) Description

As with the majority of the IBExpert Editors, the Table Editor's Description page can be used to insert, edit and delete text by the user as wished. It enables the database to be simply and quickly documented.

(9) DDL

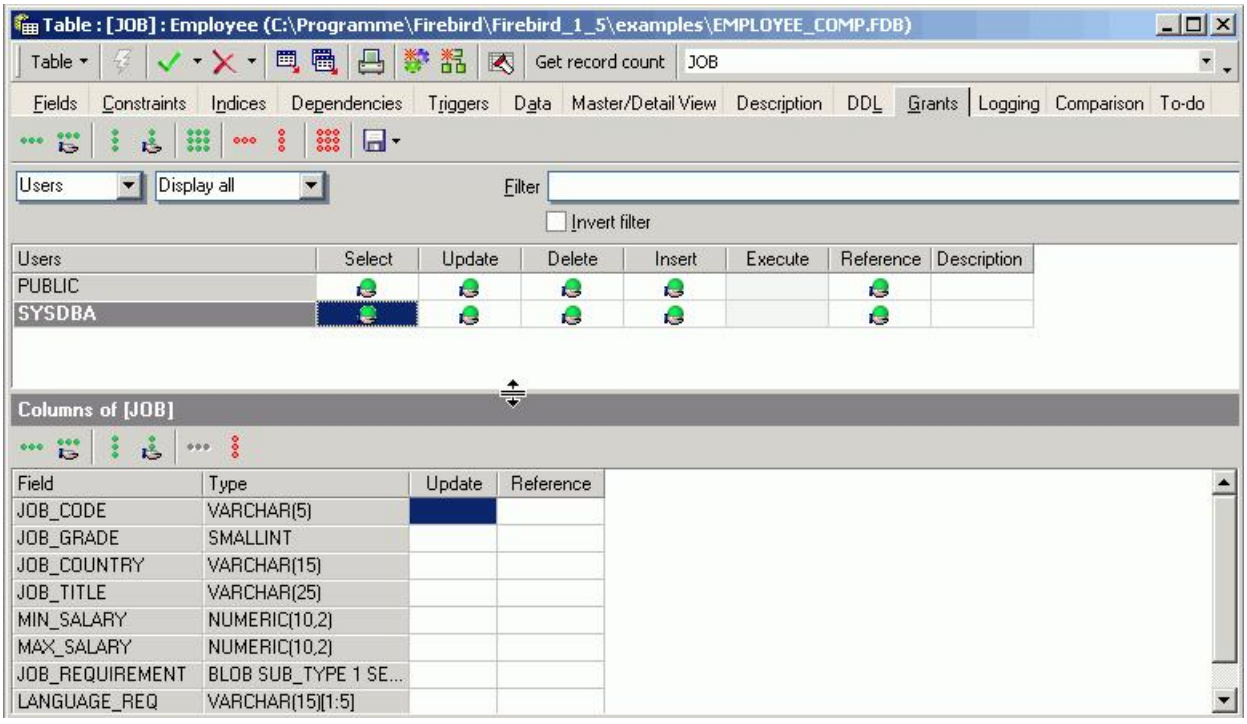
This displays the database table definition as SQL script.



This DDL text cannot be edited here, but it can be copied to the clipboard.

(10) Grants

Here individual users can be assigned rights to SELECT, UPDATE, DELETE and INSERT for the current table. In some cases rights can also be assigned to individual fields.

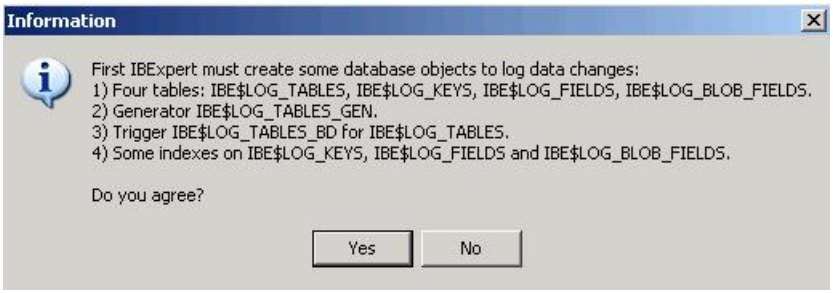


Using the pull-down list, grants can also be assigned for not just users and roles, but also for views, triggers and procedures in the same database, without having to leave the Table Editor.

For more details regarding this subject, please refer to Grant Manager.

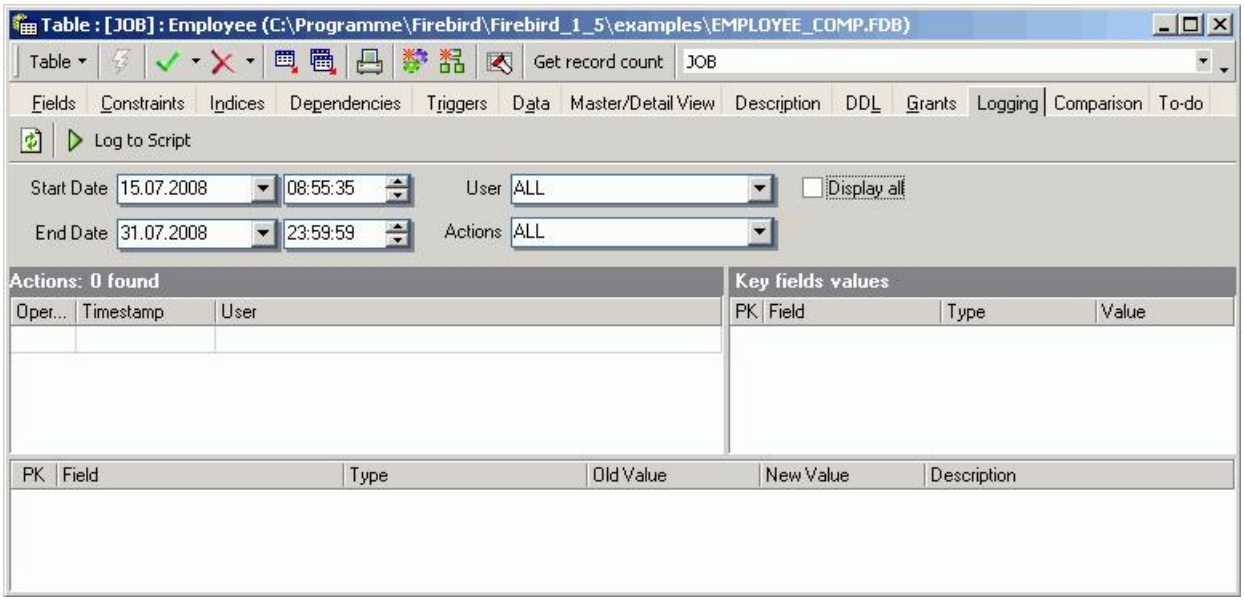
(11) Logging

Data manipulation can be documented here in system tables generated by IBExpert. When this page is opened for the first time, IBExpert asks whether it should generate certain system tables:



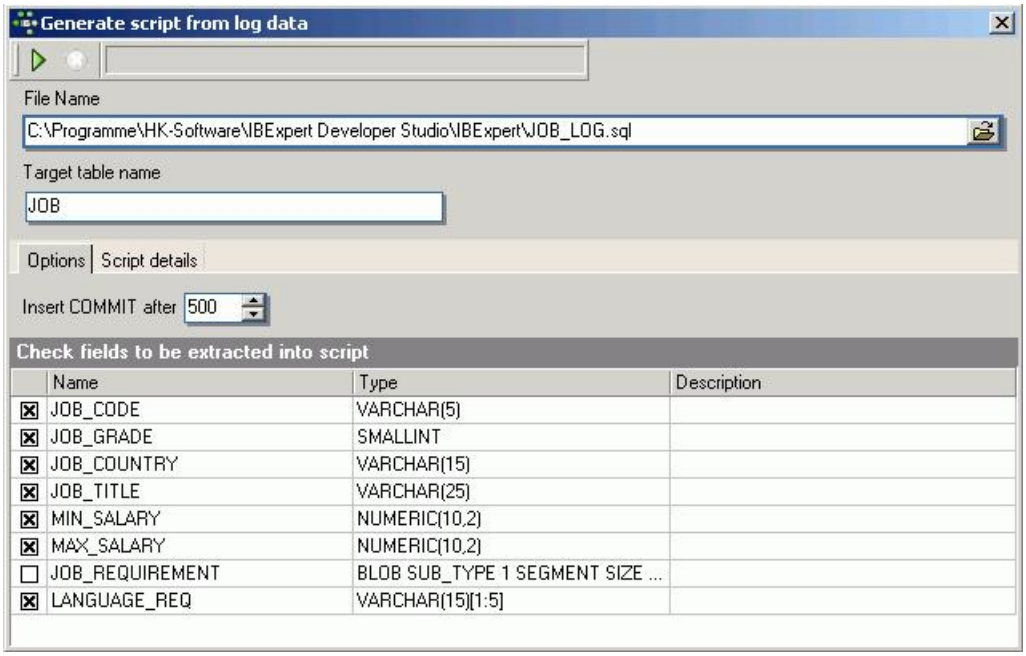
After confirming and committing, you will need to prepare all tables for logging using the respective menu item found in the Log Manager, which is located in the IBExpert Tools menu. Once the preparation has been successfully committed, you can specify whether you wish to log insert, update and/or delete actions.

After generating the script (using the green arrow icon or [F9]), triggers are created for the table, and from now on, regardless of which programm or user makes any changes, all specified alterations are now logged.



Log to script by clicking the respective button:

The log file name, how often should be committed and which fields should be logged can be stipulated on the *Options* page. And the beginning and end of script may be specified under *Script Details* if wished. The script can then simply be generated using the respective icon or [F9].



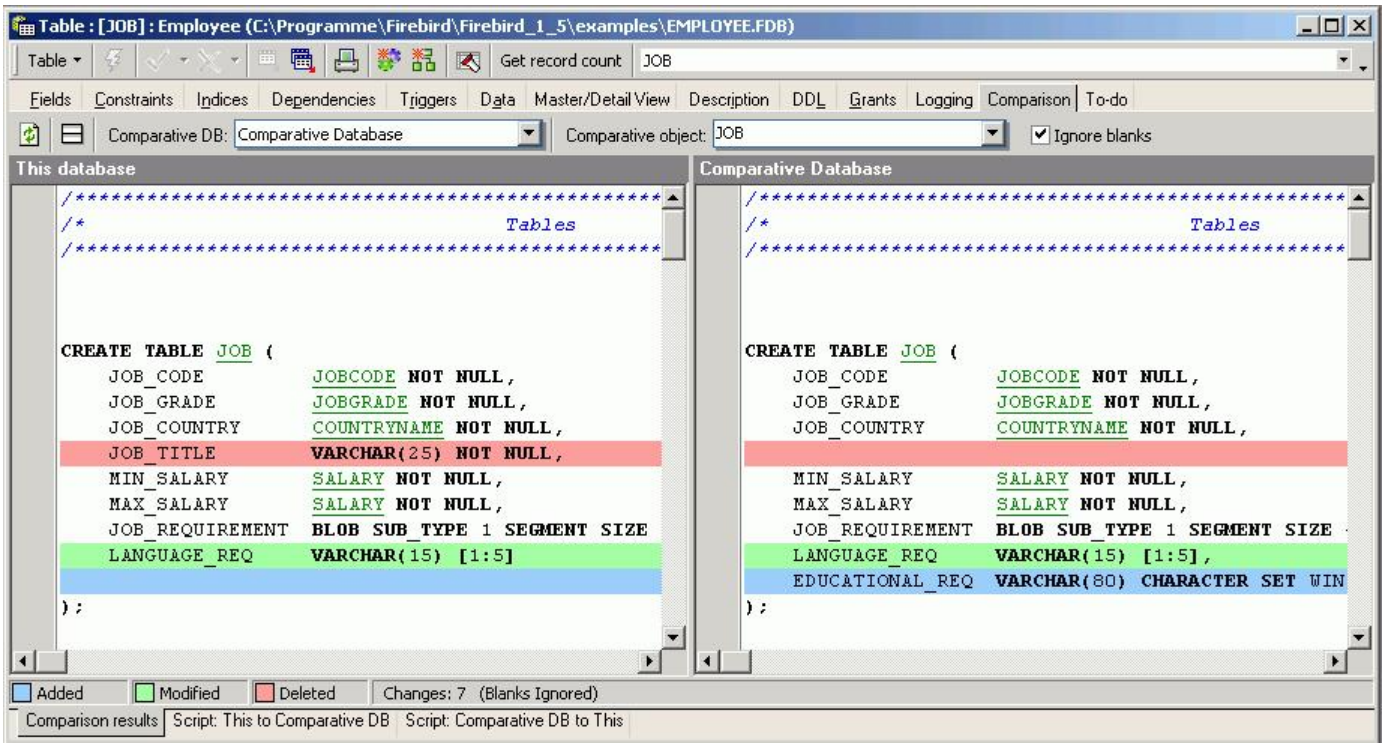
In order to integrate the prepared database object and individual fields into the *Logging* file, you will need to use the IBExpert Tools menu item Log Manager.

(12) Comparison

This new feature was introduced in IBExpert version 2006.03.06.

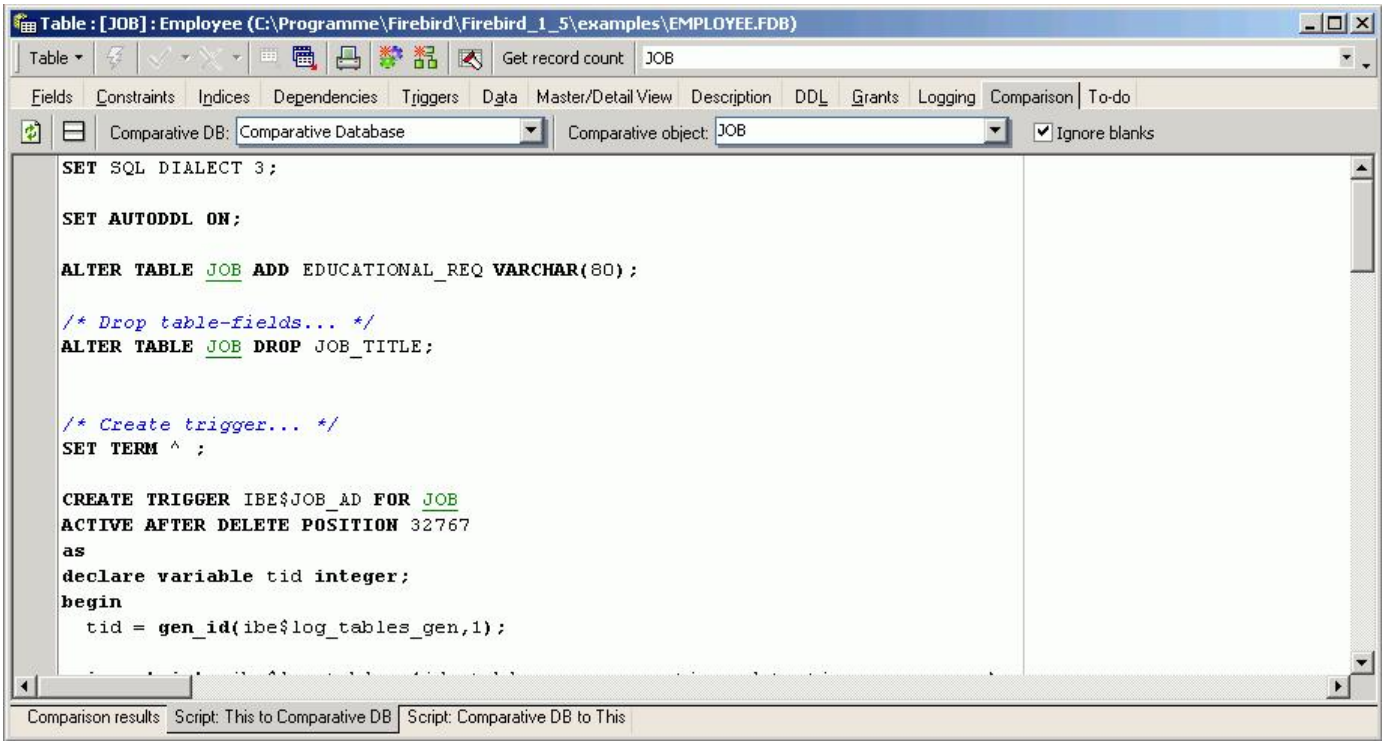
The *Comparison* page allows you to compare a selected database object with one in another (comparative) database. The comparative database must first be specified in the IBExpert Database Registration Info (Comparative Database).

To perform a comparison simply open the object to be compared, click the *Comparison* tab and specify the comparative database:



Uncheck the *Ignore Blanks* checkbox if desired and then click the top left icon (*Compare Again*) to perform the object comparison. The status bar displays the color key, so that the type of alterations made are immediately apparent, as well as the number of changes made.

Below the status bar, there are a further two pages: *Script: This to comparative DB and Script:* and *Script: Comparative DB to This*. Both scripts are supplemented with comments, so that it is quick and simple to detect which alterations need to be made where, in order to update the object either in the main or the comparative database.



(13) To-Do

This feature was introduced in IBExpert version 2007.12.01 and can be used to organize your database development. You can add ToDo items for each object in the database.

Create View from Table (Updatable View)

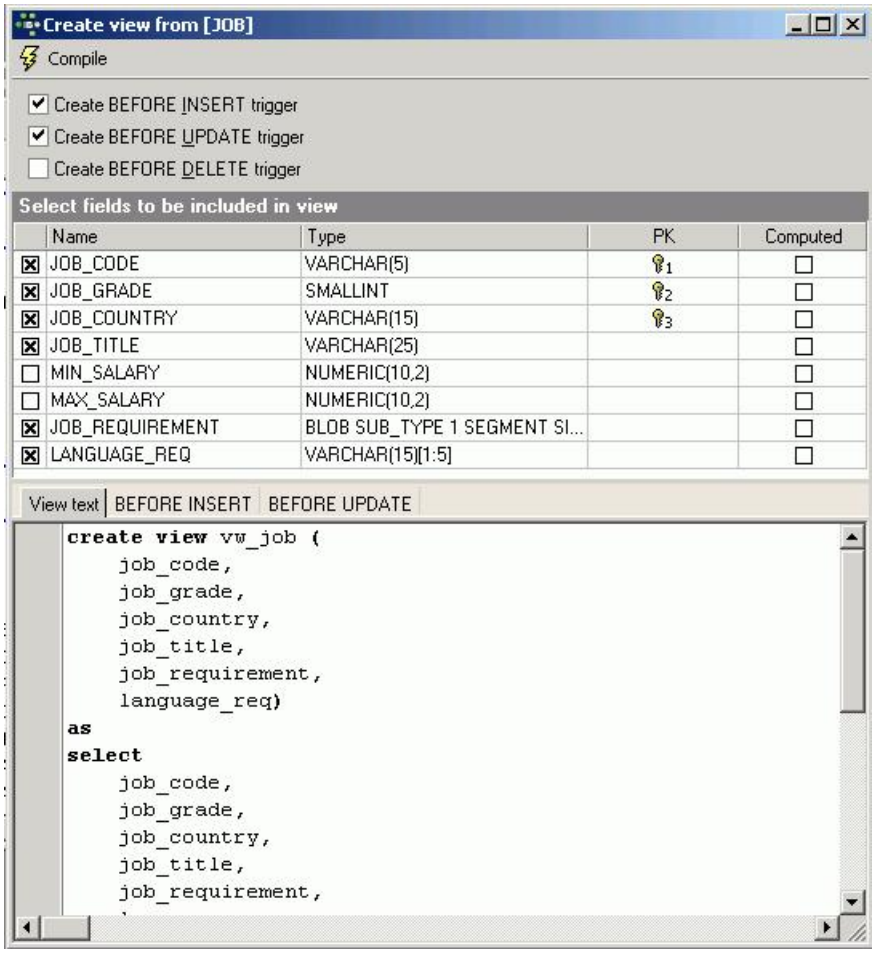
It is possible to create a view directly from a table, using the Table Editor's *Create View* icon:

Select the trigger type simply by activating/deactivating the relevant trigger type checkbox (`BEFORE INSERT`, `BEFORE UPDATE`, `BEFORE DELETE`).

The list of fields to be included in the view may be specified by simply clicking on the check boxes to the left of the field names, or by double-clicking or using the space bar on a selected field.

The view code is displayed in the lower window and may also be amended as wished.

As with the view default name, the trigger default name is automatically generated by IBExpert, comprising the prefix `VW_` followed by the table name and ending with the trigger type suffix (`_BI` = Before Insert, `_BU` = Before Update, `_BD` = Before Delete). This can of course be overwritten if wished.



One or more trigger types may be specified - whereby further tabs appear in the lower area, allowing the pre-defined trigger code to be simply amended as wished, automatically creating an updatable view - this is, in fact, an extremely quick and simple way to create a view that is updatable, and which can otherwise only be realized with considerable manual labor! These triggers are already prepared, and require little work in order to create an updatable view.

Finally compile and commit to create the new view or updatable view.

Create Procedure from Table

A procedure can be created directly from a table, using the Table Editor's *Create Procedure* icon:

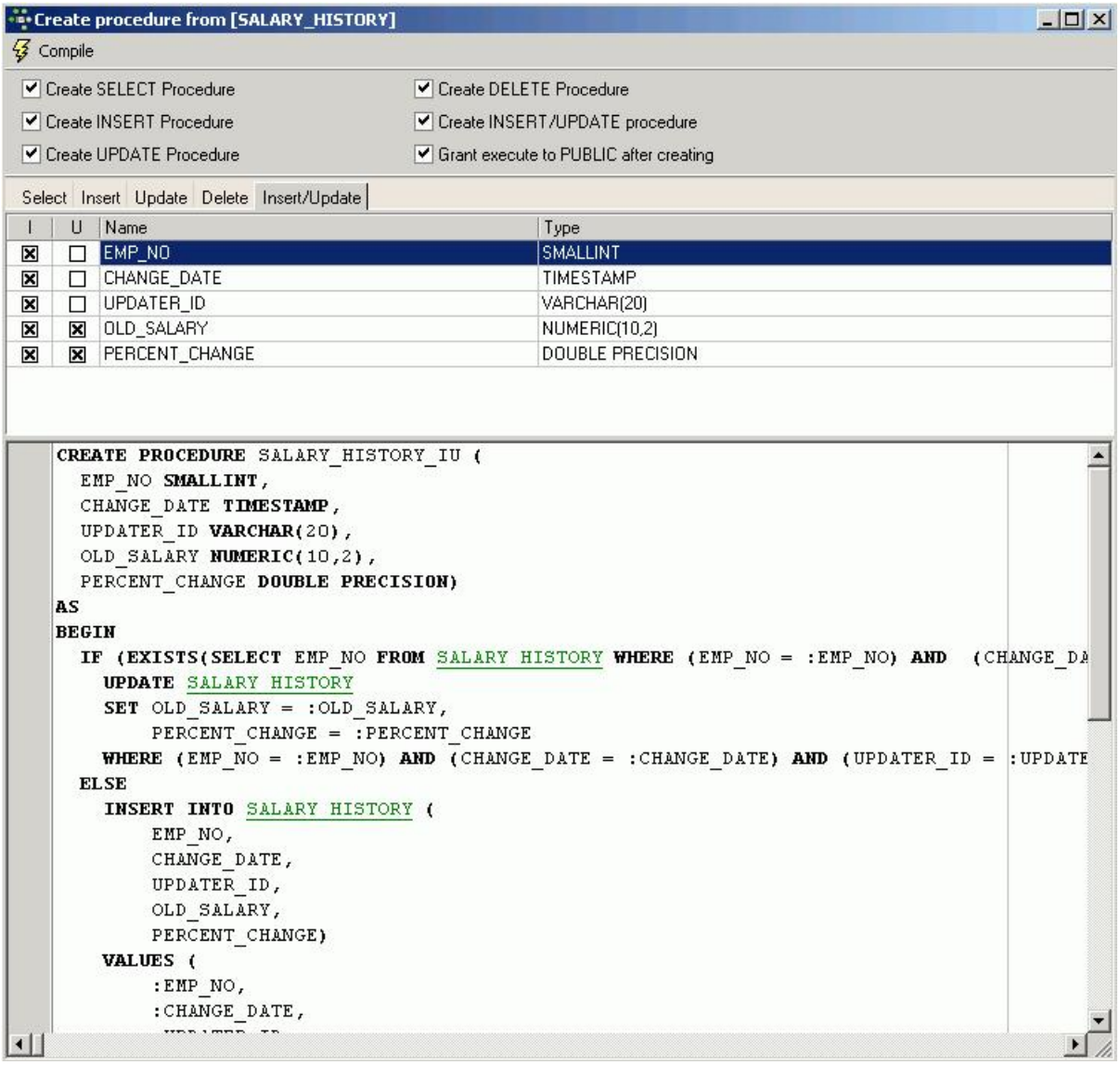
The sort of procedure to be created can be specified by checking/unchecking the boxes in the upper area.

Options include:

- SELECT
- INSERT
- UPDATE
- DELETE
- INSERT/UPDATE

with a further checkbox option to:

- *Grant execute to PUBLIC* after creating.



A procedure default name is automatically generated by IBExpert, comprising the table name followed by one of the following suffixes:

- S = SELECT
- I = INSERT
- U = UPDATE
- D = DELETE
- IU = INSERT/UPDATE

This name can of course be overwritten or altered directly in the code if wished.

The list of fields to be included in the procedure may be specified as wished by simply clicking on the check boxes to the left of the field names, or by double-clicking or using the space bar on a selected field.

The procedure text is displayed in the lower window and may also be altered if wished. Switch from one page to the next by clicking on the tabs (displayed above the fields lists).

Finally compile and commit to create the new procedure.

Print Table

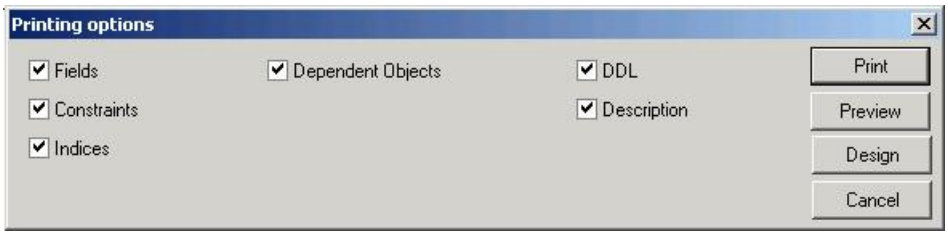
Please refer to the IBExpert Edit Menu item Print and the Table Editor Menu item Printing Options.

Print Preview and Print Design

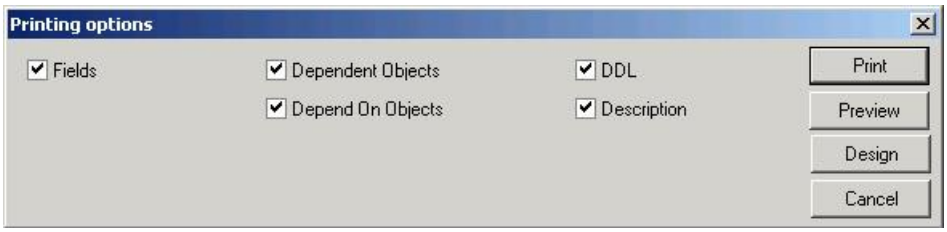
Please refer to the IBExpert Report Manager for further information.

Printing Options

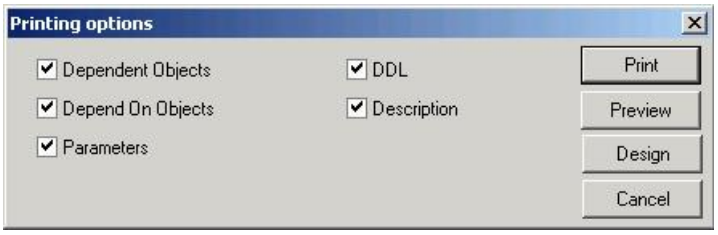
The *Printing Options* dialog can be started using the *Print Table Metadata* icon or [Shift + Ctrl + P]. The *Printing Options* dialog offers different options depending upon which Editor it is started from. For example, when started from the Table Editor:



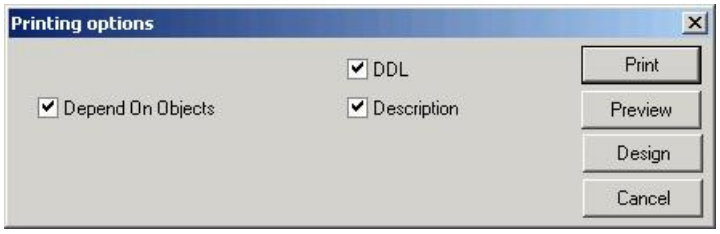
the View Editor:



the Procedure Editor:



the Trigger Editor:



These options include the following:

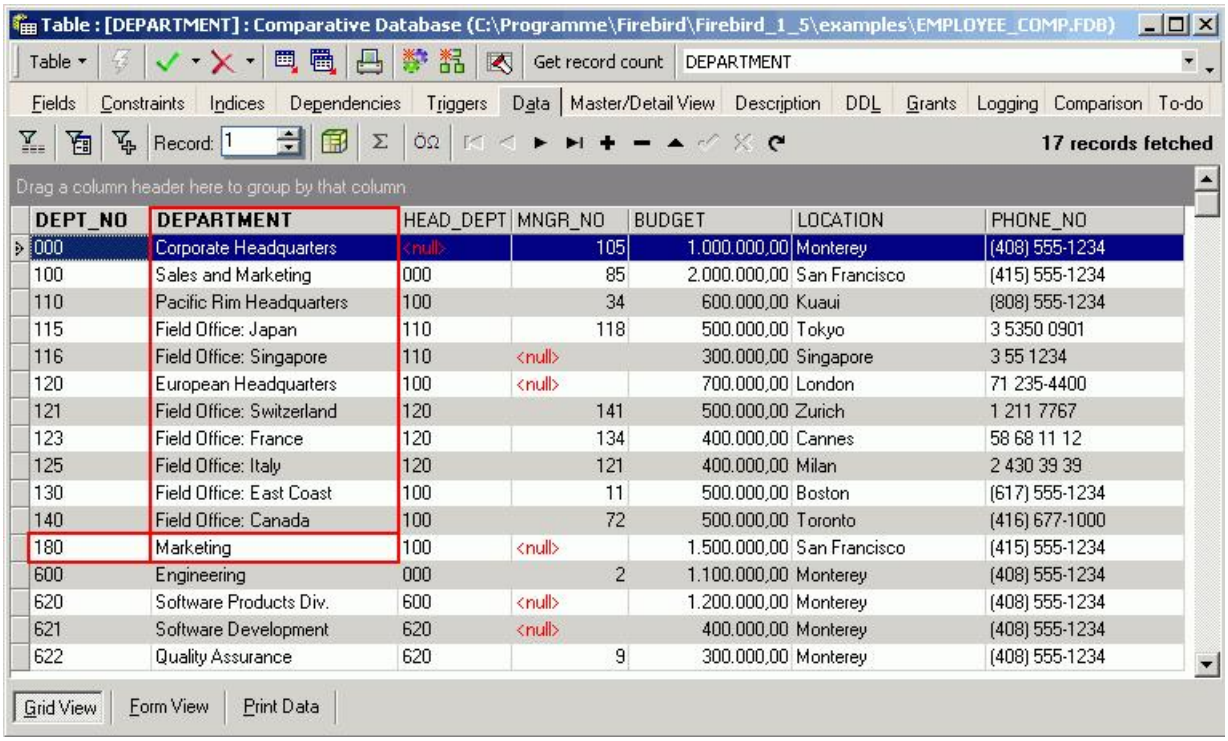
- Fields
- Constraints
- Indices
- Dependent Objects
- Depend On Objects
- Parameters
- DDL
- Description

Simply check as wished, and then click *Preview* (to view the report as it will be printed - see Print Preview for further information), *Design* (to customize the report - refer to Report Manager for further information) or *Print* to proceed to the standard *Windows Print* dialog.

See also:
Grant Manager
Log Manager
Database Registration Info
DCL-DataControlLanguage
DDL-DataDefinitionLanguage
DML-DataManipulationLanguage

Field

A field can be defined as the intersection in a table where a row meets a column, containing a clearly differentiated atomic piece of information. Each data field should be unique and represent and indivisible quantity of information.



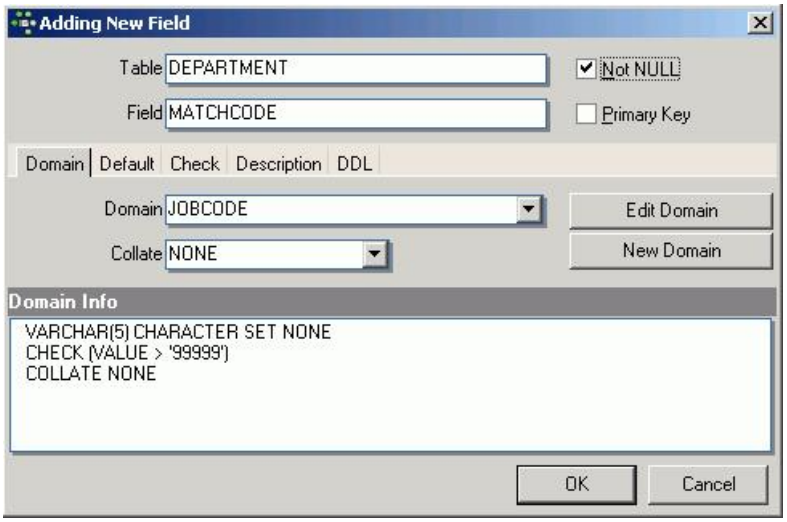
DEPT_NO	DEPARTMENT	HEAD_DEPT	MNGR_NO	BUDGET	LOCATION	PHONE_NO
000	Corporate Headquarters	<null>	105	1.000.000,00	Monterey	(408) 555-1234
100	Sales and Marketing	000	85	2.000.000,00	San Francisco	(415) 555-1234
110	Pacific Rim Headquarters	100	34	600.000,00	Kuaui	(808) 555-1234
115	Field Office: Japan	110	118	500.000,00	Tokyo	3 5350 0901
116	Field Office: Singapore	110	<null>	300.000,00	Singapore	3 55 1234
120	European Headquarters	100	<null>	700.000,00	London	71 235-4400
121	Field Office: Switzerland	120	141	500.000,00	Zurich	1 211 7767
123	Field Office: France	120	134	400.000,00	Cannes	58 68 11 12
125	Field Office: Italy	120	121	400.000,00	Milan	2 430 39 39
130	Field Office: East Coast	100	11	500.000,00	Boston	(617) 555-1234
140	Field Office: Canada	100	72	500.000,00	Toronto	(416) 677-1000
180	Marketing	100	<null>	1.500.000,00	San Francisco	(415) 555-1234
600	Engineering	000	2	1.100.000,00	Monterey	(408) 555-1234
620	Software Products Div.	600	<null>	1.200.000,00	Monterey	(408) 555-1234
621	Software Development	620	<null>	400.000,00	Monterey	(408) 555-1234
622	Quality Assurance	620	9	300.000,00	Monterey	(408) 555-1234

Each database field has a name, which enables the data to be accessed. A database field can be based on a domain definition or defined individually in the IBExpert Create Table or Table Editors, in which case InterBase/Firebird automatically creates a system domain for the field definition.

Adding new field (insert field) using the Field Editor

Fields can be inserted into a table at the time of table creation, using the IBExpert DB Explorer or menu item *New Table*. It is however often necessary to add new fields, after the table has been created. This can be easily done in IBExpert by opening the Table Editor (double-click on the relevant table in the IBExpert DB Explorer) or using the DB Explorer right-click menu *Edit Table ...* (or key combination [Ctrl + O]), and then inserting a field using the

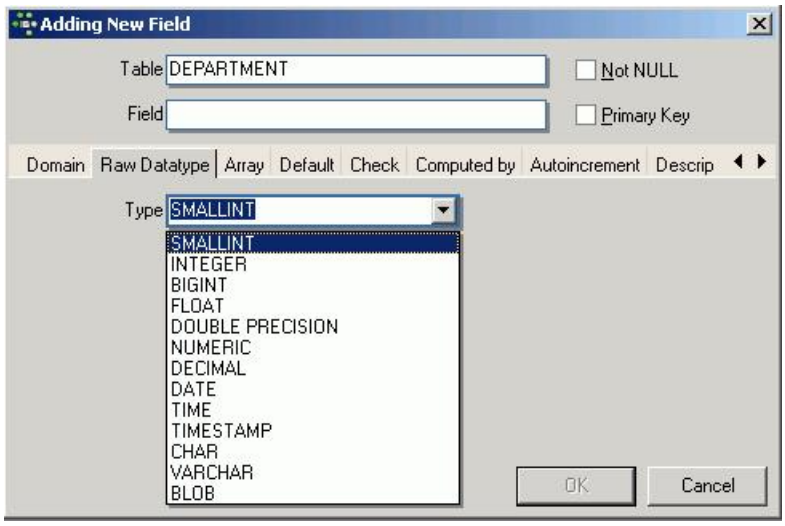
Add Field icon (or [Ins] key) or the Table Editor right-click menu *Insert Field*, to open the *Adding New Field* Editor.



The *Adding NewField* Editor displays the table name, into which the field is to be inserted. The new field name can be specified by the user, along with the parameters NOT NULL and Primary Key. Further options are to be found on the *Default* and *Check* pages, and the usual IBExpert *Desc* (= Description) and *DDL* (= [[DDL-Data Definition Language | Data Definition Language) information pages are also included.

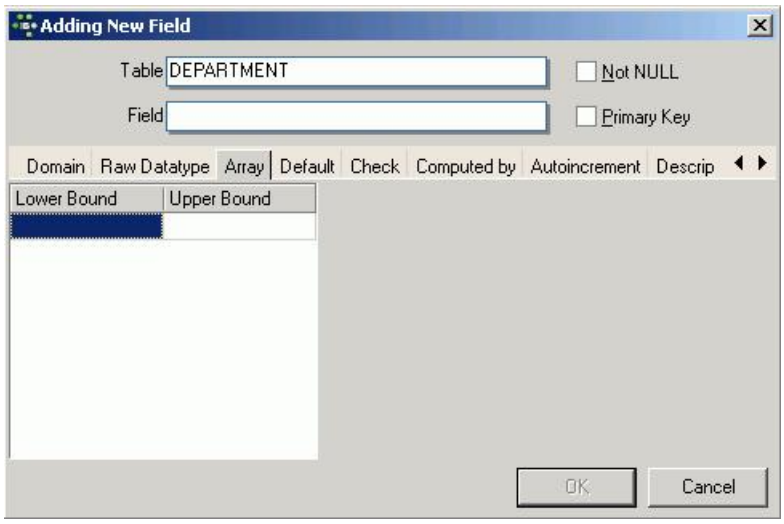
The new field may be based upon an existing domain (which may be edited using the *Edit* button) or a New Domain can be created directly from the *NewField* Editor. All existing domains (in the connected database) can be viewed in the "Domain" pull-down list. The domain information can be viewed in the Editor's lower panel.

It is also possible to define certain numeric formats as standard using the Options menu, Environment Options / Grid / Display Formats, if wished. These format standards can be overwritten in individual fields here in the Field Editor.

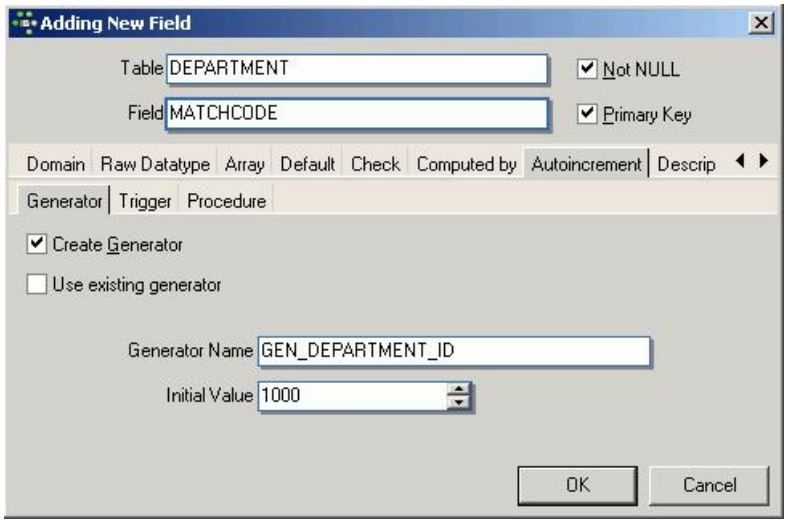


Of course a new field doesn't have to be based on a domain. The datatype can be specified using the pull-down list under the *RawDatatype* tab. However, InterBase/Firebird automatically generates a system domain for all specified fields, so when a new field is inserted, or existing field altered, InterBase/Firebird inserts or alters the respective system domain.

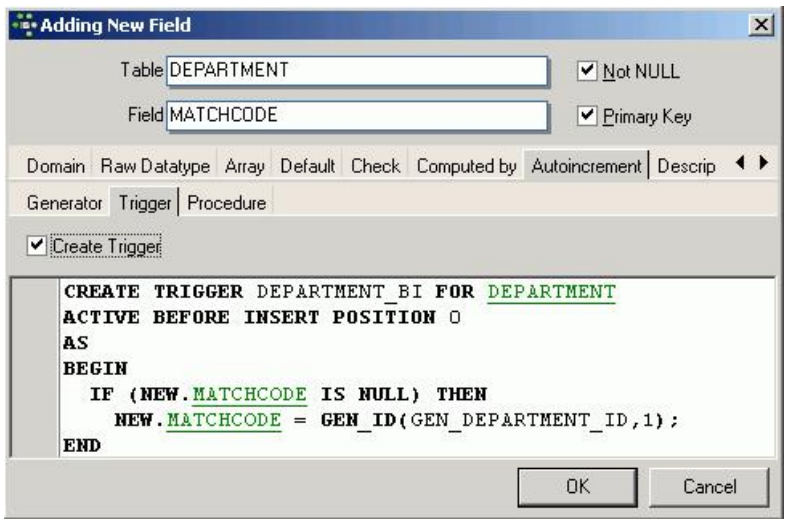
Additional context-sensitive input fields appear, relevant to the datatype selected (e.g. when VARCHAR is selected, options for specifying *Length*, *Charset*, and *Collate* are offered; in the case of NUMERIC, *Precision* and *Scale* can be specified).



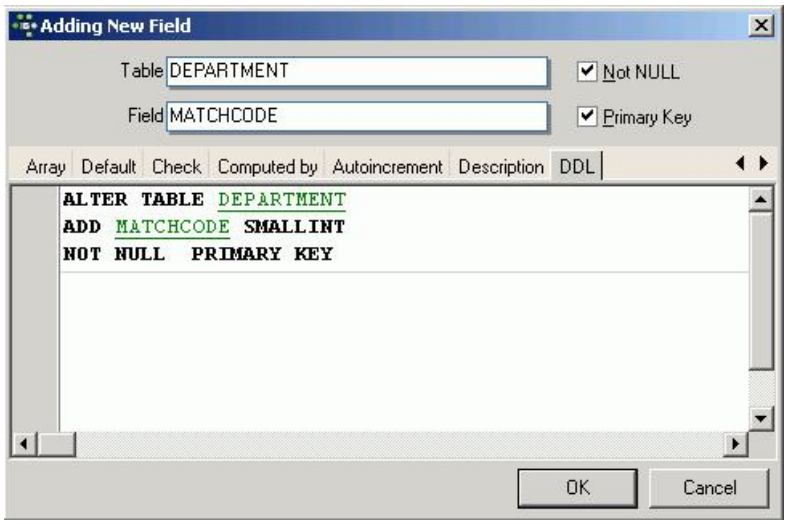
Furthermore arrays can be defined, as well as default values, check constraints, "computed by" calculations and autoincrements.



The autoincrement page allows new generators to be created, or an existing generator to be selected. New triggers and procedures can also be created directly here in this Editor for this field, if desired.



As with the majority of the IBExpert Editors, the last two pages display the object *Description* (which can be inserted, edited and deleted here by the user as wished), and the *DDL* page,

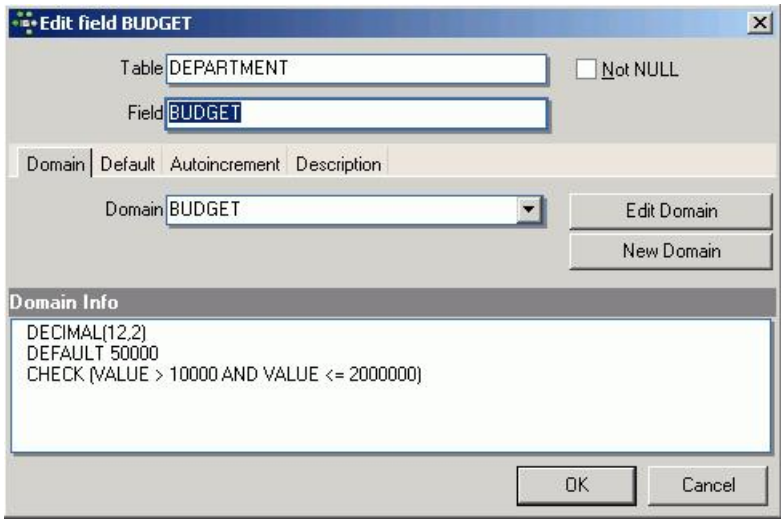


which displays the SQL code for the field as specified by the user.

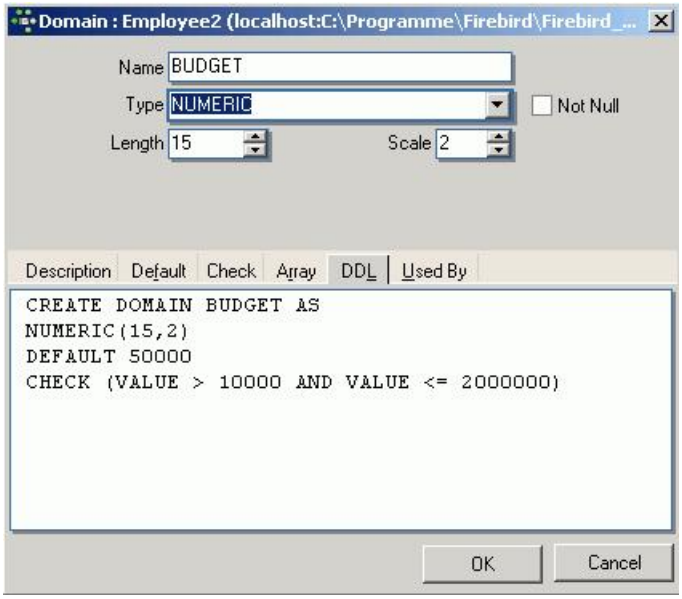
Alter field

Similar to Alter Domain, only certain field attributes may be altered. For example, `CHECK` instructions and default values may be added, altered or deleted. However it is not possible to alter the basic datatype (for example, from `NUMERIC` to `VARCHAR`). Neither is it possible to drop a `NOT NULL` constraint. To alter these the field has to be dropped and recreated (see Drop Field).

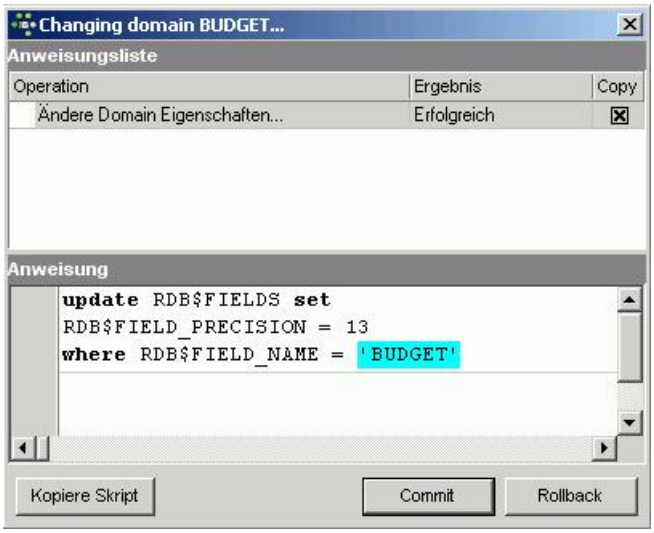
Fields can be altered in the Table Editor by double-clicking on the selected field, or right-clicking and selecting Edit Field from the menu, or pressing the [Enter] key, to open the Field Editor:



However you will notice that you need to switch to the Domain Editor to perform any actual changes, as even if the field is not based on a user-defined domain, InterBase/Firebird automatically creates a system domain for all field definitions. Simply click *Edit Domain* to spring to the Domain Editor:



The desired alterations can however be easily made to the user-defined or system domain and executed and checked before finally committing:

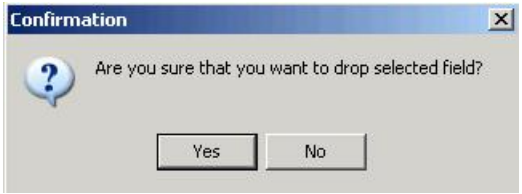


Please refer to Alter Domain and Alter Table for further information.

Drop field/delete field

Fields can be dropped directly in the Table Editor on the Fields page, by using the "-" icon in the Table Editor toolbar, selecting from the right-click menu or using the key combination [Shift + Del].

IBExpert asks for confirmation:



before finally dropping the field. Once dropped, it cannot be retrieved.

When dropping fields, it is important to note that the field may not be part of the table's primary key, have a foreign key relationship with another table, contain a unique constraint, be part of a table constraint or part of another column's CHECK constraint.

The Constraints page in the Table Editor lists all such fields, so that the developer can quickly ascertain whether constraint alterations/deletions are necessary, before dropping the field in question (or whether, in fact, the field should be dropped at all!).

Using SQL the syntax is:

```
ALTER TABLE <table_name>
DROP <field_name>;
```

See also:
Field Definitions

Field Definitions

Charset / Character Set

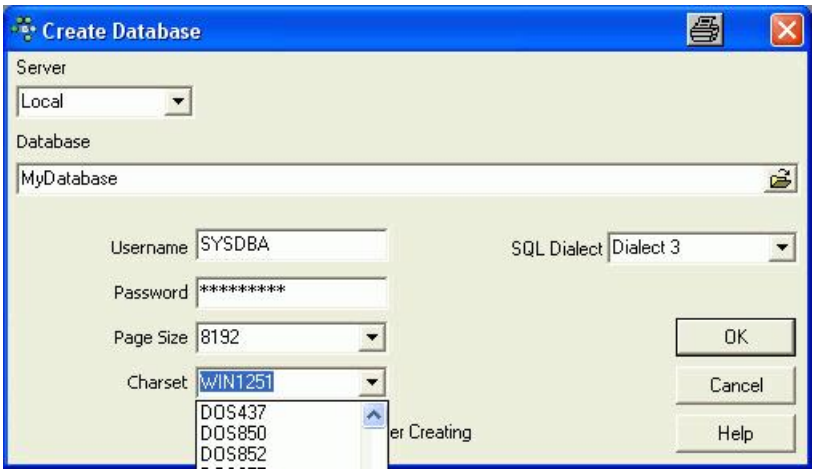
A character set is specified in InterBase/Firebird to define which characters are allowed in a CHAR, VARCHAR or BLOB field. It also provides collation options when InterBase/Firebird needs to sort a column.

Character set definition becomes increasingly important as the world of database programming spreads more and more across national borders. Today it is often necessary for applications to also meet the requirements of other countries. The problem of multilingual interfaces is just one aspect of internationalization. A modern application needs to handle the particularities specific to individual countries such as, for example, sorting order (collation). In the German language the umlauts ä, ö und ü are integrated in the alphabet using the letter combinations ae, oe and ue. At the same time there are also special characters in the French language, which are not used in the German language such â, á and à.

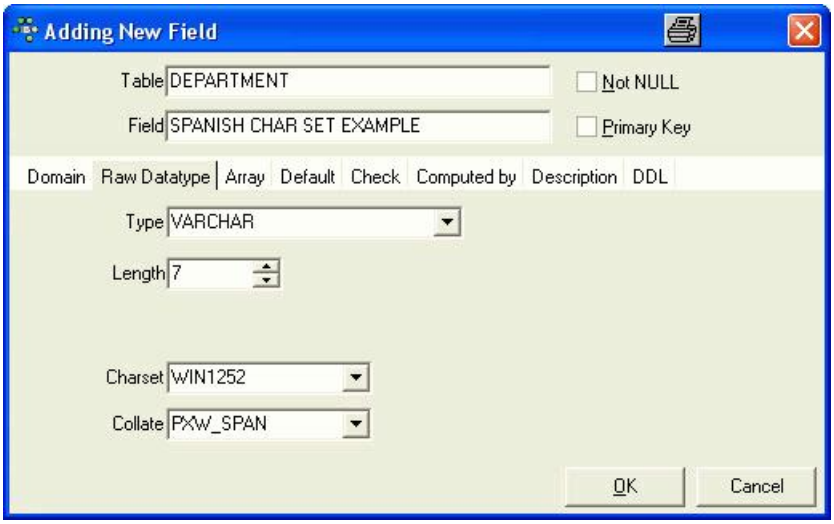
There are completely different problems with versions whose characters are not known in the European character sets, for example Korean or Chinese. These character sets also often contain many more characters, which cannot be incorporated in the 8 bit character sets, as the technical upper limit lies at 256 (=28) different characters. For this reason InterBase/Firebird implements character set support.

Important character sets are, for example, ISO8859_1, to be recommended is Win1252 - the West European character set. Unicode_FSS is the global character set, however there is hardly a program that can read this; Win1251 is the East European character set.

Character sets can be defined for the database (default character set):



or for domains and fields (where the collation can also be specified):



See also:
SET NAMES
Default character set

Overview of the main character sets

By Stefan Heymann

Character sets are an issue every programmer has to deal with one day. This is an overview of the most important character sets.

Name	Bytes per Character	Description	Range	IANA/MIME Code
7-bit ASCII	1	The mother of all character sets. Contains 32 invisible control characters, the Latin letters A-Z, a-z, the Arabic digits 0-9 and a bunch of punctual characters. Code Range 0..127.	0..127	US-ASCII

Unicode-based Character Sets

Unicode, ISO 10646	N.A.	A universal code for all characters anyone can think of. Defines characters, assigns them a scalar value, but does not define how characters are rendered graphically or in memory.	U+0000..U +100000	N.A.
UTF-8	1..6	A Unicode transformation format which uses 1-Byte characters for all 7-bit US-ASCII characters and sequences of up to 6 bytes for all other Unicode characters.	All Unicode characters	UTF-8
UCS-2	2	A unicode transformation format which uses 2 Bytes (16 Bits) for every character. This character set is not able to render all Unicode scalars and is therefore obsolete. However, it is still used by a lot of systems (Java, NT)	U+0000..U +FFFF	ISO-10646-UCS-2
UTF-16	2	A unicode transformation format which uses 2 Bytes (16 Bits) for every character. Using the concept of "Surrogate Pairs", this format is able to render all Unicode characters.	All Unicode characters	UTF-16
UCS-4, UTF-32	4	Two unicode transformation formats which use 4 Bytes (32 Bits) for every character. UCS-4 and UTF-32 are the only character sets, which are able to render all Unicode characters in equally long words. UCS-4 and UTF-32 are technically identical.	All Unicode characters	ISO-10646-UCS-4, UTF-32

Single-byte Character Sets

ISO 8859-x	1	An extension of US-ASCII using the eighth bit.	0..127, 160..255	ISO-8859-x
Windows 125x	1	Equal to ISO 8859-x, plus additional characters in the 128..159 range.	0..255	Windows-125x

ISO 8859-x Character Sets

Name		Covered Languages	MS Windows counterpart
ISO 8859-1	Latin-1		Windows-1252
ISO 8859-2	Latin-2	Central and East European languages (Czech, Polish, etc.)	Windows-1250
ISO 8859-3	Latin-3	South European, Maltese, Esperanto	
ISO 8859-4	Latin-4	North European	
ISO 8859-9	Latin-5	Turkish	Windows-1254
ISO 8859-10	Latin-6	Nordic (Sami, Inuit, Icelandic)	
ISO 8859-13	Latin-7	Baltic	Windows-1257
ISO 8859-14	Latin-8	Celtic	
ISO 8859-15	Latin-9	Similar to ISO 8859-1, adds Euro sign (€) and a few other characters	

MS Windows Character Sets

Number	Name

1250	Latin 2
1251	Cyrillic
1252	Latin 1
1253	Greek
1254	Latin 5
1255	Hebrew
1256	Arabic
1257	Baltic
1258	Viet Nam
874	Thai

Declaring character sets in XML and HTML (IANA charset definitions)

By Stefan Heymann

Declaring character sets in XML

Every XML document or external parsed entity or external DTD must begin with an XML or text declaration like this:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

In the encoding attribute, you must declare the character set you will use for the rest of the document.

You should use the IANA/MIME-Code from Character Set Overview.

Declaring character sets in HTML

In the head of an HTML document you should declare the character set you use for the document:

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
 ...
</head>
```

Without this declaration (and, by the way, without an additional `DOCTYPE` declaration), the W3C Validator will not be able to validate your HTML document.

IANA Character Set Definitions

The Internet Assigned Numbers Authority IANA maintains a list of character sets and codes for them. This list is:

IANA-CHARSETS Official Names for Character Sets, http://www.iana.org/assignments/character-sets

Datatype

InterBase/Firebird tables are defined by the specification of columns, which accommodate appropriate information in each column using datatypes, for example, numerical (`NUMERIC`, `DECIMAL`, `INTEGER`), textual (`CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`), date (`DATE`, `TIME`, `TIMESTAMP`) or blobs.

The datatype is an elemental unit when defining data, which specifies the type of data which may be stored in tables, and which operations may be performed on this data. It can also include permissible calculative operations and maximum data size.

The datatype can be defined in IBExpert using the DB Explorer, by creating a domain or creating a new field in the Create Table or Table Editor.

It can of course, also be defined using SQL directly in the IBExpert SQL Editor. The syntax for the datatype definition is as follows:

```
<data_type> = {
{SMALLINT | INTEGER | BIGINT | FLOAT | DOUBLE PRECISION}
[<array_dim>]
| {DECIMAL | NUMERIC} [(precision [, scale])]
[<array_dim]
| DATE [<array_dim>]
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
 [(int)] [<array_dim>] [CHARACTER SET charname]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
 [VARYING] [(int)] [<array_dim>]
| BLOB [SUB_TYPE {int | subtype_name}) (SEGMENT SIZE int]
 [CHARACTER SET charname]
| BLOB [(seglen [, subtype])]
}
```

The InterBase/Firebird datatype definitions included in this section have been kept as close as possible to original InterBase definitions to avoid any potential misunderstanding or conflict with the datatypes of other database programs.

Blob - Binary Large OBject

A blob is a datatype storing large binary information (Binary Large OBject).

Blobs can contain any binary or ASCII information, for example, large text files, documents for data processing, CAD program files, graphics and images, videos, music files etc.

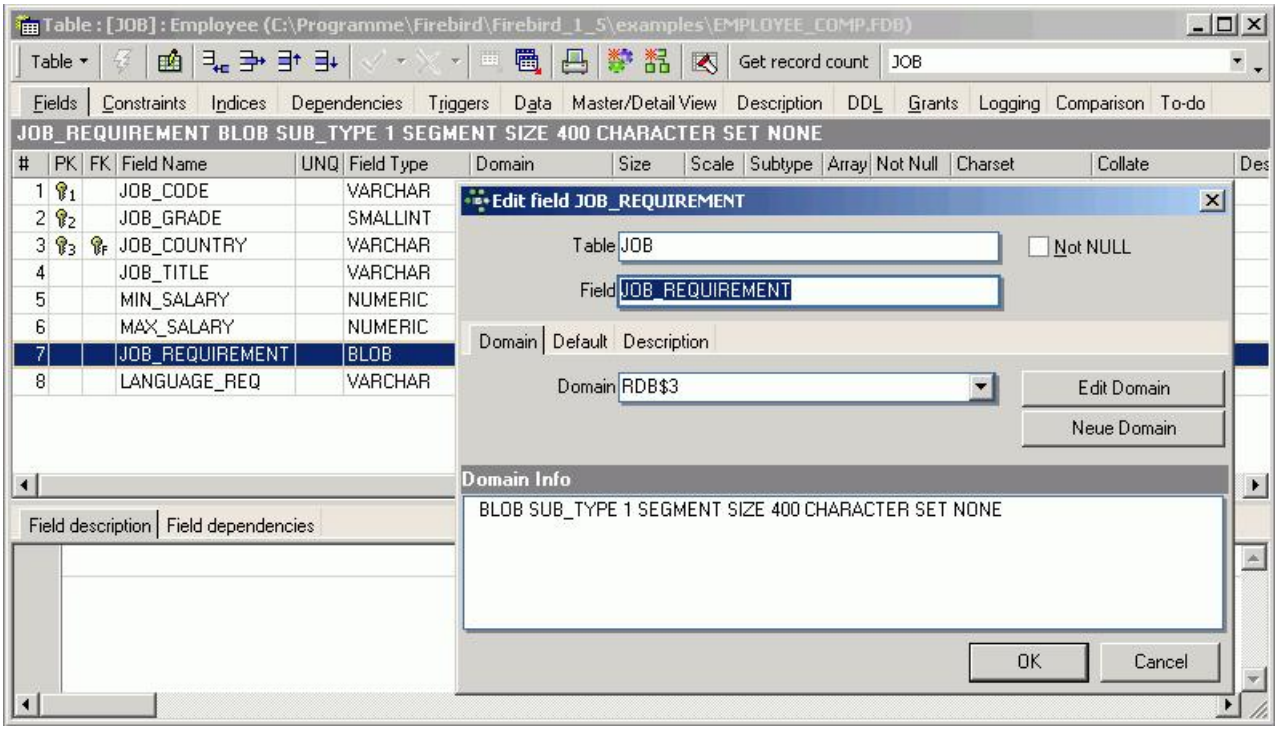
Blobs are defined as table columns. Their memory size is almost unlimited as they can be stored across several pages. This assumes however that a sufficient database page size has been specified. For example, using a 1k page, the blob may not exceed 0.5 GB, using a 4k page size, the blob size is limited to 8GB.

The ability to store such binary data in a database provides a high level of data security, data backup, version management, categorization and access control.

The advantage of blob text fields over VARCHAR fields (e.g. `VARCHAR (32000)`) is that a network protocol transfers all 32,000 VARCHAR characters when using an ISDN connection (analog lines compress the data to an extent). With a blob field, only the actual file size is transferred. Although - since Borland InterBase version 6.5/7 this disadvantage with VARCHAR datatype transfer has been solved, i.e. in these newer InterBase versions the full VARCHAR length including spaces is no longer transferred each time across the network. However, even here, blobs are still more effective when working with such large data sizes.

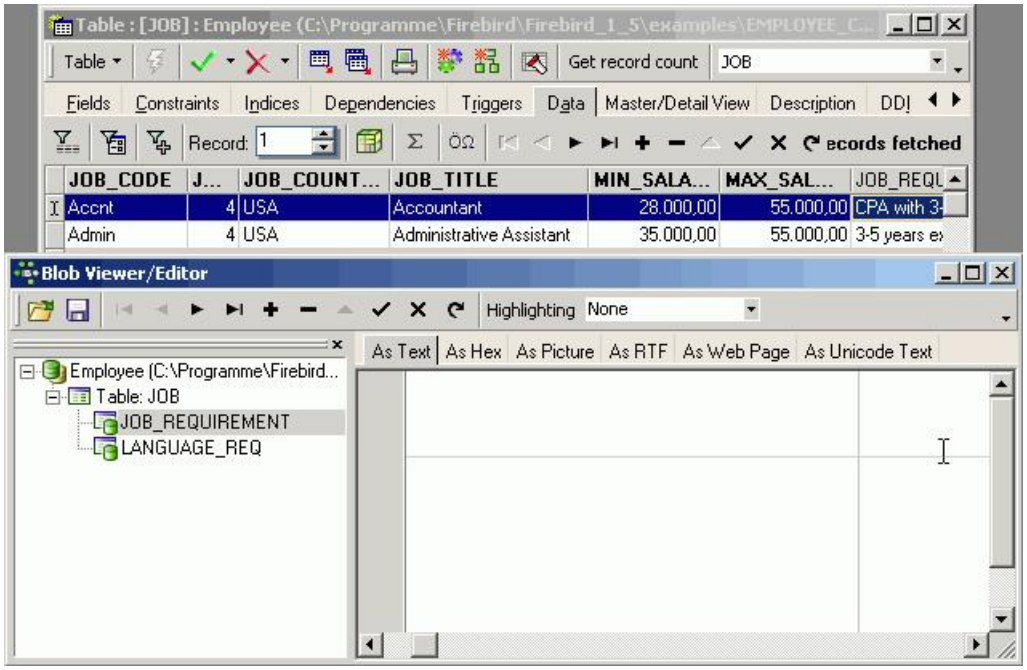
InterBase/Firebird supports quick and efficient algorithms for reading, writing and updating blobs. The user can manipulate blob processing with blob routines - also called blob filters. These filters are ideal tools for the compression and translation of blobs, depending upon the application requirements.

Blobs can be specified using the IBExpert DB Explorer or the IBExpert SQL Editor.



Blob specification includes the subtype, segment size and, if wished, the character set.

When the *Data View* (i.e. Data page) in the Table Editor is selected, and the table shown contains a blob column, IBExpert can display the blob content of a selected data set as text (also as RTF), hex, images and web pages using the IBExpert menu item Tools / Blob Viewer/Editor.



It is important when using blobs in a database, to consider the database page size carefully. Blobs are created as part of a data row, but because a blob could be of unlimited length, what is actually stored with the data row is a BlobID, the data for the blob is stored separately on special blob pages elsewhere in the database.

The BlobID is an 8 byte value that allows InterBase/Firebird to uniquely identify a blob and locate it. The BlobIDs can be either temporary or permanent; a temporary blob is one which has been created, but has not yet been stored as part of a table, permanent blobs have been stored in a table. The first 4 bytes represent the relation ID for the blob (like data rows, blobs are bound to a table), the second four bytes represent the ID of the blob within the table. For temporary blobs the relation ID part is set to 0.

A blob page stores data for a blob. For large blobs, the blob page could actually be a blob pointer page, i.e. be used to store pointers to other blob pages. For each blob that is created a blob record is defined, the blob record contains the location of the blob data, and some information about the blob's contents that will be useful to the engine when it is trying to retrieve the blob. The blob data could be stored in three slightly different ways. The storage mechanism is determined by the size of the blob, and is identified by its level number (0, 1 or 2). All blobs are initially created as level 0, but will be transformed to level 1 or 2 as their size increases.

A level 0 blob, is a blob that can fit on the same page as the blob header record, for a data page of 4096 bytes, this would be a blob of approximately 4052 bytes (page overhead - slot - blob record header).

Although the documentation states that the segment length does not affect the performance of InterBase/Firebird, the actual physical size of a blob, or its segment length can become useful in trying to improve I/O performance for the blob, especially if you can size the segment (typically) or blob to a page.

This is especially true if you plan to manipulate the blob using certain low level InterBase/Firebird blob calls. When a blob is too large to fit on a single page (level 1), and the data will be stored on one or more blob data pages, then the initial page of the blob record will hold a vector of blob page numbers.

A level 2 blob occurs when the initial page of the blob record is not big enough to contain the vector of all the blob data page numbers. Then InterBase/Firebird will create blob pointer pages, i.e. multiple vector pages that can be accessed from the initial blob header record, that now point to blob data pages.

The maximum size of a level 2 blob is a product of the maximum number of pointer pages, the number of data pages per pointer page, and the space available on each data page.

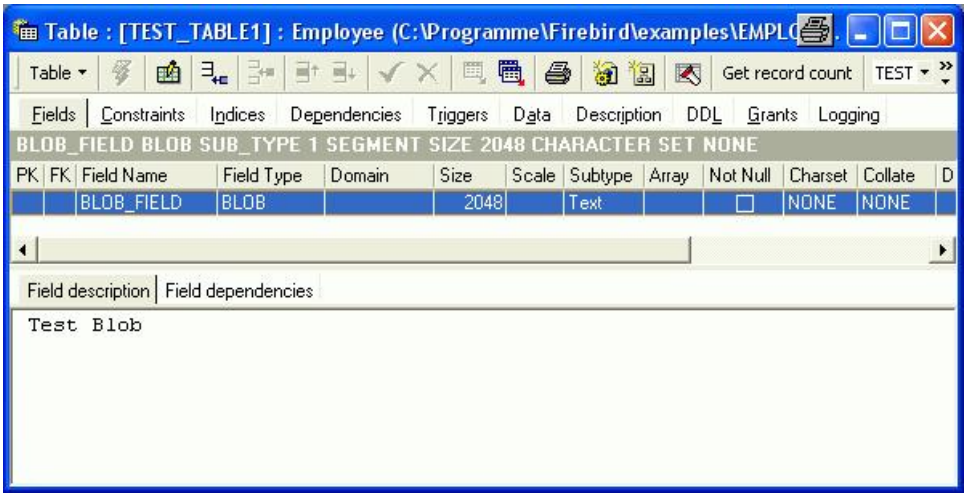
Max Blob Size:

- 1Kb page size => 64 Mb
- 2Kb page size => 512 Mb
- 4Kb page size => 4 Gb
- 8Kb page size => 32 Gb
- 16kb page size => Big enough :-).

We would like to thank Paul Beach of IBPhoenix, for allowing us to reproduce excerpts of his session, *Using and Understanding Blobs*, held at the European Firebird Conference 2003.

Segment size

Segment sizes are specified for blob fields. This can be done using the Domain Editor or the Table Editor (started from the IBExpert DB Explorer).



A blob segment size can be defined, to increase the performance when inputting and outputting blob data. This should roughly correspond to the datatype size. With a memo field, for example, for brief descriptions which could however, in individual cases, be considerably longer, the segment length could be defined as 100 bytes, whereby the blob datatype is processed in 100 byte blocks.

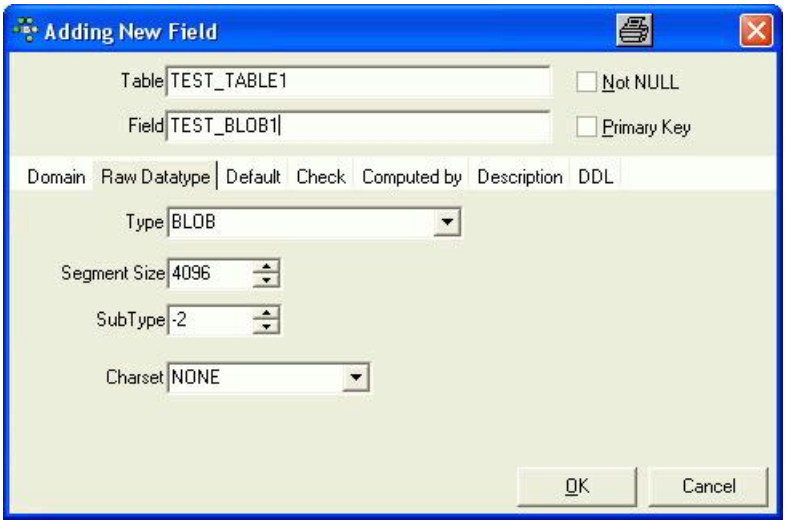
When processing videos or large graphics in the database, a large segment length should be selected. The maximum length is 65536 bytes. This is because all blob contents are stored in blocks, and are fetched via these blocks. A typical segment size from the old days is 80 (because 80 characters fit onto one monitor line).

When a blob is extracted, the InterBase/Firebird server reads the number of segments that the client has requested. As the server always selects complete blocks from the database, this value can in effect be ignored on modern powerful computers. 2048 is recommended as a standard since version InterBase 6.

Subtype

Subtypes are specified for blobs. They are used to categorize the datatype when defining blobs. A subtype is a positive or negative numerical value, which indicates the type of blob data. The following subtypes are predefined in InterBase/Firebird:

Subtype	Meaning
0	Standard blob, non-specified binary data
1	Text blob, e.g. memo fields
Text	Alternative for defining subtype 1
Positive value	Reserved for InterBase
Negative value	User-defined blob subtypes



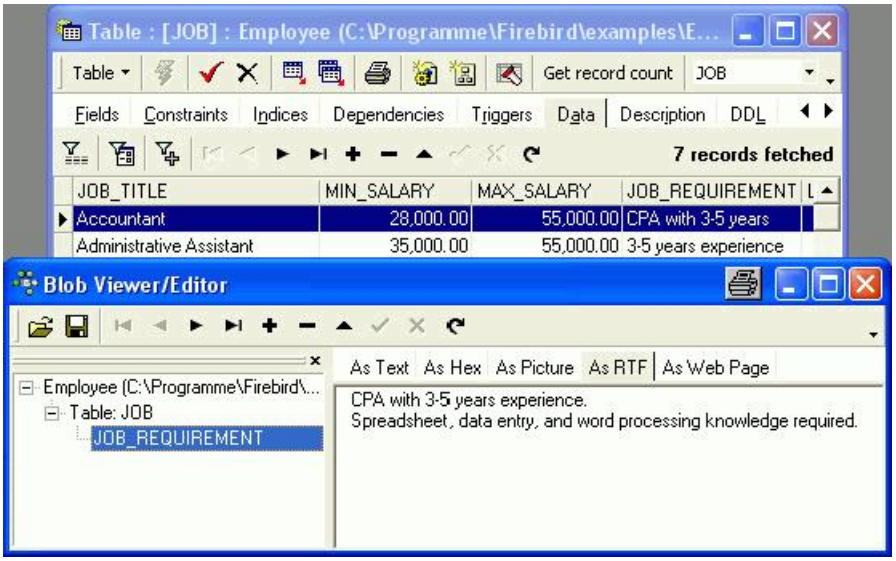
Blob fields can be specified using the Domain Editor or the Table Editor (started from the IBExpert DB Explorer).

The specification of a user-defined blob subtype has no effect upon InterBase/Firebird, as the InterBase/Firebird server treats all blob fields the same, i.e. it simply stores the data and delivers it to the client program when required.

The definitions are however required by the client programs in order to display the blob content correctly. For example, `SUB_TYPE -200` could be defined as a subtype for `GIF` images and `SUB_TYPE -201` as a subtype for `JPG` images.

Subtype specification is optional; if nothing is specified, InterBase/Firebird assumes 0 = binary data.

Under the menu item Tools, the IBExpert Blob Viewer/Editor can display blob contents as text, hex, images, RTF and web pages.



CHAR and VARCHAR

InterBase/Firebird provides two basic datatypes to store text or character information: `CHAR` and `VARCHAR` (blobs also allow character storage using the subtype text).

`CHAR` and `VARCHAR` are datatypes which can store any text information. Numbers that are not calculated, such as zip codes, are traditionally stored in `CHAR` or `VARCHAR` columns. The length is defined as a parameter, and can be between 1 and 32,767 bytes. It is particularly useful for codes that typically have a fixed or predefined length, such a the zip code for a single country.

Compared to most other databases, InterBase/Firebird only stores significant data. If a column is defined as `CHAR(100)`, but only contains entries with 10 characters, the additionally defined bytes are not used, as InterBase/Firebird stores `CHAR` and `VARCHAR` types similarly, and does not fill unused spaces with blanks. Both `CHAR` and `VARCHAR` are stored in memory buffer in their full, declared length; but the whole row is compressed prior to storing i.e. `CHARS`, `VARCHARSS`, `INTEGERS`, `DATESS`, etc. all together.

Indeed, `VARCHAR` columns require more storage than `CHAR` columns, because when storing a `VARCHAR`, InterBase/Firebird adds two bytes that state just how big the `VARCHAR` actually is.

So a `CHAR` will in fact be stored in a smaller space. However, when a `SELECT` is performed on a `VARCHAR` column, InterBase/Firebird strips the 2 byte padding and returns the stored value. When a `SELECT` is performed on a `CHAR` column, InterBase/Firebird returns the value and the "empty spaces". Thus the two bytes saved in storage of a `CHAR` must be balanced against the subsequent need to strip the spaces on the client side. These two bytes however are, with today's hardware, too negligible to have an influence upon the database performance. This can however be disadvantageous when defining short text fields.

In practical terms consider just this one rule: only use CHARS if strings of few characters are to be stored; the exception to the rule being when working with intermediate tables that are required to export data to fixed length prn files. Then the fixed length field will be a positive advantage.

This efficient storage in InterBase/Firebird can lead to considerable confusion particularly when importing data, as Paradox or dBASE databases save all blank spaces, and after importing a 10MB dBASE file into InterBase, often only 3-6 MB remain, although all data sets were imported correctly.

For this reason columns can be defined generously in InterBase/Firebird without a problem, whereas in other databases each defined byte influences the size of the database, regardless of whether data is stored in these fields or not.

Please note however that indexed CHAR fields should not be more than approx. 80 characters in length (with Firebird 1.5 the limit is somewhat higher).

The CHAR datatype definition can be written in two ways:

```
CHAR
CHARACTER
```

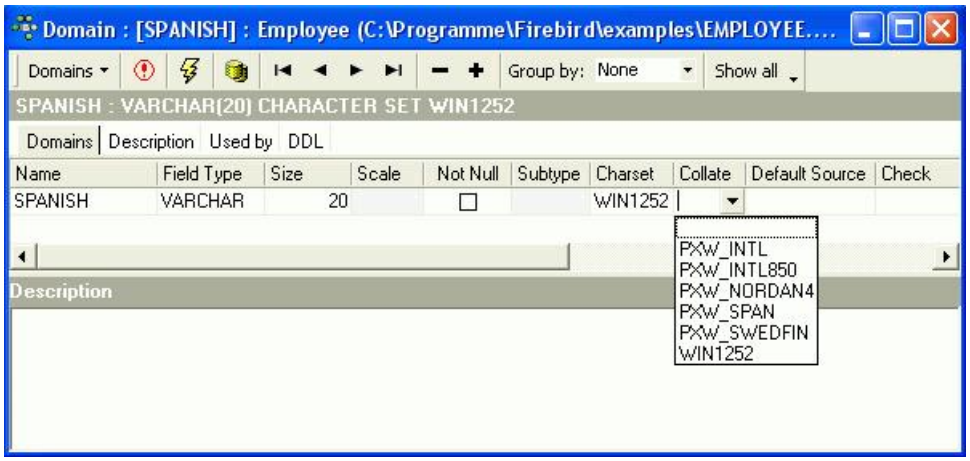
The VARCHAR datatype definition can be written as follows:

```
VARCHAR
CHARACTER VARYING
CHAR VARYING
```

Collate

A special collation sequence can be specified for CHAR and VARCHAR field columns. The COLLATE parameter allows fields to be collated according to a certain language/group of languages e.g. collate according to the German language when using Win1252.

In IBExpert the collation sequence can be specified when defining the character set for a domain or field:



The collation options are offered in IBExpert in a pull-down list, after specifying the character set.

In DDL it is specified using the keyword COLLATE and the respective character set table, for example:

```
CREATE DOMAIN dom_city VARCHAR(20)
COLLATE PXW_INTL850;

CREATE DOMAIN User_Name VARCHAR(20)
CHARACTER SET DOS437
DEFAULT USER
NOT NULL
COLLATE PDOX_ASCII
```

The parameter sequence is important, as the collation sequence must be specified last.

NCHAR and NVARCHAR

NCHAR or NATIONALCHARACTER
NVARCHAR or NATIONAL CHAR VARYING or NATIONAL CHARACTER VARYING

NCHAR/VARCHAR are datatypes, which can be defined as the NCHAR/VARCHAR datatypes with a length of 1-32,767 bytes. The only difference to the NCHAR/VARCHAR datatype is that NCHAR/VARCHAR automatically defines a special character set for this table column: "CHARACTER SET ISO8859_1".

INTEGER, SMALL INTEGER and BIG INTEGER (Int, SmallInt and BigInt)

INTEGER datatypes are used to store whole numbers. SMALLINT is the abbreviation for small integer. BIGINT was added in Firebird 1.5 and is the SQL99-compliant 64-bit signed integer type. BIGINT is available in Dialect 3 only.

Values following the decimal point are not allowed. Depending upon the numeric area required, following INTEGER types are supported:

Type	Size	Value Range
SmallInt	2 bytes	-32,768 to +32,767
Integer	4 bytes	-2,147,483,648 to +2,147,483,647
BinInt	64 bytes	-2^63^ to 2^63^-1 *or* -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

4 bytes of data storage are required for the INTEGER value, whereby 31 bits are for the number and 1 bit for the sign. 2 bytes of data storage are required for the small integer value, whereby 15 bits are for the number and 1 bit for the sign. It is usually preferable to use an INTEGER datatype as 2 bytes more or less are fairly irrelevant these days.

An INTEGER is a 15-digit number and although extremely large, is by far not as large as the NUMERIC(18). INTEGER types are particularly suited for unique identification numbers, as InterBase/Firebird contains mechanisms for the automatic generation of whole number values (please refer to generator for further information). The resulting indices for the connection of multiple tables to each other are relatively small and offer extremely quick access, as the highest computer performance on all computer platforms is generally found in INTEGER operations. It is possible to specify the display format of an INTEGER under Environment Options / Grid / Display Formats.

SMALLINTS can also be used for BOOLEAN datatypes e.g. true/false, male/female.

FLOAT and DOUBLE PRECISION

FLOAT datatypes are used to store values with significant decimals. The following FLOAT types are supported:

Type	Size	Value range
Float	4 bytes	7 significant decimals; -3.4 x 10^-38 to 3.4 x 10^38
Double Precision	8 bytes	15 significant decimals; -1.7 x 10^-308 to 1.7 x 10^308

A column with the defined datatype FLOAT can store a single-precision figure with up to 7 significant decimals. The decimal point can float between all seven of these digits. If a number with more than 7 decimal places needs to be saved, decimals beyond the seventh position are truncated. FLOAT columns require 4 bytes of storage.

A column with the defined datatype DOUBLE PRECISION can store numbers with 15 significant decimals. This uses 8 bytes of storage. As with the FLOAT column, the decimal point can float within the column. The DOUBLE PRECISION datatype is implemented in the majority of InterBase platforms as a 64 bit number.

FLOAT types can be implemented for any calculative operations. They offer an optimal performance and sufficient range of values. It is possible to specify the display format of a FLOAT field under Environment Options / Grid / Display Formats.

The DOUBLE PRECISION datatype can be written as follows:

```
DOUBLE PRECISION
DOUBLE
```

The main advantage of a DOUBLE PRECISION datatype is the large number of decimal places e.g. 1/3 in DOUBLE PRECISION would be 0,33333333333333 in NUMERIC(18,4) it would be 0,3333. Please note: up until dialect 1 NUMERIC and DOUBLE PRECISION were identical i.e. an SQL with the datatype NUMERIC(15,2) results in the following:

Result with dialect 1:

```
CREATE TABLE TEST(WERT NUMERIC(15,2));
INSERT INTO TEST(WERT) VALUES(100);
SELECT * FROM TEST;  result 100
UPDATE TEST SET WERT=WERT/3;
SELECT * FROM TEST;  result 33,33
UPDATE TEST SET WERT=WERT*3;
SELECT * FROM TEST;  result 100
```

Result with dialect 3:

```
CREATE TABLE TEST(WERT NUMERIC(15,2));
INSERT INTO TEST(WERT) VALUES(100);
SELECT * FROM TEST;  result 100
UPDATE TEST SET WERT=WERT/3;
SELECT * FROM TEST;  result 33,33
UPDATE TEST SET WERT=WERT*3;
SELECT * FROM TEST;  result 99,99
```

Since dialect 3 NUMERIC data is rounded according to commercial rounding rules; up to dialect 1 NUMERIC data is rounded according to technical rounding rules.

NUMERIC and DECIMAL

The NUMERIC datatype specifies a numeric column where the value has a fixed decimal point, such as for currency data. NUMERIC(18) is a 64-bit integer value in SQL dialect 3 and is almost infinite. Since SQL dialect 3 numeric and decimal datatypes are stored as INTEGERS of the respective size.

SQL dialect 1 offers `NUMERIC(15)`.

Syntax:

```
NUMERIC(precision, scale);
```

or

```
DECIMAL(precision, scale);
```

`PRECISION` refers to the total number of digits, and `SCALE` refers to the number of digits to the right of the decimal point. Both numbers can be from 1 to 18 (SQL dialect 1: 1-15), but `SCALE` must be less than or equal to `PRECISION`.

It is better to define `NUMERIC` always at its maximum length, as in this case, the 32 bit `INTEGER` value is used. Otherwise a 16 bit value is used internally, for example with `NUMERIC(4,2)`, and this is not always transformed back correctly by the client program environments (an older BDE version could, for example, transform Euro 12.40 with `NUMERIC(4,2)` into Euro 1,240).

InterBase/Firebird supports a number of options for specifying or not specifying `PRECISION` and `SCALE`:

1. If neither `PRECISION` nor `SCALE` are specified, InterBase/Firebird defines the column as `INTEGER` instead of `NUMERIC` and stores only the integer portion of the value.

2. When using SQL dialect 1, if just `PRECISION` is specified, InterBase/Firebird converts the column to a `SMALLINT`, `INTEGER` or `DOUBLE PRECISION` datatype, based on the number of significant digits being stored.

In SQL dialect 3, if just `PRECISION` is specified, InterBase/Firebird converts the column to a `SMALLINT`, `INTEGER` or `INT64` datatype, based on the number of significant digits being stored.

It is important to distinguish between the two dialects, because since `INT64` is an `INTEGER` datatype, and `DOUBLE PRECISION` is not, you will occasionally have rounding errors in SQL dialect 1, but not in SQL dialect 3 or later.

The `NUMERIC` datatype should only be used for fields that are later to be used as part of a calculation.

InterBase/Firebird converts the columns as follows:

Definition	Datatype Created
Decimal(1)-Decimal(4)	Small Integer
Decimal(5)-Decimal(9)	Integer
Decimal(10)-Decimal(18)	Int (64)

Note that if a `DECIMAL(5)`datatype is specified, it is actually possible to store a value as high as a `DECIMAL(9)` because InterBase/Firebird uses the smallest available datatype to hold the value. For a `DECIMAL(5)` column, this is an `INTEGER`, which can hold a value as high as a `DECIMAL(9)`.

DATE

The `DATE` datatype stores values which represent a date. InterBase/Firebird supports a single `DATE`-type column that requires 8 bytes of storage space. It uses 4 bytes for the date and 4 bytes for the time.

Valid dates are from January 1, 100 AD through February 28, 32,767 AD. Note: for `DATE` arithmetic purposes, `DATE 0` (the integer value of zero) as a `DATE` in InterBase/Firebird is November 17, 1898.

Different date formats are supported. There are however slight differences between SQL dialect 1 and SQL dialect 3.

- SQL dialect 1: `DATE` also includes a time slice (equivalent to `TIMESTAMP` in dialect 3).
- SQL dialect 3: `DATE` does not include any time slice.

Using SQL dialect 1 the default `NOW` for datatype `DATE` means current time and date of the server; there is also `TODAY` (only date; the time is always set at midnight, `YESTERDAY`, `TOMORROW`).

Example:

```
SELECT CAST ("NOW" AS DATE) FROM RDB$DATABASE
```

`SELECT CAST` is an SQL dialect 1 command (although it also functions in SQL dialect 3); `SELECT` is used in SQL dialect 3. These values are primarily compatible to older InterBase versions. When working with SQL dialect 3, the `CURRENT_` constants (see below) should be used as far as possible.

From InterBase 6 upwards and Firebird there are the following for dialect 3: `CURRENT_TIME`, `CURRENT_TIMESTAMP`, `CURRENT_DATE` (without quotation marks and without `CAST`). Example:

```
SELECT CURRENT_DATE-1 FROM RDB$DATABASE
```
Result: the date yesterday, etc.

```
SELECT CURRENT_TIMESTAMP-(1/24) FROM RDB$DATABASE
```
Result: the current time minus one hour (one twenty-fourth of a day).

It is possible to specify the display format of a date field under Environment Options / Grid / Display Formats. For the various options available, please refer to Date Time Format.

TIME

The `TIME` datatype is new to InterBase v 6.0. It is an SQL dialect 3 datatype. `TIME` is a 32-bit field type of `TIME` values. The range is from `0:00 AM` to `23:59:9999 PM`.

It is possible to specify the display format of a date field under Environment Options / Grid / Display Formats. For the various options available, please refer to Date Time Format.

TIMESTAMP

`TIMESTAMP` is new to InterBase v 6.0. It is an SQL dialect 3 datatype. `TIMESTAMP` is a 64-bit field type comprised of both date and time. The range is from January 1,100 AD to February 28, 32768 AD. It is the equivalent of `DATE` in SQL dialect 1.

It is possible to specify the display format of a date field under Environment Options / Grid / Display Formats. For the various options available, please refer to Date Time Format.

New to Firebird 2.0: `CURRENT_TIMESTAMP` now returns milliseconds by default

The context variable `CURRENT_TIMESTAMP` now returns milliseconds by default, while it truncated sub-seconds back to seconds in former versions. If you need to continue receiving the truncated value, you will now need to specify the required accuracy explicitly, i.e. specify `CURRENT_TIMESTAMP(0)`.

Array

InterBase/Firebird allows a column to be defined as an array of elements, i.e. data information can be stored in so-called arrays. An array is a range of values determined by setting a lower and an upper limit. An array consists of any amount of information that can be split into different dimensions. The array can be managed as a whole, as a series of elements in one dimension of the array, or as individual elements.

Arrays should be used with caution. Database normalization usually supplies an alternative format for storing such data, so that normal table structures are just as suitable, and also preferable. There are however occasionally exceptions, for example for measurement value logging, when arrays are the preferred option.

The array datatype is used relatively seldom, as it is not very simple to process, and does not really conform to the typical demands of an SQL database (usually one or more detail tables would be created, and not an array).

Arrays can be declared as a domain or directly in the table definition following the datatype definition. Array data can be of any type except blob. Between 1 and 16 dimensions can be specified; each dimension can store as many elements as can be fitted into the database. The values are stored as a blob and are therefore almost unlimited in scope.

The only difference compared to the normal datatype definition is the specification of the dimensions in square brackets, each dimension being separated by commas. By default, the lower bounds ID number is 1 and the upper bounds ID number is the maximum of that dimension. Alternate bounds IDs can be specified in place of the array size by separating them with a colon. For example, an array with 5 measurements with 2 dimensions starting at the default value 1 is defined as follows:

```
[2,5]
```

Counting begins at 1 and ends at the value entered by the user. In this case 2x5=10 measurements can be logged. If counting is to begin at, for example, 0, the array definition is as follows:

```
[0:2, 0:5]
```

One-dimensional arrays

Definition: `NAME DATATYPE [LOWER_DIMENSION:UPPER_DIMENSION]`
Example: `LANGUAGE_REQ VARCHAR(15) [1:5]`

In this field 5 data entries of the `VARCHAR(15)` type can be stored. `LANGUAGE_REQ[1]` up to `LANGUAGE_REQ[5]` can be accessed.

Multi-dimensional arrays

Definition: `NAME DATATYPE [LOWER_DIMENSION1:UPPER_DIMENSION1] [LOWER_DIMENSION2:UPPER_DIMENSION2]`

Example: `DAILY_MEASUREMENTS NUMERIC(18,2) [1:24][1:365]`

When using arrays, it is important to be aware of the advantages and limitations.

Advantages of arrays

1. InterBase operations can be performed upon the total datatype as a single element. Alternatively operations can be executed on part of an array only for certain values of a dimension. An array can also be broken down into each single element.

2. Following operations are supported:

- SELECT statement from array data.
- Insertion of data in an array.
- Updating data in an array slice.
- Selecting data from an array slice.
- Examination of an array element in a SELECT statement.

Array limitations

1. A user-defined function can only access one element in an array.

2. The following operations are not supported:

- Dynamically referencing array dimensions using SQL statements.
- Inserting data into an array slice.
- Setting individual array elements to null.
- Using aggregate functions such an MIN(), MAX(), SUM(), AVG() and COUNT() on arrays.
- Referencing an array in the GROUP BY clause in a SELECT query.
- Creating a view, which selects from array slices.

3. The data stored in this way cannot be selected per index; each query always accesses the fields unindexed.

Boolean

InterBase/Firebird does not offer a native BOOLEAN datatype. However, they can be implemented using domains.

The first step is to define a domain (which should logically be named Boolean). The domain can be defined in one of two ways:

1. Using a SMALLINT (16 bits), defaulting to zero, with a check constraint to ensure only the values of zero or one are entered. i.e:

```
CREATE DOMAIN D_BOOLEAN AS SMALLINT DEFAULT 0
CHECK (VALUE BETWEEN 0 AND 1);
```

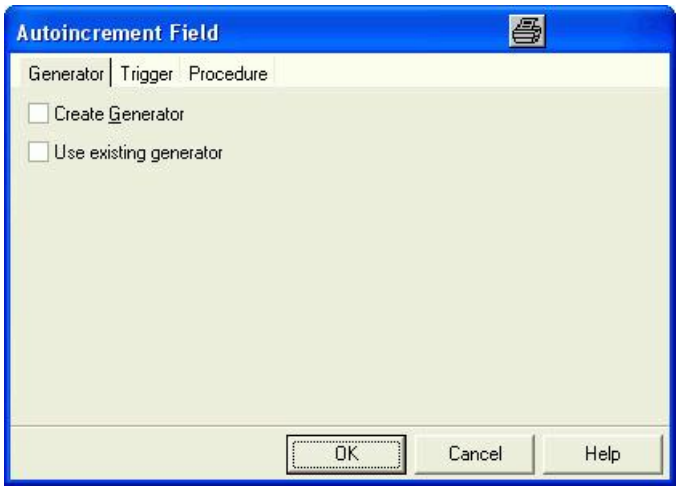
Once you have defined this domain you can forever use it as a BOOLEAN datatype without further concern. It is particularly suitable from a Delphi point of view, as Pascal BOOLEANs work in a similar manner.

2. Alternatively, the domain can be defined as a CHAR(1) and appropriate single character values ensured using a check constraint. If T and F or Y and N are more meaningful for your application then use this approach.

We'd like to thank Paul Beach of IBPhoenix for this article about Boolean datatypes.

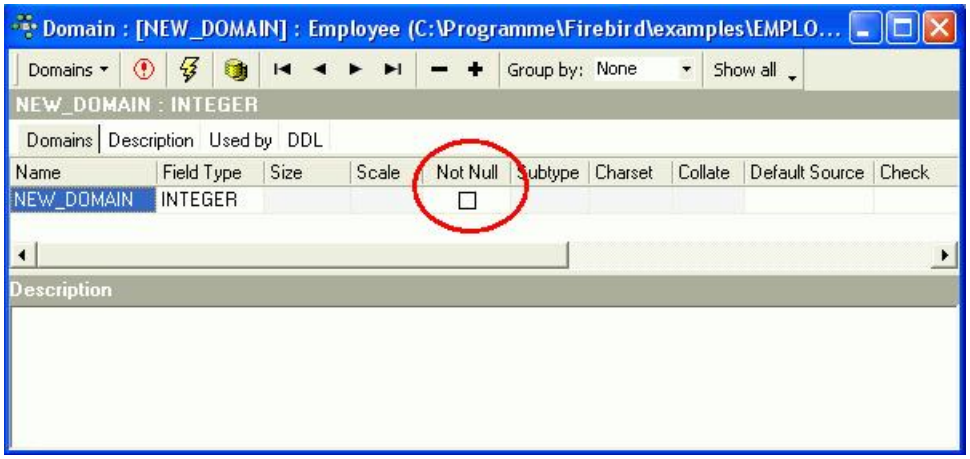
Autoincrement

An autoincrement is an automatic counter/calculator, such as a generator, trigger or stored procedure.



NOT NULL

NOT NULL is a parameter that does not allow a column field to be left blank. It can be defined for a field or a domain.

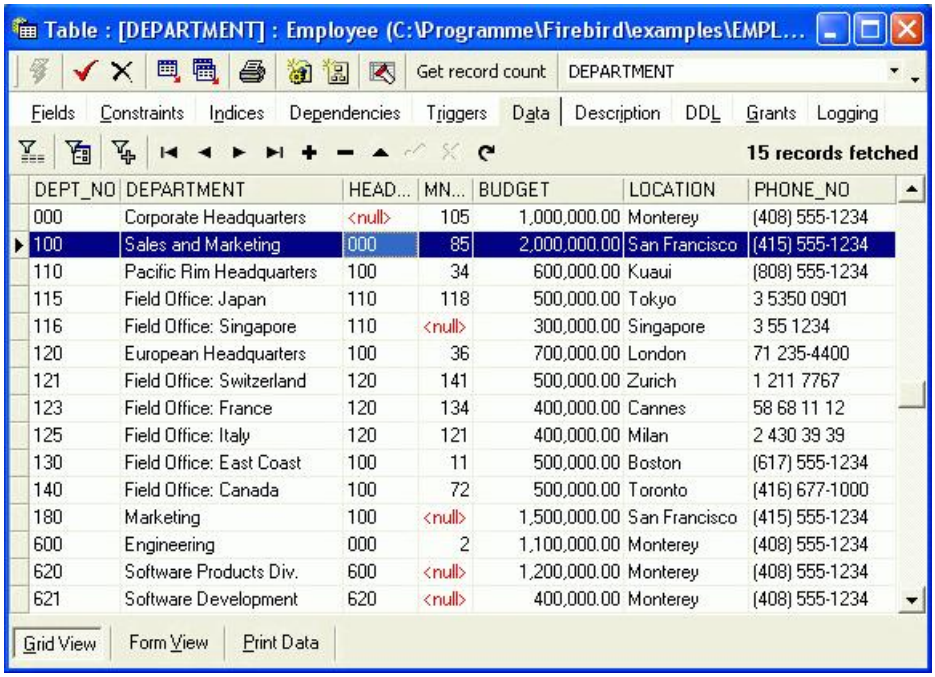


It forces a value to be entered into the column. It operates in the same way for tables as for domains. The parameter `DEFAULT NULL` and `NOT NULL` cannot be used in the same column definition. The `NOT NULL` parameter must be specified if the column is to be defined as `PRIMARY KEY` or `UNIQUE`.

NULL

`NULL` is the term used to describe a data field without a value, i.e. the field has been left blank because the information is either not known or not relevant for this record/data set. The `NULL` value can be stored in text, numeric and date datatypes.

A relational database is able to store `NULL` values as data content. A `NULL` value does not mean numerical zero. For example, a product can have zero sales (`0`) or unknown sales (`<null>`).



DEPT_NO	DEPARTMENT	HEAD...	MN...	BUDGET	LOCATION	PHONE_NO
000	Corporate Headquarters	<null>	105	1,000,000.00	Monterey	(408) 555-1234
100	Sales and Marketing	000	85	2,000,000.00	San Francisco	(415) 555-1234
110	Pacific Rim Headquarters	100	34	600,000.00	Kuaui	(808) 555-1234
115	Field Office: Japan	110	118	500,000.00	Tokyo	3 5350 0901
116	Field Office: Singapore	110	<null>	300,000.00	Singapore	3 55 1234
120	European Headquarters	100	36	700,000.00	London	71 235-4400
121	Field Office: Switzerland	120	141	500,000.00	Zurich	1 211 7767
123	Field Office: France	120	134	400,000.00	Cannes	58 68 11 12
125	Field Office: Italy	120	121	400,000.00	Milan	2 430 39 39
130	Field Office: East Coast	100	11	500,000.00	Boston	(617) 555-1234
140	Field Office: Canada	100	72	500,000.00	Toronto	(416) 677-1000
180	Marketing	100	<null>	1,500,000.00	San Francisco	(415) 555-1234
600	Engineering	000	2	1,100,000.00	Monterey	(408) 555-1234
620	Software Products Div.	600	<null>	1,200,000.00	Monterey	(408) 555-1234
621	Software Development	620	<null>	400,000.00	Monterey	(408) 555-1234

A `NULL` value can occur for the following reasons:

- The value is not yet known, but will be added at a future date.
- The value is not yet available for some reason, e.g. the date of receipt of payment.
- The value is not important, e.g. the credit card expiry date of someone who has paid cash.

InterBase/Firebird does not use a special byte sequence to indicate a `NULL`, but administrates this information internally. `NULL` values can influence query contents considerably, for example, when a column average is calculated. The values filled by the `NULL` value, i.e. empty fields, are not taken into consideration. A field containing the value 0 is included in the calculation of the average.

Examples from the Firebird 1.5 Quick Start Guide:

- `1 + 2 + 3 + NULL = NULL`
- `not (NULL) = NULL`
- `'Home ' || 'sweet ' || NULL = NULL`
-
```
if (a = b) then
        MyVariable = 'Equal';
else
        MyVariable = 'Not equal';
```

After executing this code, `MyVariable` will be `Not equal` if both `a` and `b` are `NULL`. The reason is that the expression `a = b` yields `NULL` if at least one of them is `NULL`. In an `if...then` context, `NULL` behaves like `FALSE`. So the `then` block is skipped, and the `else` block executed.

```
if (a <> b) then
   MyVariable = 'Not equal';
else
   MyVariable = 'Equal';
```

Here, `MyVariable` will be `Equal` if `a` is `NULL` and `b` isn't, or vice versa. The explanation is analogous to that of the previous example.

```
FirstName || ' ' || LastName
```

will return `NULL` if either `FirstName` or `LastName` is `NULL`.

Think of `NULL` as `UNKNOWN` and all these strange results suddenly start to make sense! If the value of `Number` is unknown, the outcome of `1 + 2 + 3 + Number` is also unknown (and therefore `NULL`). If the content of `MyString` is unknown, then so is `MyString || YourString` (even if `YourString` is non-`NULL`). Etcetera.

New to Firebird 2.0: `NULL`s are now "lowest" for `SORTS`

`NULL` is now treated as the lowest possible value for ordering purposes and sets ordered on nullable criteria are sorted accordingly. Thus: .

- for ascending sorts `NULL`s are placed at the beginning of the result set,
- for descending sorts `NULL`s are placed at the end of the result set.

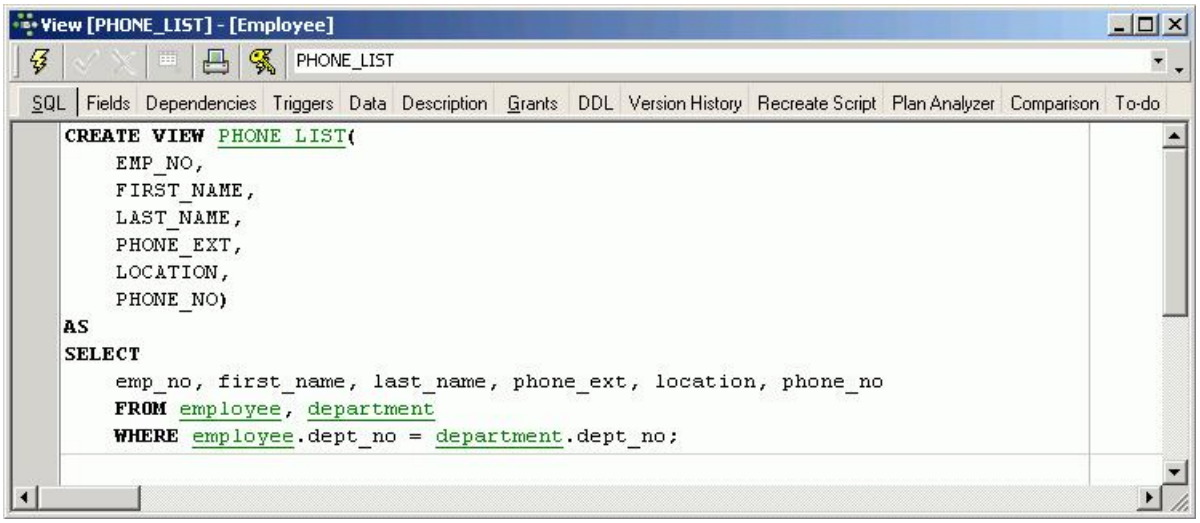
Important: In former versions, `NULL`s were always at the end. If you have client code or PSQL definitions that rely on the legacy `NULL`s placement, it will be necessary to use the `NULLS LAST` option in your `ORDER BY` clauses for ascending sorts.

Please also refer to the Firebird 2.0.4. Release Notes for further information regarding Enhancements to `NULL` logic in Firebird 2.

See also:
Table Editor
SQL Editor
Division of an integer by an integer
SQL Language Reference
Expressions involving `NULL`
Database Normalization

View

A view is a stored SELECT of one or more tables. The rows to be returned are defined by the SELECT statement that lists columns from the source tables. Only the view definition is stored in the database, it does not directly represent physically stored data. The WHERE command can also be used. A view has no input parameters.



It can be likened to a virtual table. The view can be treated, in almost all respects, as if it were a table, using it as the basis for queries and even updates in some cases. It is possible to perform SELECT, PROJECT, JOIN and UNION operations on views as if they were tables.

Views give end users a personalized version of the underlying tables in the database and also simplify data access, by protecting them from the details of how information is spread across multiple tables. They also provide security by hiding certain columns in the table(s) from various users. InterBase/Firebird allows user rights to be granted to the view and not the underlying table(s).

Advantage of views (and stored procedures): as these are part of InterBase or Firebird, it is irrelevant which front end is subsequently used, be it Delphi, PHP or other.

They allow the developer to denormalize data, combining information from two or more tables into a single virtual table. Instead of creating an actual table with duplicate data, a view can be created using SELECT, JOIN and WHERE. Even when you change the underlying structure of the tables concerned, the view remains consistent.

Views cannot be sorted, they merely display the result of a specified SELECT. (A view can therefore be compared to a saved query). The ORDER BY instruction cannot be used in a view (the data sets are displayed as determined by the optimizer, which is not always intelligent!). In such a case, a stored procedure would have to be used (stored procedures being more flexible in any case, and offering more control).

Views can be used, for example, for internal telephone lists, or when information from more than one table needs to be linked, e.g. the first modular result needs to be linked to the second result.

The underlying SELECT definition can contain all the performance features of a select query on tables, it is however subject to the following restrictions:

1. All columns must be explicitly specified, so that the view always returns the same columns in the correct order.

2. If reference is made to a SELECT * statement in a view, the result is returned in the column sequence of the definition of the underlying tables, and can therefore deliver different results should changes later be made to the table structure.

3. No ORDER BY statements may be used.

4. Indices can only be placed on the columns of the base tables, not the view columns. When the view is generated, these indices are automatically used.

Views allow a data modularization, particularly useful with complex data quantities, as another view can be incorporated in the view definition.

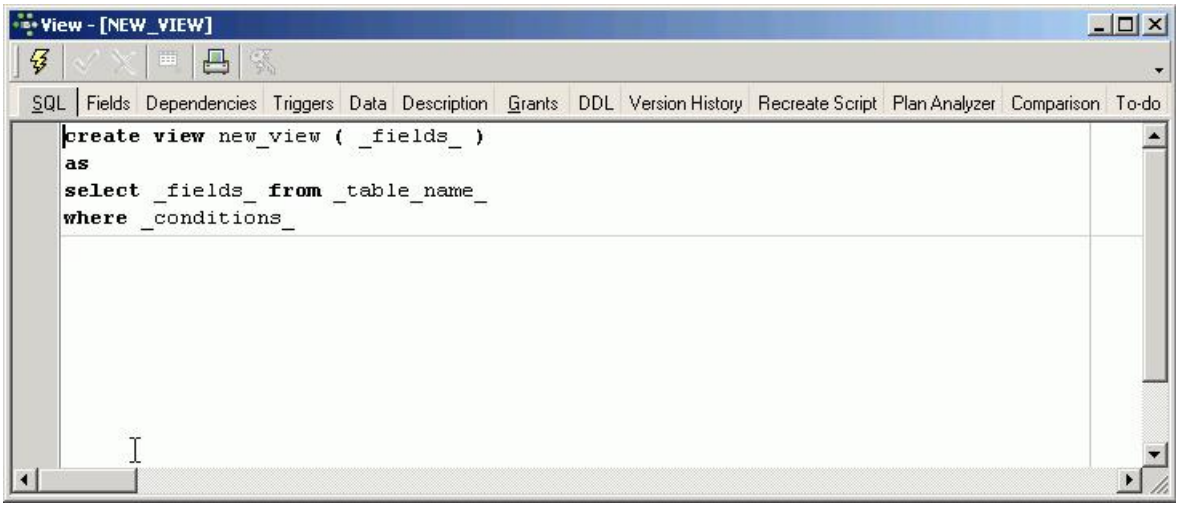
If you are new to database development, please refer to the chapter *Understanding and using views*.

New view / View Editor

A new view can be created in a connected database, either by using the menu item Database / New View, the respective icon in the New Database Object toolbar, or using the DB Explorer right mouse button (or key combination [Ctrl + N]), when the view heading of the relevant connected database is highlighted.

Alternatively, a new view can be created directly in the IBExpert SQL Editor, and then saved as a view.

A *NewView* dialog appears, with its own toolbar:



The view can be created directly in the SQL dialog, and subsequently committed using the respective icon or [Ctrl + F9].

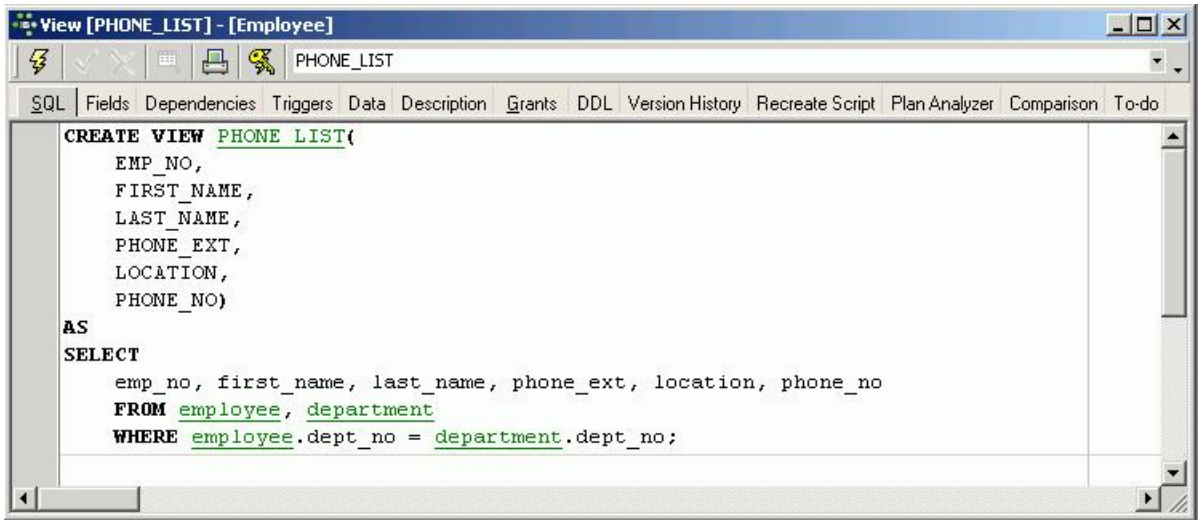
SQL

When creating a view it is necessary to define a view name that is unique in the database. All data manipulation operations such as SELECT, INSERT, UPDATE and DELETE are carried out using this name.

The view can then be created in the SQL dialog using the following syntax:

```
CREATE VIEW ViewName (<List_of_field_names>)
AS
SELECT <fields_ from _table_name>
[WITH CHECK OPTION];
```

An example can be viewed in the InterBase/Firebird sample EMPLOYEE database:



The view name must be unique. As InterBase/Firebird only stores the view definition (i.e. it does not copy the data from the tables into the view), views depend a lot upon indices set in the base tables, in order to locate data rapidly from the original tables. It is therefore important to analyze views carefully, and place indices on those columns that are used to join tables and to restrict rows.

The tables and fields can be easily inserted into the SQL script by dragging the relevant table and field names from the DB Explorer and SQL Assistant, and dropping them in the respective position in the SQL dialog in the New View Editor. After naming the view fields and inserting the relevant base table fields, the new view can be committed using the respective icon or [Ctrl + F9].

The view contents result from the returns of the `SELECT` statement that corresponds, with few exceptions, to the SQL `SELECT` command. The `SELECT` statement specifies which tables, columns and rows are to be returned as part of the view.

If the view is an updatable view, the optional `WITH CHECK OPTION` parameter may also be used to control data input.

The field names, as they are to appear in the view, can be optionally specified under a different name to the field names in the base tables. If no specification is made, the original base table column names automatically become the view field names. If column names are specified, they must be unique within the view and a name must be specified for every column returned by the view (even if some of the view field names correspond to the original field names). Please note that if the `SELECT` statement includes derived columns, column names must be specified.

If the view is to be used as part of a query, or indeed any other SQL statement, InterBase/Firebird queries the original data directly. This important feature offers the flexibility of being able to make alterations to the underlying database structure without affecting the user's view of the data or the view of any programs, which reference the view instead of the base tables.

Finally compile the new view using the respective toolbar icon or [F9], and, if desired, autogrant privileges, again using the respective toolbar icon or key combination [Ctrl + F8].

New to Firebird 2.0: Extensions to `CREATE VIEW` specification

`FIRST/SKIP` and `ROWS` syntaxes and `PLAN` and `ORDER BY` clauses can now be used in view specifications.

From Firebird 2.0 onward, views are treated as fully-featured `SELECT` expressions. Consequently, the clauses `FIRST/SKIP`, `ROWS`, `UNION`, `ORDER BY` and `PLAN` are now allowed in views and work as expected.

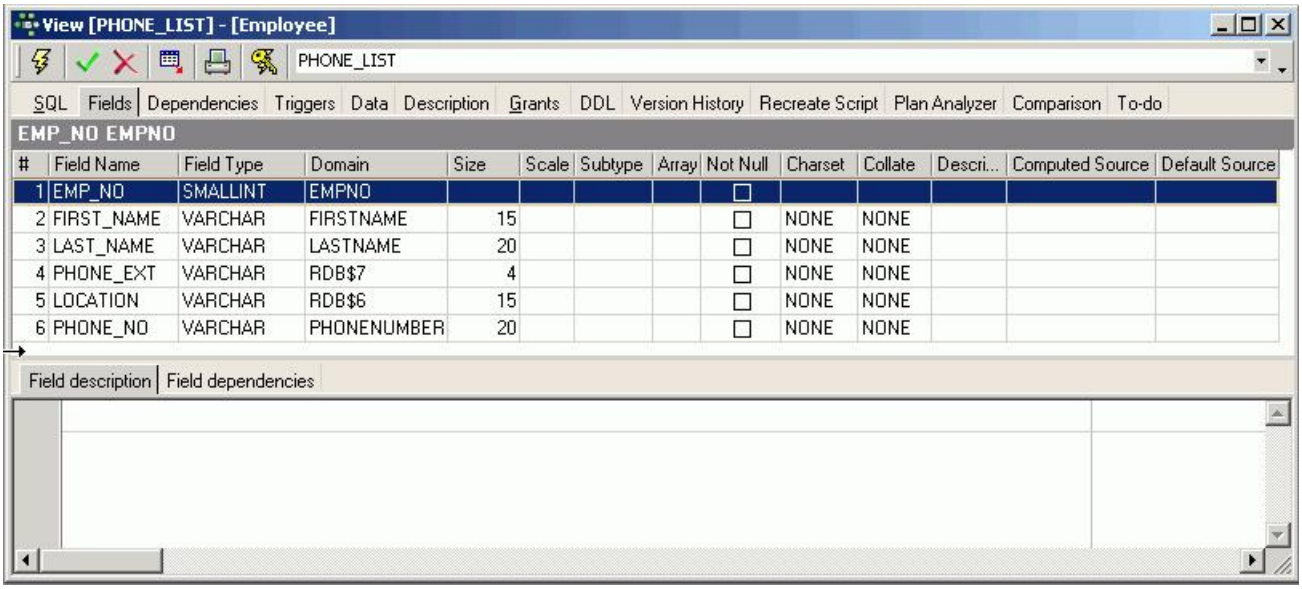
Syntax

For syntax details, refer to Select Statement & Expression Syntax in the Firebird 2.0.4 Release Notes chapter about DML.

See also:
`SELECT`
`SELECT` statement

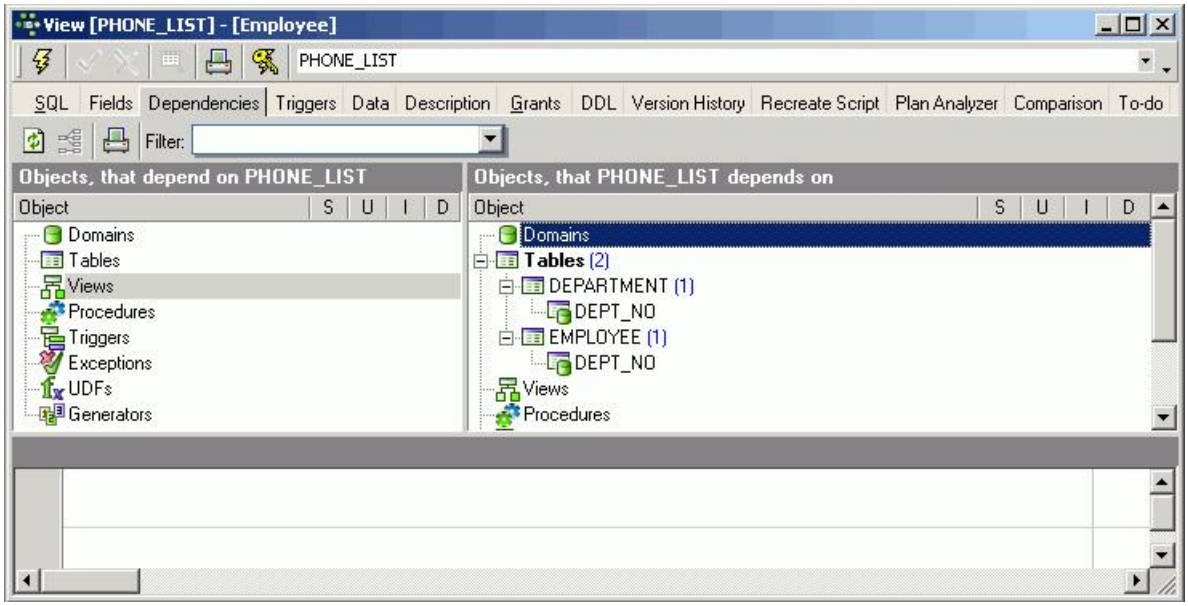
Fields

The *Fields* page displays the fields selected from the base table (with their new view names, if they have been specified), along with their properties.



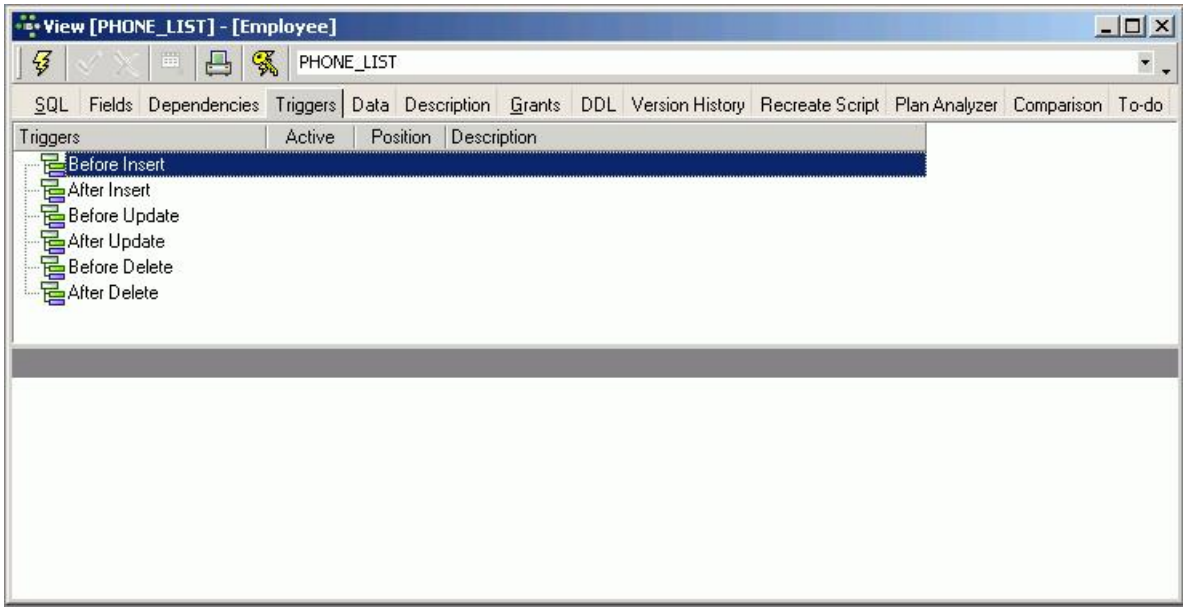
The individual fields may not be edited directly from this dialog; to alter fields, please refer to the Table Editor / Fields. These fields can however be sorted here into ascending or descending order based upon the column where the mouse is, by clicking on the column headers (i.e. *Field Name* etc.). By double-clicking on the right edge of the column header, the column width can be adjusted to the ideal width.

Dependencies



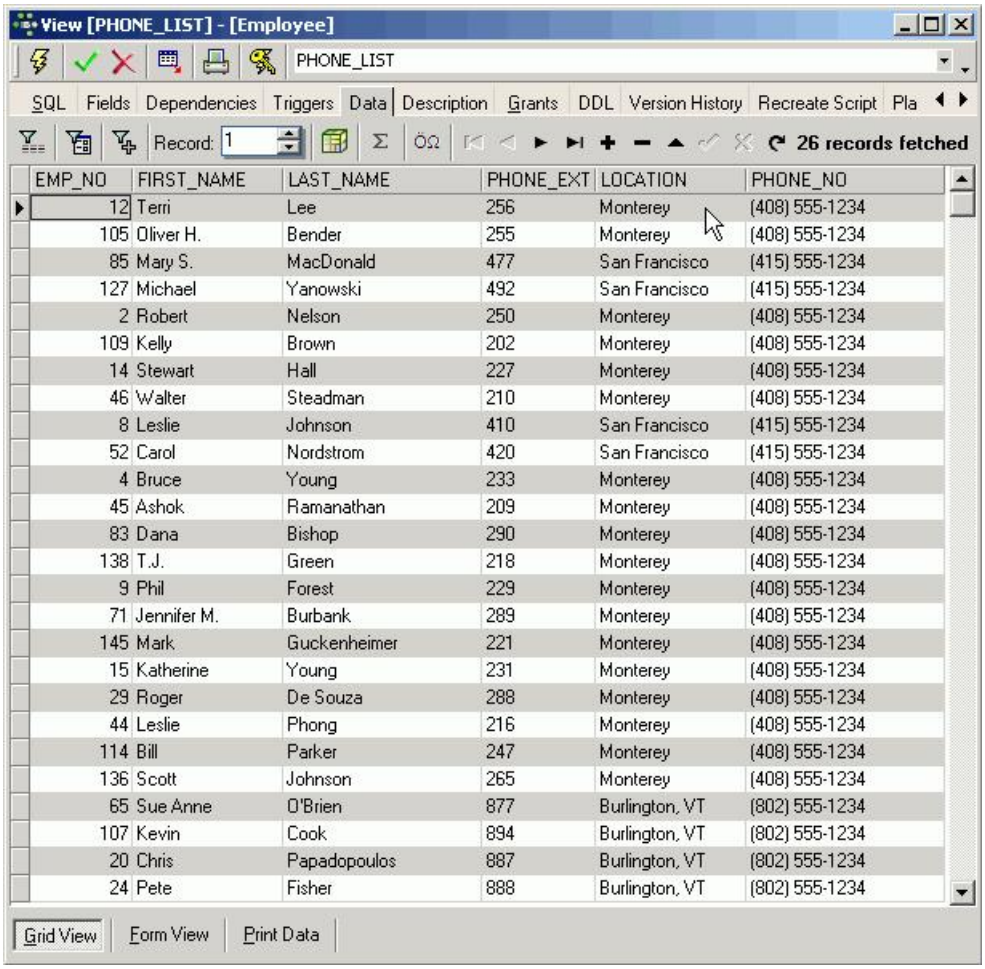
Please refer to Table Editor / Dependencies.

Triggers



Please refer to Table Editor / Triggers.

Data

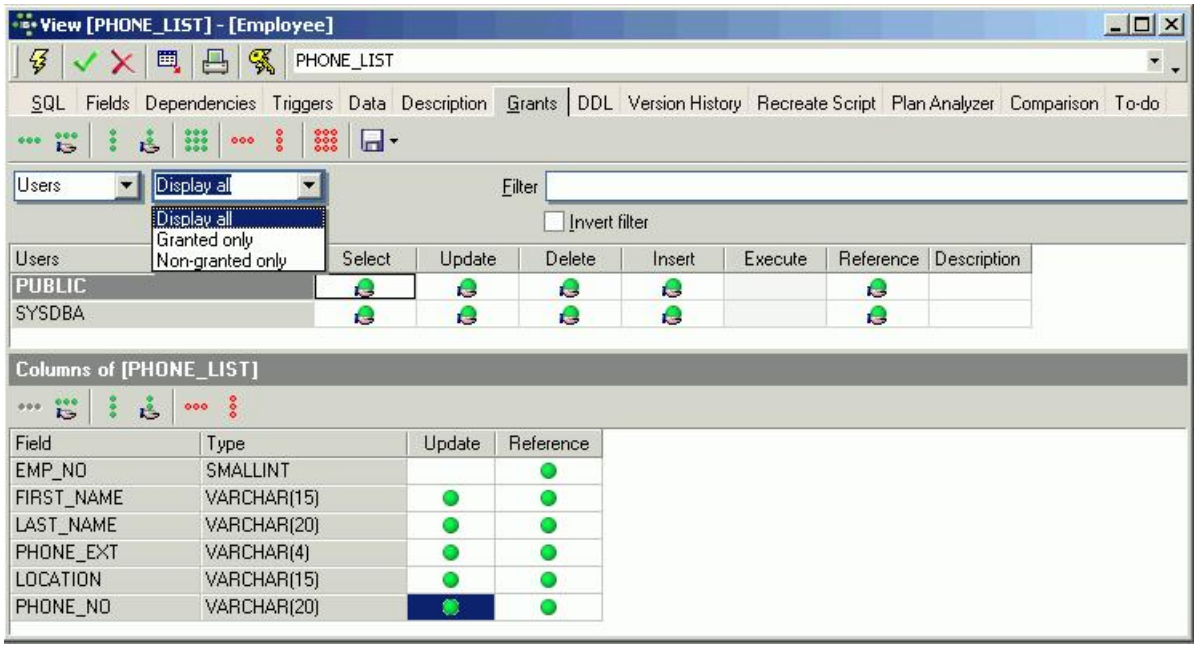


Please refer to Table Editor / Data. Please note that data may only be manipulated in this dialog if the view is defined as, and meets all conditions required by an updatable view.

Description

Please refer to Table Editor / Description.

Grants



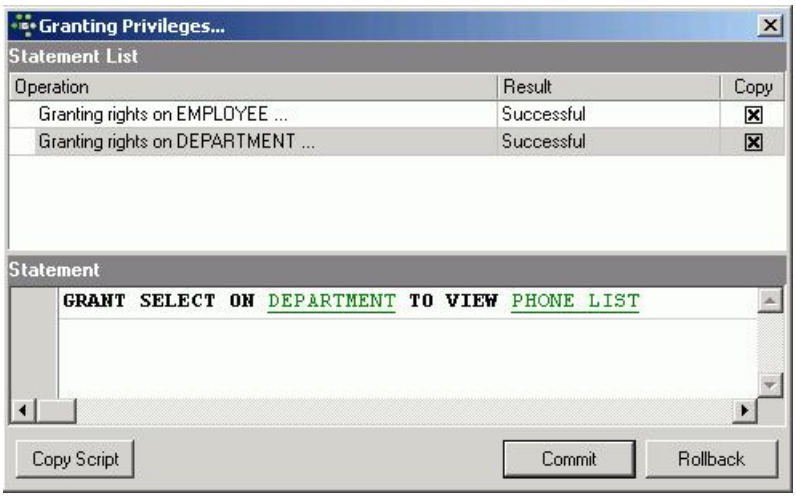
Please refer to Table Editor / Grants.

Autogrant Privileges

The Autogrant Privileges icon

can be found in the View Editor toolbar, Procedure Editor toolbar and Trigger Editor toolbar. Privileges can also be autogranted using the key combination [Ctrl + F8]. It allows all privileges to be automatically granted for views, procedures and triggers.

(This feature is unfortunately not included in the IBExpert Personal Edition.)

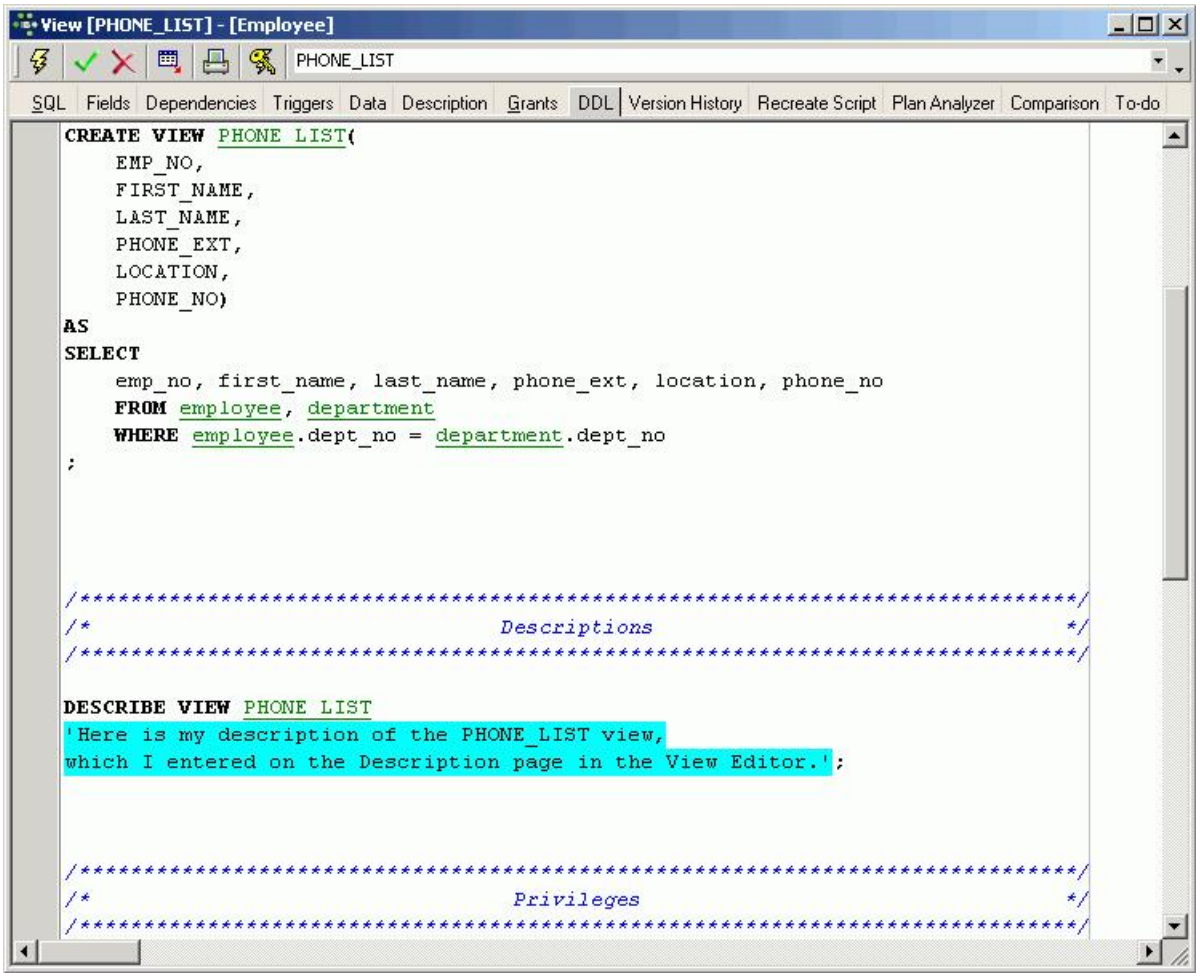


This assigns all rights for newly created objects for all users, and helps to prevent the frequent problem that developers often initially create multitudes of objects for their new database, and suddenly realize that they have not assigned any rights for these views, triggers or procedures.

For those preferring to limit the assignment of rights, please use the *Grants* page, offered in the majority of object editors, or the IBExpert Tools / Grant Manager.

Under the IBExpert Option menu item, Environment Options / Tools the default option, *Autogrant privileges when compiling procedures, triggers and views*, needs to be checked, for this function to work. Since IBExpert version 2005.02.12.1 it is also possible to specify here whether existing privileges should first be deleted, before new ones are granted.

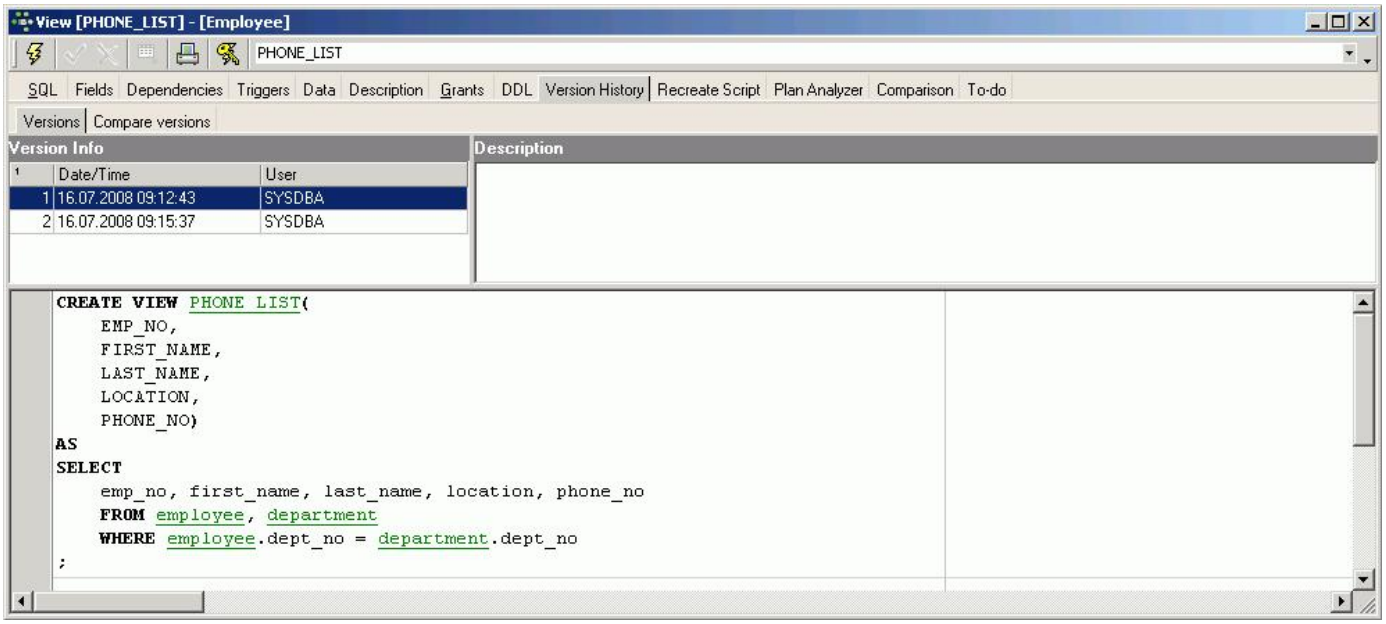
DDL



Please refer to Table Editor / DDL.

Version History

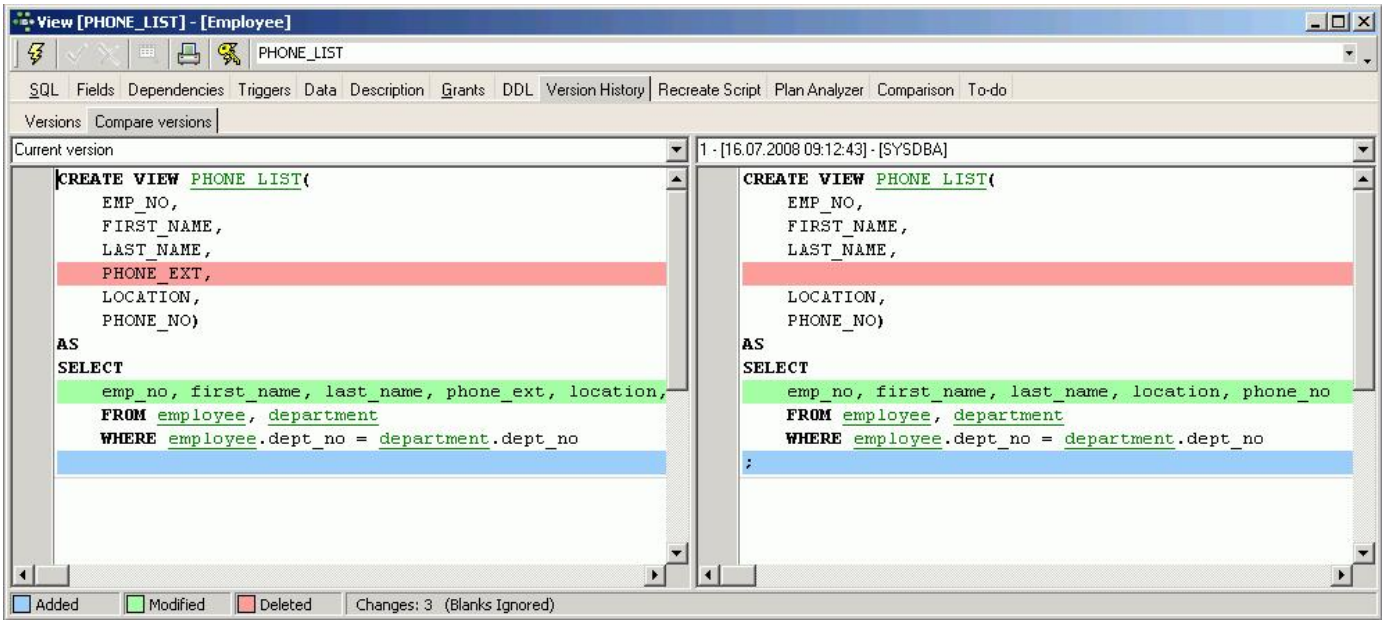
The *Version History* page offers a unique and automatic documentation. It is available in the View Editor, Procedure Editor and Trigger Editor. It displays different versions of the view, procedure or trigger (if existent), and lists the dates when changes were made, along with the person(s) responsible.



The first time the *Version History* is opened, IBExpert asks for confirmation, as it needs to create certain system tables for the version history logging. This only needs to be confirmed once. After this the *Version History* appears immediately in all relevant editors, and all object changes are automatically stored.

Versions listed in the *Version Info* panel can be marked, and deleted using the right mouse click menu (key combinations: *Delete version* [Del]; *Remove duplicates* [Shift + Ctrl + Del]).

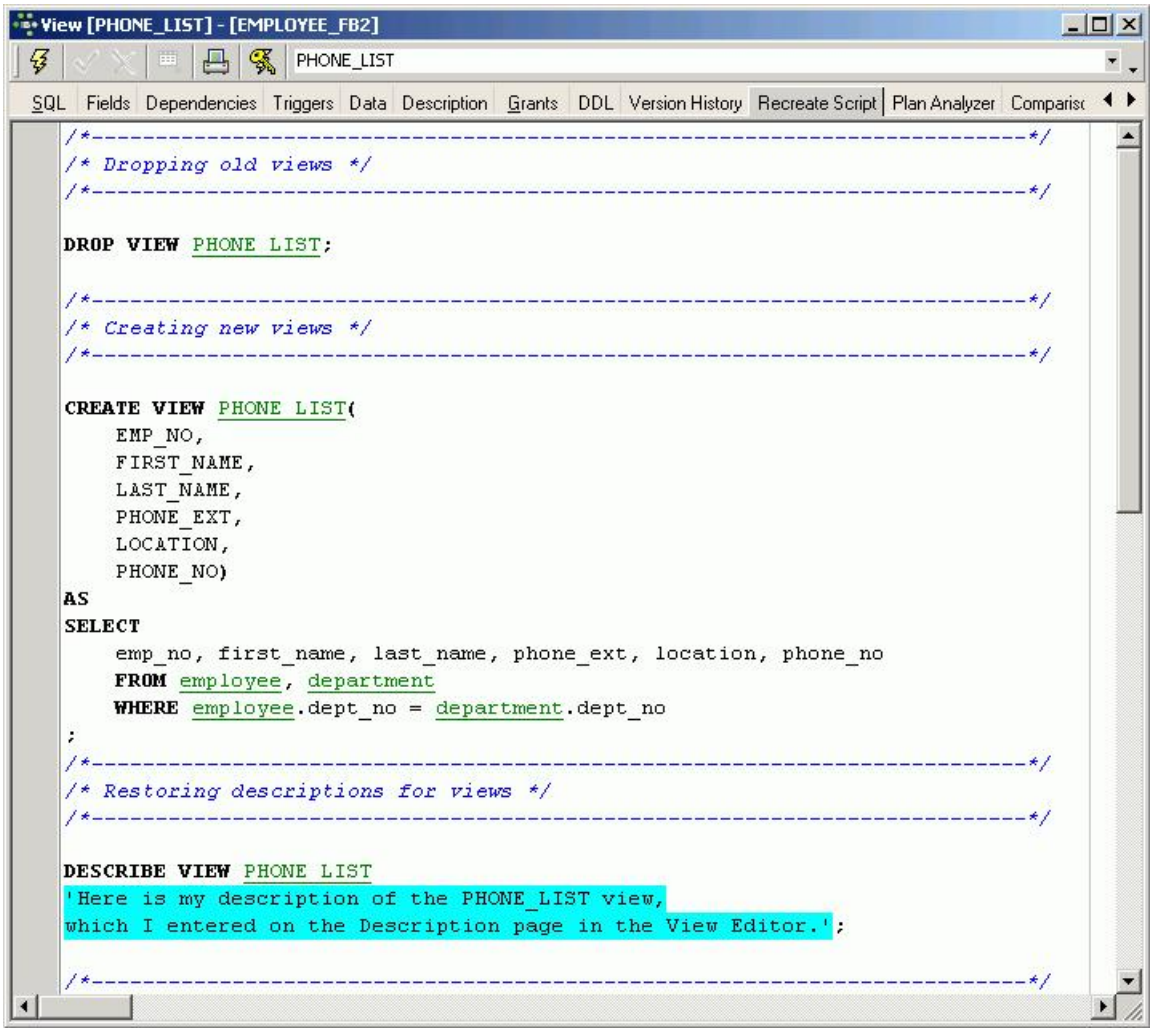
The SQL scripts of the different versions can even be compared, under the *Compare Versions* tab.



The pull-down list at the top of the two script panels allows different versions to be selected, without having to switch back to the *Versions* page. Alterations are highlighted by colored bars, marking the line where an alteration has been made. The color code key can be viewed in the dialog's status bar, along with a note of the number of changes made between the two versions.

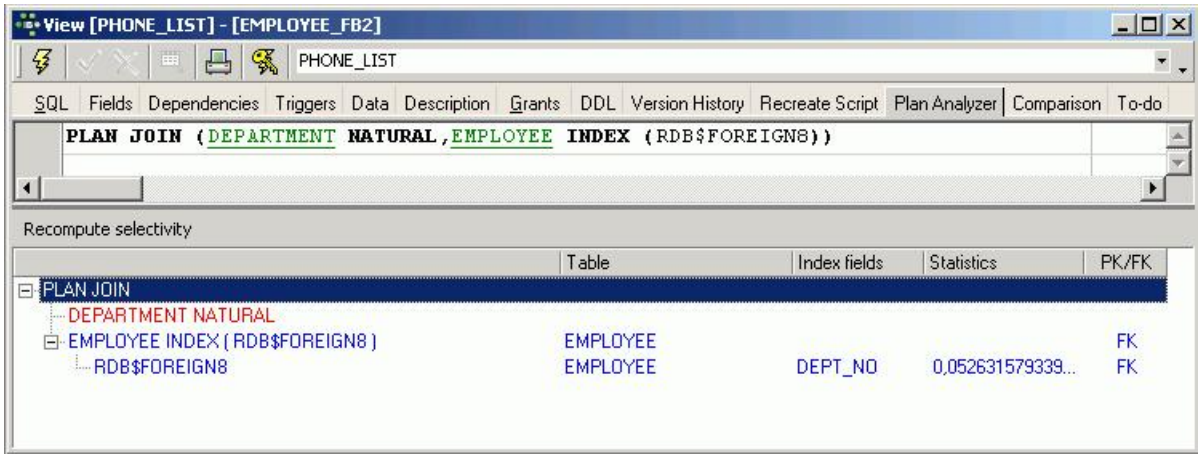
Recreate Script

The *Recreate Script* page displays the full SQL script for the view, beginning with the `DROP VIEW` command, and then recreating the current view. This is useful should errors arise in a view where it is almost impossible, due to the complexity of the view or the multitude of different versions, to detect the source.



The script can even be edited directly in this dialog, and the changes committed. The right-click menu is the same as that in the SQL Editor, allowing a number of further operations directly on the SQL script (please refer to SQL Editor Menu).

Plan Analyzer



Please refer to SQL Editor / Plan Analyzer. Please note that the performance information is not available here in the View Editor's *Plan Analyzer*.

Comparison

Please refer to Table Editor / Comparison.

To-Do

Please refer to Table Editor / To-Do.

Updatable views and read-only views

The simplest and quickest way to create an updatable view is to use the *Create View from Table* option in the IBExpert Table Editor, and create a trigger (checkbox options to create `BEFORE INSERT`, `BEFORE UPDATE` or `BEFORE DELETE`). Complete the trigger text in the lower code editor window (taking into consideration the notes below), and the updateable view is complete!

If the view is to be an updatable view, the optional parameter `WITH CHECK OPTIONS` needs to be used to control data input. If this parameter is used, only those values corresponding to the view's `SELECT` statement may be input. A view needs to meet all of the following conditions if it is to be used to update data in the base table:

1. The view is based on a single table or on another updatable view. Joined tables result in a read-only view. (The same is true if a subquery is used in the `SELECT` statement.)
2. Any columns in the base table that are not part of the view allow `NULL`s. This condition requires that the base table's primary key be included in the view.
3. The `SELECT` statement does not include a `DISTINCT` operator. This restriction might have the effect of removing duplicate rows, making it impossible for InterBase/Firebird to determine which row to update.
4. The `SELECT` statement does not include aggregate functions or the `GROUP BY` or `HAVING` operators.
5. The `SELECT` statement does not include stored procedures or user-defined functions.

In a normalized database, a view is usually updatable if it is based on a single table and if the primary key column or columns are included in the view definition.

However it is possible to input data into a view and then allocate the new data / data changes to several individual tables by using triggers.

Specifying a view with the `CHECK OPTION`

If a view is updatable, `INSERT`, `UPDATE`, or `DELETE` operations can be made on the view to insert new rows into the base table(s), or to modify or delete existing rows.

However, the update could potentially cause the modified row to no longer be a part of the view, and what happens if the view is used to insert a row that does not match the view definition?

To prevent updates or inserts that do not match the `WHERE` condition of the view, the `WITH CHECK OPTION` needs to be specified after the view's `SELECT` statement. This clause tells InterBase/Firebird to verify an `UPDATE` or `INSERT` statement against the `WHERE` condition. If the modified or inserted row does not match the view definition, the statement fails and InterBase/Firebird returns an error.

Alter view

A view can be altered in the View Editor, opened by double-clicking on the view name in the DB Explorer. Alternatively use the DB Explorer's right mouse-click menu item *Edit View* or key combination [Ctrl + O].

Alterations may be made directly in the SQL input page; fields, dependencies and triggers can be examined in their respective pages before field deletion.

When altering a view, IBExpert actually does nothing other than create a new view of the same name as the old one, replacing it after committing.

Drop view/delete view

When a view is dropped it is deleted for good. A view cannot be dropped if it is used elsewhere in the database's metadata. For example, if the view to be dropped is included in the definition of another view, a stored procedure or any `CHECK` constraint, the dependent object must first be dropped before the view can be dropped. Any existent dependencies can be viewed on the View Editor / Dependencies page. Most database objects can be dropped here directly on the *Dependencies* page or using the IBExpert Dependencies Viewer (found in the IBExpert Tools menu) by using the right-click menu on the selected object, and choosing the menu item *Drop Object* or [Ctrl + Del].

To drop a view, use the DB Explorer right mouse button menu item Drop View... (or [Ctrl + Del]).

IBExpert asks for confirmation:

before finally dropping the view. Once dropped, it cannot be retrieved.

Alternatively the `DROP VIEW` statement can be used in IBExpert's SQL Editor. It has the following syntax:

```
DROP VIEW <view_name>;
```

For example, to drop the `PHONE_LIST` view in the sample `EMPLOYEE` database, the following statement should be issued:

```
DROP VIEW PHONE_LIST;
```

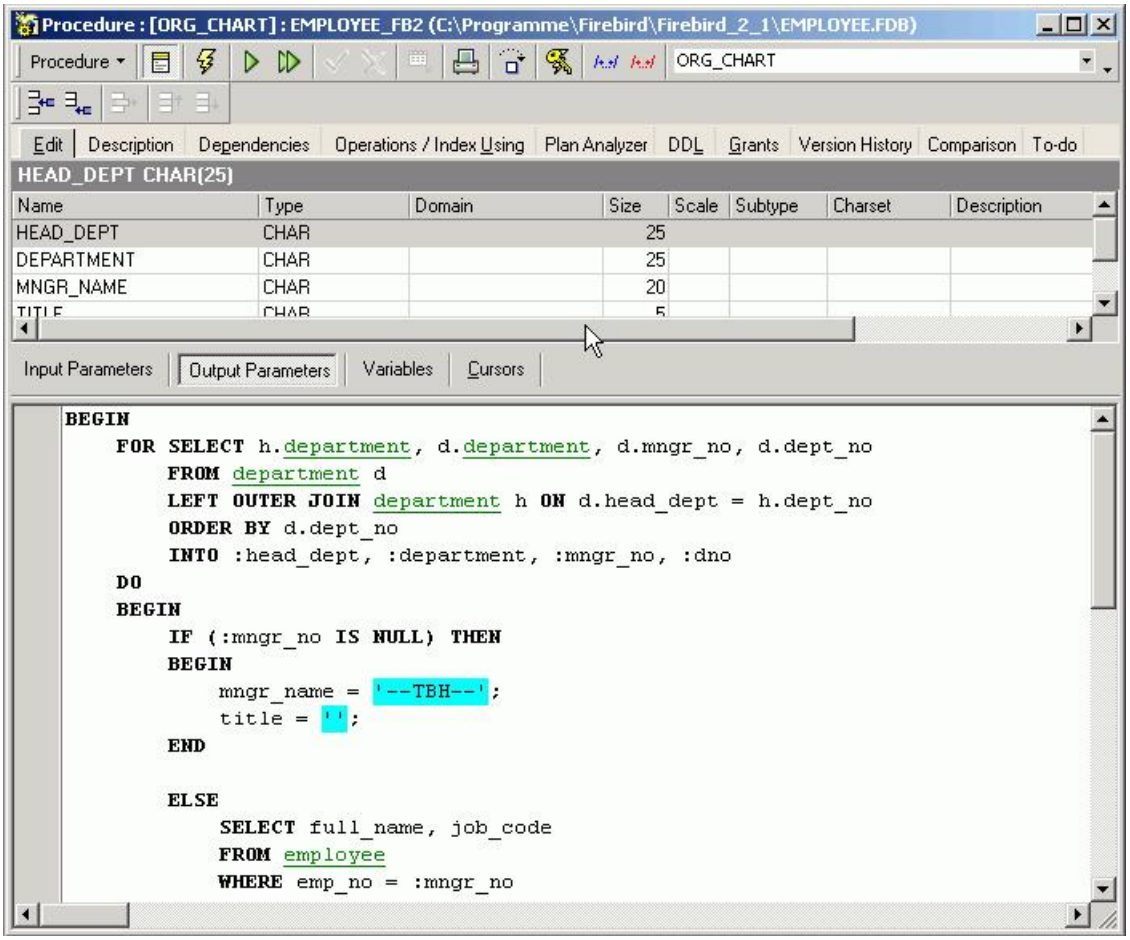
Please note that a view can only be dropped by its creator or the SYSDBA.

See also:
Create a trigger for a view
Create view or procedure from SELECT

Stored Procedure

A stored procedure is a series of commands (also known as routines) stored as a self-contained program in the database as part of the database's metadata. They are pre-compiled, so they don't need to be sent over the network and parsed every time, they are just executed. They can be started by the EXECUTE PROCEDURE command with specification of the procedure name and a list of parameters. Procedures can take parameteres and - like SELECTS - give back their data in the form of a table.

It is similar to a trigger, but is not automatically executed or bound to a specific table.



It is written in Firebird/InterBase procedure and trigger language, also known as PSQL. It can perform special processing on the metadata and data within the database. Program execution occurs on the server.

Currently the maximum size of a stored procedure or trigger in InterBase and Firebird is 48 KB of BLR (the size of the byte code language compiled from stored procedure or trigger language and not the source code itself, which may include comments). However, as this comprises well over 1,000 lines of code, it is wiser to split any procedures of this size into smaller ones anyway, as this will improve not just the readibility and ease of maintenance but also, more often than not, the efficiency.

Each stored procedure is a stand-alone module of code that can be executed interactively or as part of a SELECT statement, from another stored procedure or from another application environment.

They can be invoked directly from applications, or can be substituted for a table or view in a SELECT statement; they can receive input parameters and return values to applications.

With the Client/Server database concept, it is important that the database is not just used to store data, but is actively involved in the data query and data manipulation processes. As the database must also be able to guarantee data integrity, it is important that the database can also handle more complex operations than just simple comparisons. InterBase/Firebird uses stored procedures as the programming environment for integrating active processes in the database.

The stored procedure language is a language created to run in a database. For this reason its range is limited to database operations and necessary functions.

Stored procedures provide SQL enhancements that support variables, comments, declarative statements, conditional testing and looping as programming elements. They have full access to SQL DML statements allowing a multitude of command types; they cannot however execute DDL statements, i.e. a stored procedure cannot create a table.

Stored procedures offer the following advantages when implementing applications:

1. Reduction of network traffic by off-loading application processes from the client to the server. This is particularly important for remote users using slower modem connections. And for this reason of course, they are fast.

2. Splitting up of complex tasks into smaller and more logical modules. Stored procedures can be invoked by each other. Stored procedures allow a library of standardized database routines to be constructed, that can be called in different ways.

3. They're reusable. Rather than recreate a statement on the client each time it's needed, it's better to store it in the database. They can be shared by numerous applications using a single database. Alterations to the underlying data definitions only need to be implemented in the stored procedure and not in the individual applications themselves. Readability is enhanced, and redundancy, maintenance, and documentation are greatly reduced.

4. Full access to SQL and the database's metadata. This allows certain environments to perform extended operations on the database that might not be possible from another application language. The language even offers functions that are not available in SQL, e.g. IF...WHEN...ELSE, DECLARE VARIABLE, SUSPEND, etc.

5. Enhanced security: if database operations such as INSERT, ALTER or DROP can only be performed on a table by stored procedures, the user has no privileges to access the table directly. The only right the user has is to execute the stored procedure.

6. As stored procedures are part of InterBase or Firebird, it is irrelevant which front end is subsequently used, be it Delphi, PHP or other.

There are no disadvantages to using stored procedures. There are however, two limitations. Firstly, any variable information must be able to be passed to the stored procedure as parameters or the information must be placed in a table that the stored procedure can access. Secondly, the procedure and trigger language may be too limited for complex calculations. Stored procedures should be used under the following circumstances:

1. If an operation can be carried out completely on the server with no necessity to obtain information from the user while the operation is in process. When invoking a stored procedure these input parameters can be incorporated in the stored procedure.

2. If an operation requires a large quantity of data to be processed, whose transfer across the network to the client application would cost an enormous amount of time.

3. If the operation must be performed periodically or frequently.

4. If the operation is performed in the same manner by a number of different processes, or processes within the application, or by different applications.

The stored procedure must contain all statements necessary for the database connection, creation or alteration of the stored procedure, and finally the disconnection from the database.

All SQL scripts can be incorporated into a stored procedure and up to 10 SQLs in one procedure, as well as the additional functions already mentioned, making stored procedures considerably quicker and more flexible than SQL.

Stored procedures can often be used as an alternative to views (being more flexible and offering more control) as the ORDER BY instruction cannot be used in a view (the data sets are displayed as determined by the optimizer, which is not always intelligent!). In such a case, a stored procedure should be used.

Stored procedures are almost identical to triggers, the only exception being the way they are called. Triggers are called automatically when a change to a row in a table occurs. Most of what is said about stored procedures applies to triggers as well.

Executing stored procedures

InterBase/Firebird stored procedures are divided into two groups with respect to how they are called. Select procedures return result values through output parameters, because they can be used in place of a table name in an SQL SELECT statement. Execute or non-select procedures perform an action and do not return values. To ba able to call a procedure, the user must have EXECUTE rights (see Grant Manager). In IBExpert the template already includes this statement for you (refer to the illustration in the *SET TERM* chapter below).

The simplest way to execute a stored procedure is to use the EXECUTE PROCEDURE statement. This statement can be used in one of the following ways:

1. From within another stored procedure.
2. From within a trigger.
3. From an application.

When a procedure is executed from within an InterBase/Firebird application, such as another procedure or a trigger, it has the following syntax:

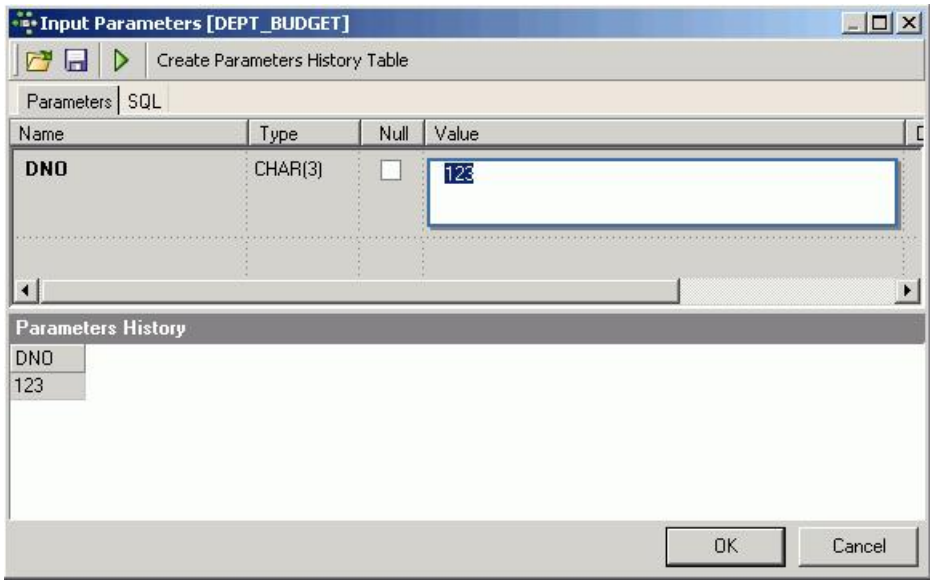
```
EXECUTE PROCEDURE
<procedure_name>
<input_parameter_list>
RETURNING_VALUES
<parameter_list>
```

If the procedure requires input variables, or if it is to return output variables, the relevant parameters need to be specified. In each case, `<parameter_list>` is a list of parameters, separated by commas (see stored procedure parameters for further information).

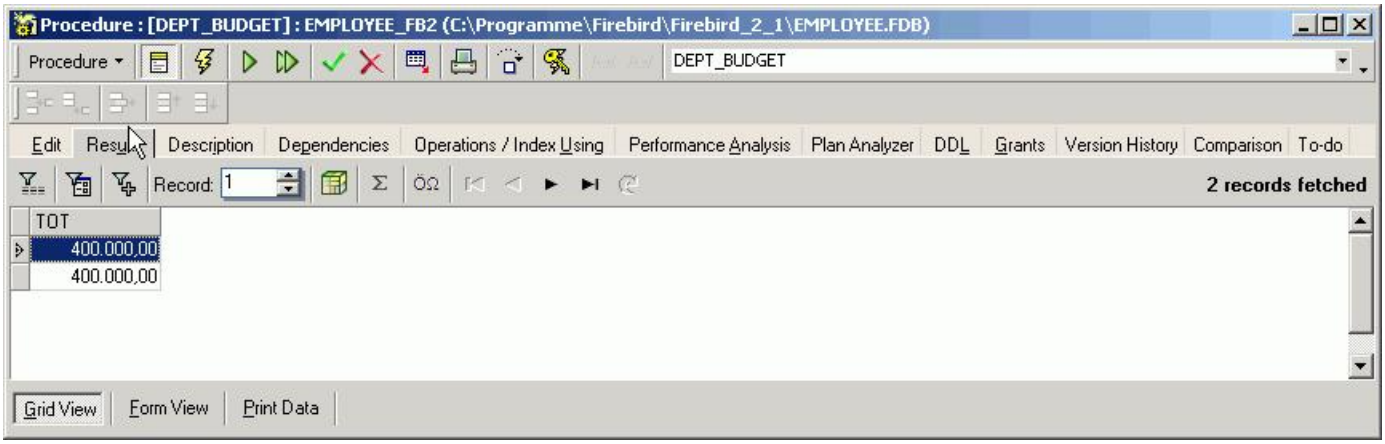
Each time a stored procedure calls another procedure, the call is said to be nested because it occurs in the context of a previous and still active call to the first procedure.

Stored procedures can be nested up to 1,000 levels deep. This limitation helps to prevent infinite loops that can occur when a recursive procedure provides no absolute terminating condition. Nested procedure calls may be restricted to fewer than 1,000 levels by memory and stack limitations of the server.

When using IBExpert's Procedure Editor to execute a procedure, IBExpert tells you whether input parameters need to be entered:



before displaying the return values (= output or results) on the *Results* page:



Select procedures

It is possible to use a stored procedure in place of the table reference in a SELECT statement. This type of procedure is known as a select procedure.

When a stored procedure is used in place of a table, the procedure should return multiple columns or rows, i.e. it assigns values to output parameters and uses SUSPEND to return these values. This allows the SELECT statement to filter the results further by different criteria.

SUSPEND is used to suspend execution of the procedure and return the contents of the output variables back to the calling statement. If the stored procedure returns multiple rows, the SUSPEND statement needs to be used inside a FOR SELECT ... DO loop to return the rows one at a time.

Non-select procedures

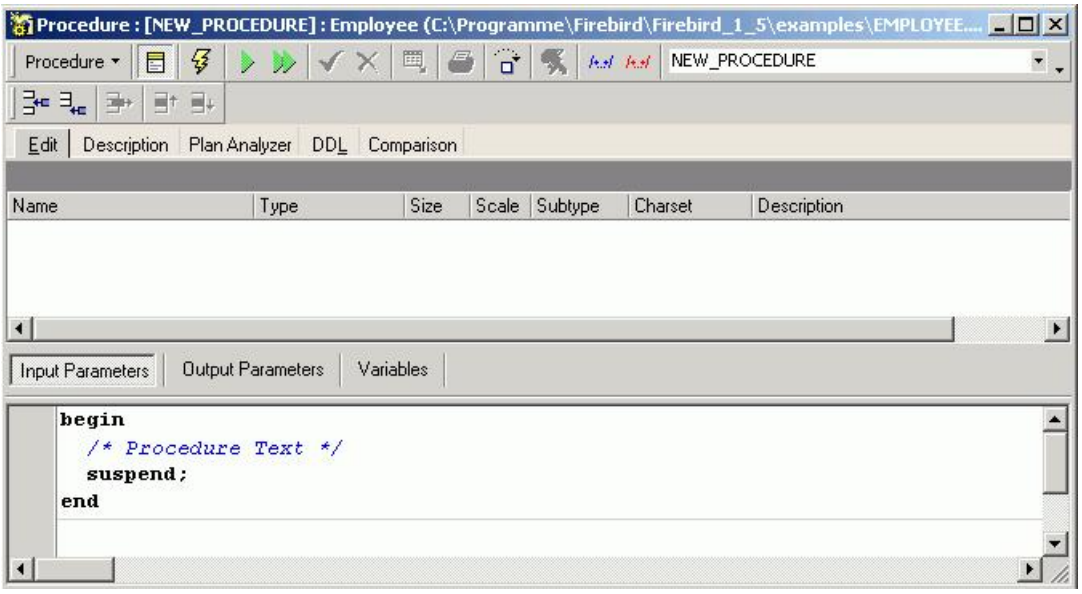
Execute or non-select procedures perform an action and do not return any results.

New procedure

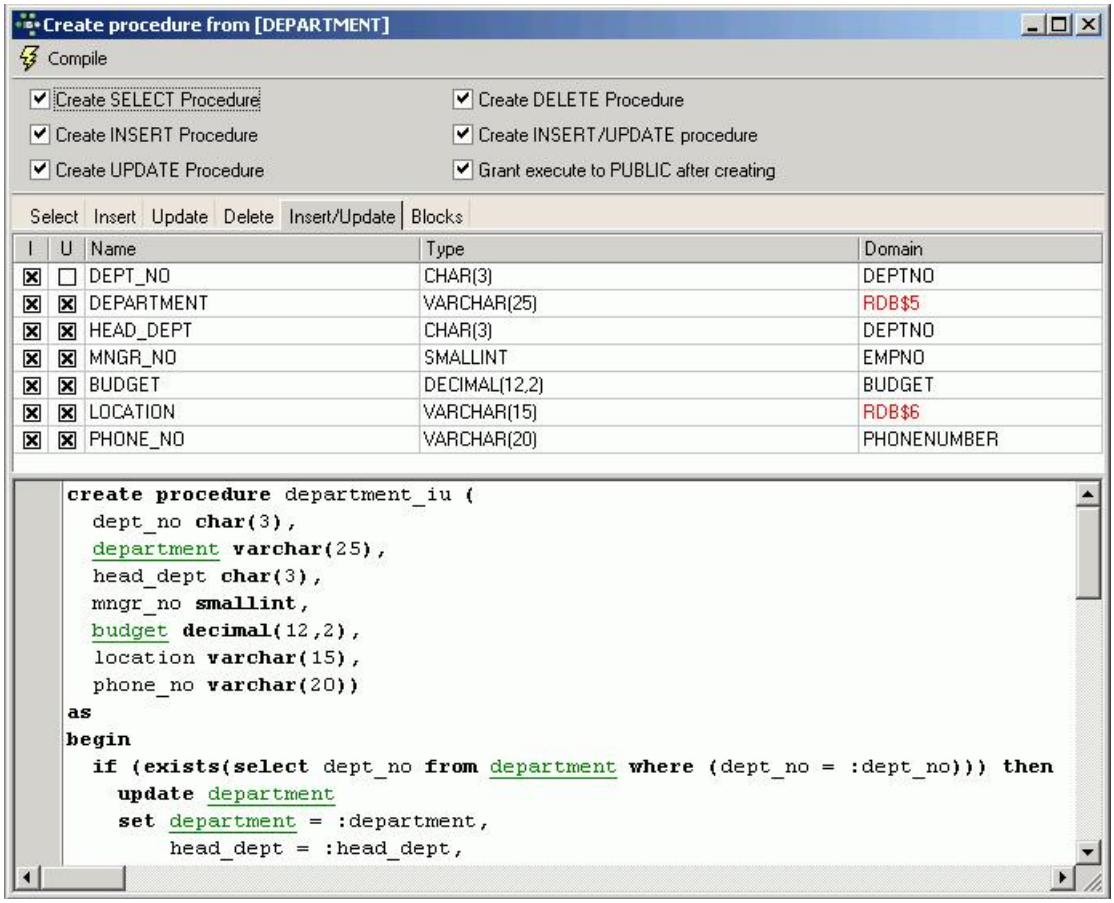
There are numerous ways to approach creating a new stored procedure:

1. Using the IBExpert menu item Database / New Procedure or using the *New Procedure* icon on the New Database Object toolbar to start the Procedure Editor.

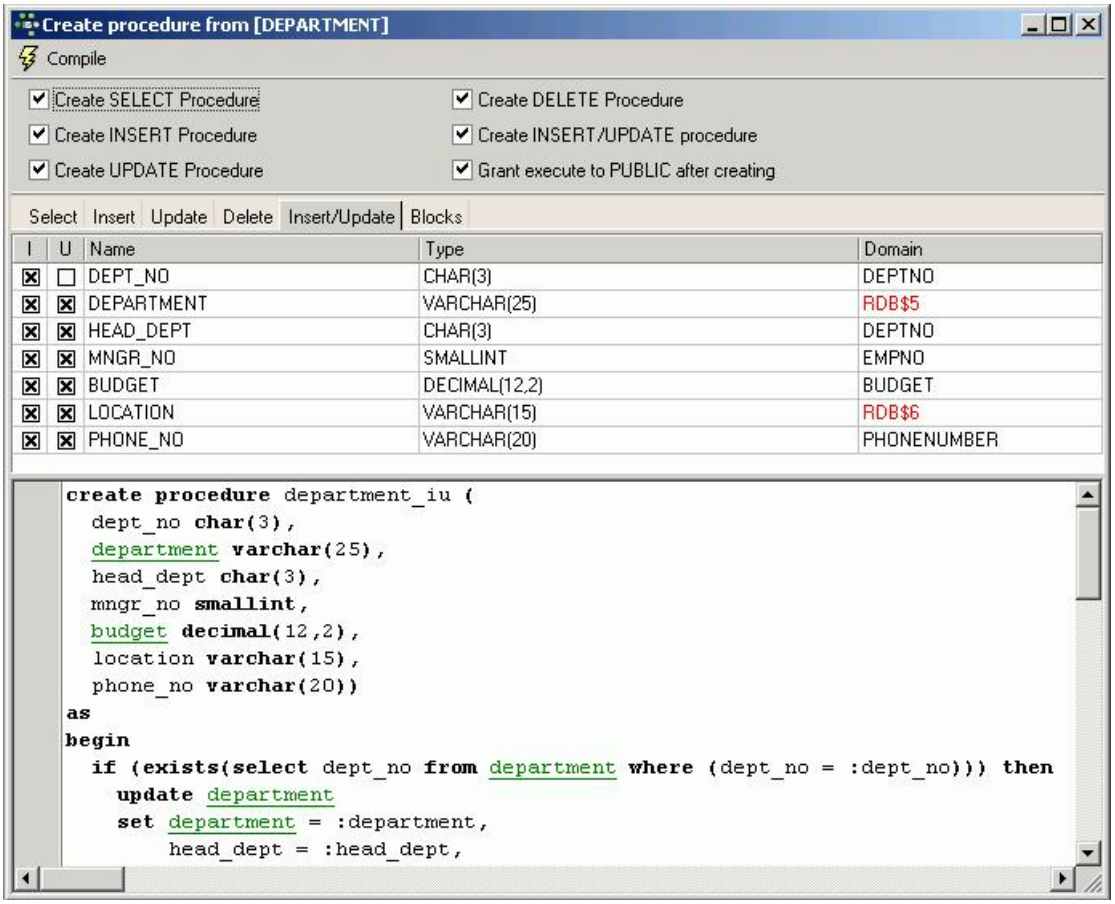
2. From the DB Explorer by right-clicking on the highlighted procedure branch of the relevant connected database (or key combination [Ctrl + N]) which also starts the Procedure Editor.



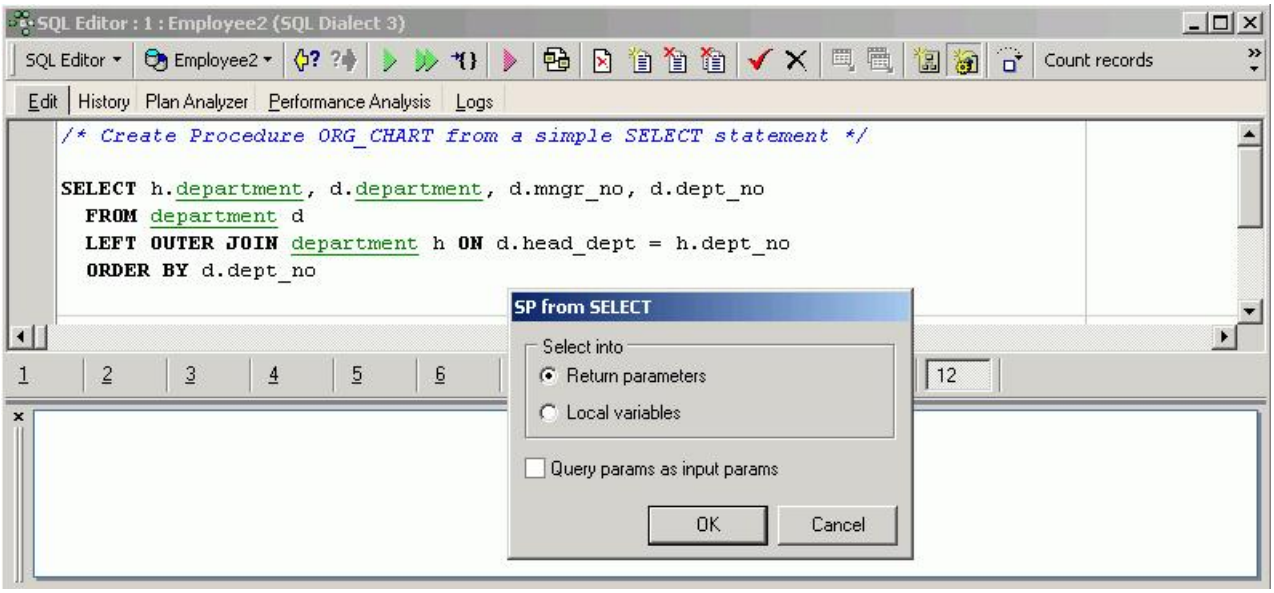
3. A stored procedure can also be created directly from a selected table in the DB Explorer, using the right-click pop-up menu item *Create SIUD procedures*.



4. Or created directly from the Field Editor.



5. Or created in the IBExpert SQL Editor, and then saved as a stored procedure. When an SQL script has been successfully committed, and the results are as wished, the script can be integrated into a stored procedure using the *Stored Procedure* button. The stored procedure script appears, and simply needs to be named and completed.



The `CREATE PROCEDURE` statement has the following syntax:

```
CREATE PROCEDURE <Procedure_Name>
<Input_Parameter_List>
RETURNS
<Return_Parameter_List>
AS
<Local_Variable_Declarations>
BEGIN
<Procedure_Body>
END
```

The `CREATE` and `RETURNS` statements (if there is a return statement) comprise the stored procedure's header. Everything following the `AS` keyword is the procedure's body. There can also be statements between the `AS` and `BEGIN` keywords that are also considered part of the body. These statements declare local variables for the stored procedure, and are detailed under Stored Procedure Language.

Since IBExpert version 2005.03.12 there is added support for following Firebird 2 features:

- DECLARE <cursor_name> CURSOR FOR ...
- OPEN <cursor_name>
- FETCH <cursor_name> INTO ...
- CLOSE <cursor_name>
- LEAVE <label>
- NEXT VALUE FOR <generator>

The possibility to create SUID procedures was implemented in IBExpert version 2007.02.22. This is a new mechanism of composing texts of SIUD procedures based on IBEBlock.

Further information explaining the necessary components can be found under Procedure Editor, started using the first two menu options (i.e. IBExpert Database menu and DB Explorer right mouse button menu).

The Procedure Editor has its own toolbar (see Procedure Editor toolbar). To the right of the toolbar, the new procedure name can be specified. The procedure name follows the naming convention for any InterBase/Firebird object and must be unique. The *Lazy Mode* icon can be used to switch the lazy mode on and off as wished:

The New Procedure Editor has five pages:

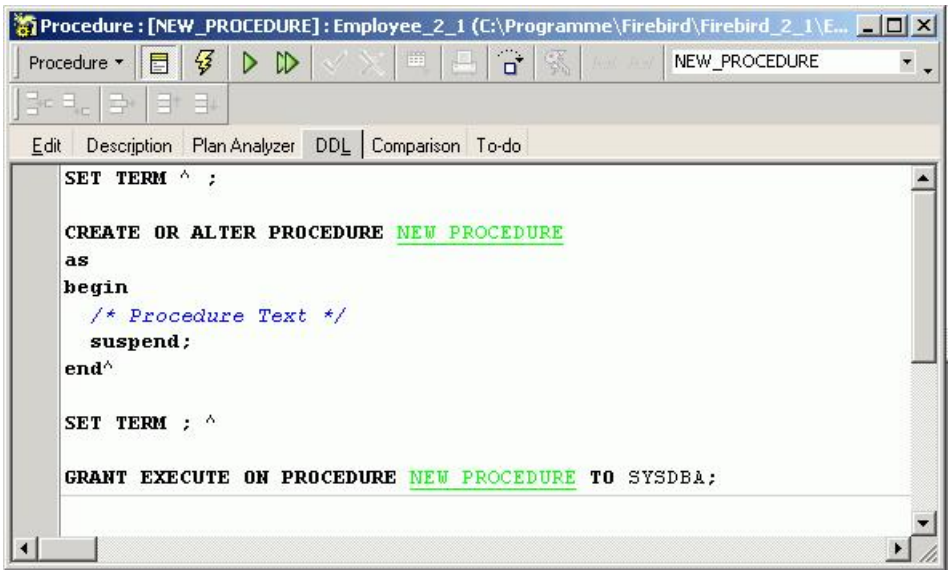
1. Edit
2. Description
3. Plan Analyzer
4. DDL
5. Comparison

described under Procedure Editor. A new procedure is created on the Procedure Editor / Edit page.

SET TERM

Every command in a Firebird/InterBase script must be terminated by a semicolon, including the procedure itself. To distinguish the semicolons in the procedure from the terminating semicolon, another temporary terminator is needed for the end of the procedure. `SET TERM` replaces the terminator semicolon with a user-defined character. After the procedure itself is terminated by this new terminator, the terminator symbol is set back to the semicolon.

When using the IBExpert Procedure Editor, the procedure templates already include this code, so you don't have to worry about it. If you open the New Procedure Editor and take a peek at the DDL page, you will see how much code has already be generated by IBExpert, although you haven't even started to define your procedure:



Even `SUSPEND@@` and the GRANT EXECUTE@@ statement have been included.

For those who wish to view the syntax and an example of how to use this when coding by hand, please refer to `SET TERM` terminator.

Stored procedure parameters (input and output/returns)

Input parameters are a list of variables (=values) that are passed into the procedure from the client application. These variables can be used within the procedure to modify its behavior.

The return parameter (or output parameter) list represents values that the procedure can pass back to the client application, such as the result of a calculation. Each list is in the following format:

```
ParameterName1 ParameterType,
ParameterName2 ParameterType,
```

```
...
ParameterNameN ParameterType
```

`ParameterType` is any valid InterBase/Firebird datatype except blob, domain and arrays of datatypes.

Local variables / `DECLARE VARIABLE` statement

Local variables can be defined within the procedure body. Local variables of any InterBase/Firebird type can be declared within a stored procedure. As with any other structured programming environment, these variables only exist while the procedure is running, and their scope is local to the procedure. They are invisible outside the procedure and are destroyed when the procedure finishes. There are no global variables available with stored procedures and triggers. If values need to be shared by two or more procedures, they should either be passed as parameters or stored in a table.

Local variables are declared immediately after the AS clause, using the `DECLARE VARIABLE` statement. For example the variable `ANY_SALES` is declared in the `EMPLOYEE` database's `DELETE_EMPLOYEE` procedure:

```
DECLARE VARIABLE ANY_SALES INTEGER;
```

Each variable must be declared in its own `DECLARE VARIABLE` statement, as each statement can declare only one variable.

Procedure body

The procedure body consists of a compound statement, which can be any number of InterBase/Firebird procedure and trigger language statements. The procedure body starts with a `BEGIN` statement, followed by any local variable declarations, and ends with an `END` statement.

`BEGIN` and `END` must also be used to surround any block of statements that logically belong together, such as the statements within a loop.

`BEGIN` and `END` do not need terminating characters, except for the final `END` within the procedure.

Comment Procedure Body/Uncomment Procedure Body

In certain situations it may be necessary to disable certain commands or parts of SQL text. This can be easily done temporarily, without it being necessary to delete these commands.

Simply select the rows concerned in the SQL Editor, and select either the editor toolbar icons:

the right mouse button menu item *Comment Selected*, or key combination [Ctrl + Alt + .]. This alters command rows to comments. The commented text can be reinstated as SQL text by using the *Uncomment Procedure* icon (above), the right mouse button menu item *Uncomment Selected*, or [Ctrl+ Alt + ,].

Lazy Mode

Using lazy mode, the programmer does not have to worry about which input and output parameters need to be considered. It can be switched between lazy mode and classic mode using the

icon in the Procedure Editor and Trigger Editor.

The possibility to select domains as a datatype for input/output parameters and variables has been added in IBExpert version 2004.8.5.1. In this case IBExpert copies information from the domain definition to the native datatype of the parameter/variable. It is now also possible to drag 'n' drop a domain from the Database Explorer.

And since IBExpert version 2005.06.07 it is possible to specify `SEGMENT SIZE` for blob parameters and variables whilst working in lazy mode.

Stored Procedure Editor

The Procedure Editor can be started using the Database / New Procedure menu item; from the DB Explorer, using the right mouse-click menu or double-clicking on an existing procedure.

Please refer to New Procedure when creating a stored procedure for the first time.

The Procedure Editor has its own toolbar (see Procedure Editor Toolbar) and offers the following options:

1. Edit
2. Results
3. Description
4. Dependencies
5. Operations/Index Using
6. Performance Analysis
7. Plan Analyzer
8. DDL

9. Grants
10. Version History
11. Comparison
12. To-Do

At the time of writing, the maximum size of a stored procedure is limited in InterBase and Firebird to 64K.

Edit

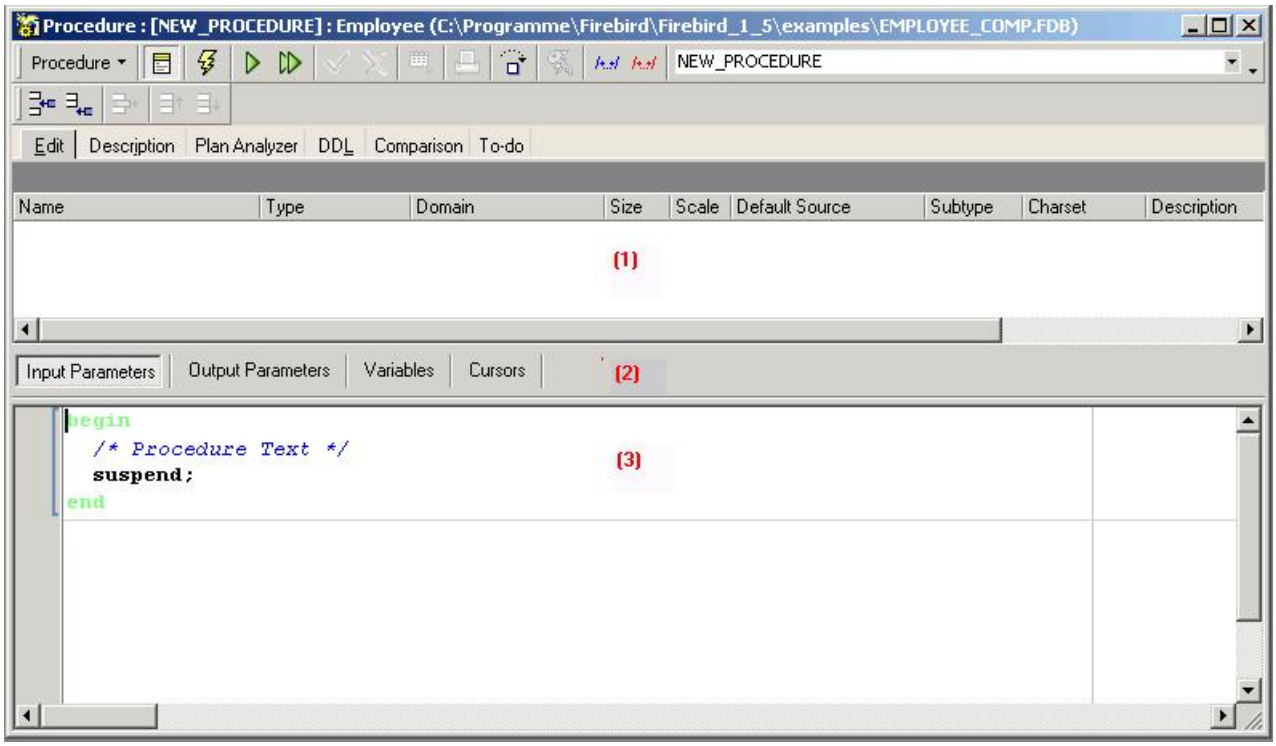
The `CREATE PROCEDURE` statement has the following syntax:

```
CREATE PROCEDURE <Procedure_Name>
<Input_Parameter_List>
RETURNS
<Return_Parameter_List>
AS
<Local_Variable_Declarations>
BEGIN
<Procedure_Body>
END
```

A stored procedure comprises the following components:

1. input parameters
2. output parameters (returns)
3. variables
4. procedure body
5. comments (optional)

If the lazy mode is switched off, the Edit dialog offers a single SQL input area, with the procedure syntax already displayed. If the lazy mode is switched on, the *Edit* dialog consists of three areas:



(1) The field grid, where new parameters can be specified.

(2) In the middle are three buttons specifying the parameter type, i.e. input parameters, output parameters and variables. It is possible to drag 'n' drop parameters/variables from the field grid onto the corresponding button to move them. For example, click the *Output Parameters* button, drag a named variable from the field grid onto the *Variable* button. Click the *Variable* button to view the new variable in the field grid.

(3) Below this is the SQL panel for direct code input. Again the procedure syntax is already displayed to help the user.

As with all Editors, it is possible to format the code text, such as:

- *Comment* or *Uncomment code* using the right-click context-sensitive menu
- indent a marked block of code with [Ctrl + Shift + I] and move back with [Ctrl + Shift + U]

Please refer to Localizing Form for further keyboard shortcuts.

For those who do not wish to use the basic syntax template, or wish to add certain statements themselves to create their own standard, this can be done using the IBExpert menu item Options / General Templates, and clicking on either the *Standard Mode* or *Lazy Mode* under *New Procedure*.

Since IBExpert version 2005.04.24 the Debugger also supports the new Firebird 2.0 feature: `SELECT ... FROM (SELECT ...)` and since IBExpert version 2005.06.07 the new Firebird 2.0 feature `IS DISTINCT FROM`.

Since IBExpert version 2005.12.04 the Code Completion list now displays cursor names when one is declared within procedure or trigger (Firebird 2).

As with all SQL input windows, the SQL Editor Menu can be called using the right mouse button.

The basic parameters of the stored procedure are set here as SQL text for creating the procedure. A parameter can have any InterBase/Firebird datatype except blob or array. The input parameters are set in brackets after the procedure name, the output parameters are set in brackets after the `RETURNS` statement, and the procedure body written in InterBase procedure and trigger language, bracketed by `BEGIN` and `END` statements.

New parameters can be quickly and easily specified, by clicking the respective button (i.e. *input*, *output* or *variables*), and inserting field information using the respective icon or right-click menu, in the same manner as creating a new table.

Local variables of any InterBase/Firebird type can be declared within a stored procedure (please refer to local variables), after the AS keyword and before the `BEGIN` (which marks the begin of the procedure body).

Alternatively, the required information can be entered directly in the editor's input panel and field names can be simply dragged from the DB Explorer or SQL Assistant into the procedure script. The code insight can be used to save time wasted searching for correct names, and to prevent any possible spelling errors. A right mouse-click within this area produces the SQL Editor menu.

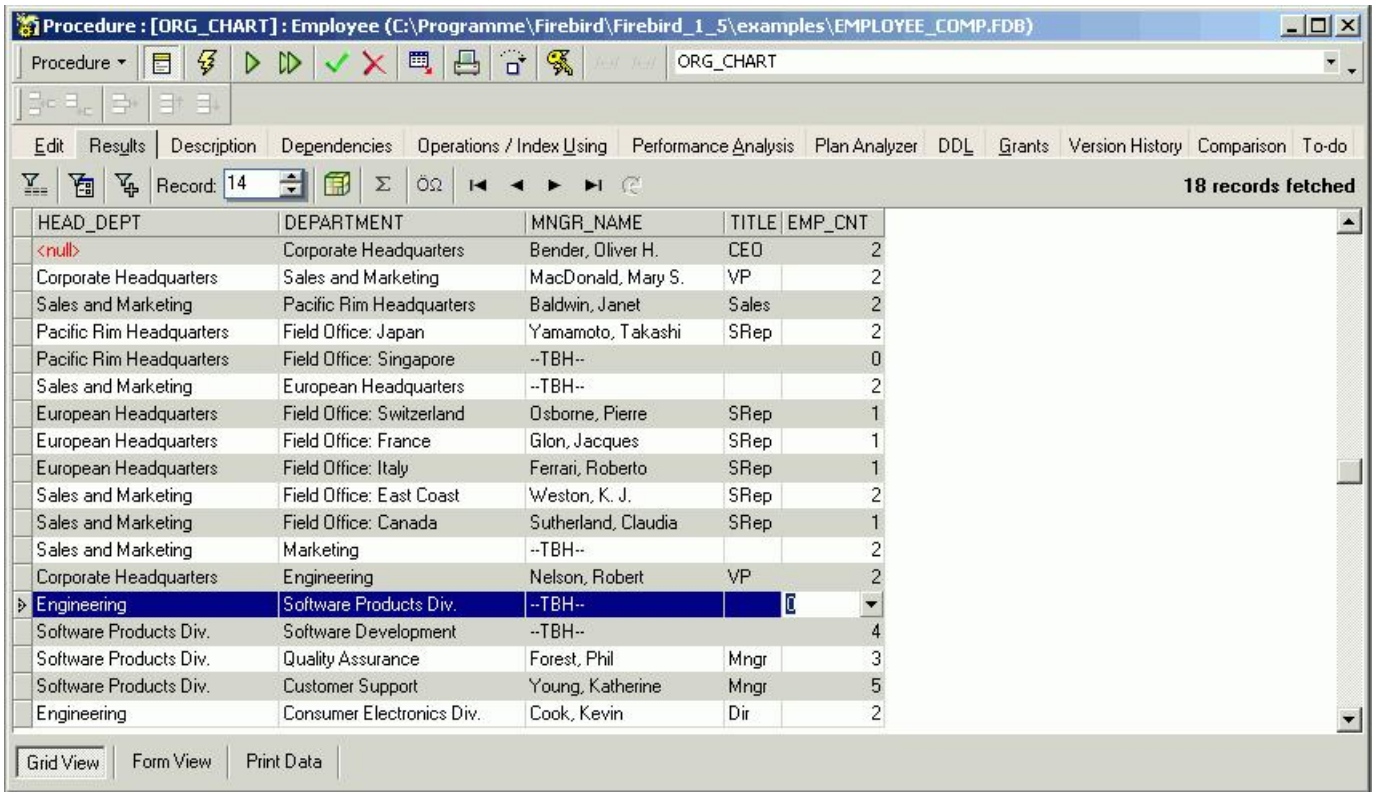
The input parameters are set with their types in brackets after the procedure name. By checking the *Code Parameter* option under Options / Editor Options / Code Insight, a list of the necessary parameters automatically appears. Output parameters are specified in the same way after `RETURNS`. The operations to be performed by the procedure are described after the `BEGIN` statement. Please refer to Stored Procedure and Trigger Language for further details.

After inputting the required information, the stored procedure can be executed using [F9] or the relevant icon. The statement window appears, where the resulting SQL statement can be viewed before committing. If necessary the code can subsequently be debugged using the debugging icon or [Shift + Ctrl + D] . (Please refer to Debug Procedure for more details.)

Don't forget to finally compile the new procedure using the respective toolbar icon or [F9], and, if desired, autogrant privileges, again using the respective toolbar icon or key combination [Ctrl + F8].

Results

The Results page appears following execution of the procedure, and displays all data sets fetched:



HEAD_DEPT	DEPARTMENT	MNGR_NAME	TITLE	EMP_CNT
<null>	Corporate Headquarters	Bender, Oliver H.	CEO	2
Corporate Headquarters	Sales and Marketing	MacDonald, Mary S.	VP	2
Sales and Marketing	Pacific Rim Headquarters	Baldwin, Janet	Sales	2
Pacific Rim Headquarters	Field Office: Japan	Yamamoto, Takashi	SRep	2
Pacific Rim Headquarters	Field Office: Singapore	--TBH--		0
Sales and Marketing	European Headquarters	--TBH--		2
European Headquarters	Field Office: Switzerland	Osborne, Pierre	SRep	1
European Headquarters	Field Office: France	Glon, Jacques	SRep	1
European Headquarters	Field Office: Italy	Ferrari, Roberto	SRep	1
Sales and Marketing	Field Office: East Coast	Weston, K. J.	SRep	2
Sales and Marketing	Field Office: Canada	Sutherland, Claudia	SRep	1
Sales and Marketing	Marketing	--TBH--		2
Corporate Headquarters	Engineering	Nelson, Robert	VP	2
Engineering	Software Products Div.	--TBH--		0
Software Products Div.	Software Development	--TBH--		4
Software Products Div.	Quality Assurance	Forest, Phil	Mngr	3
Software Products Div.	Customer Support	Young, Katherine	Mngr	5
Engineering	Consumer Electronics Div.	Cook, Kevin	Dir	2

Please refer to SQL Editor / Results for details.

Description

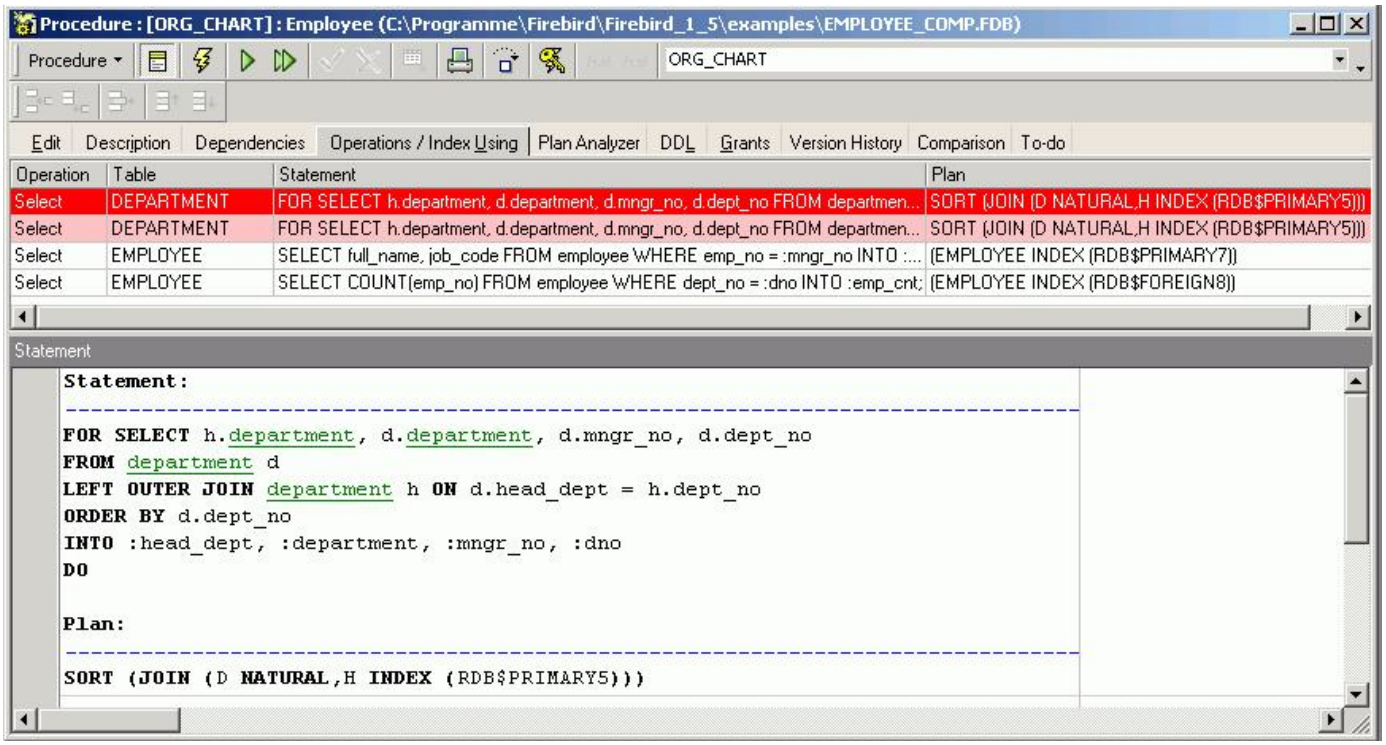
Please refer to Table Editor / Description.

Dependencies

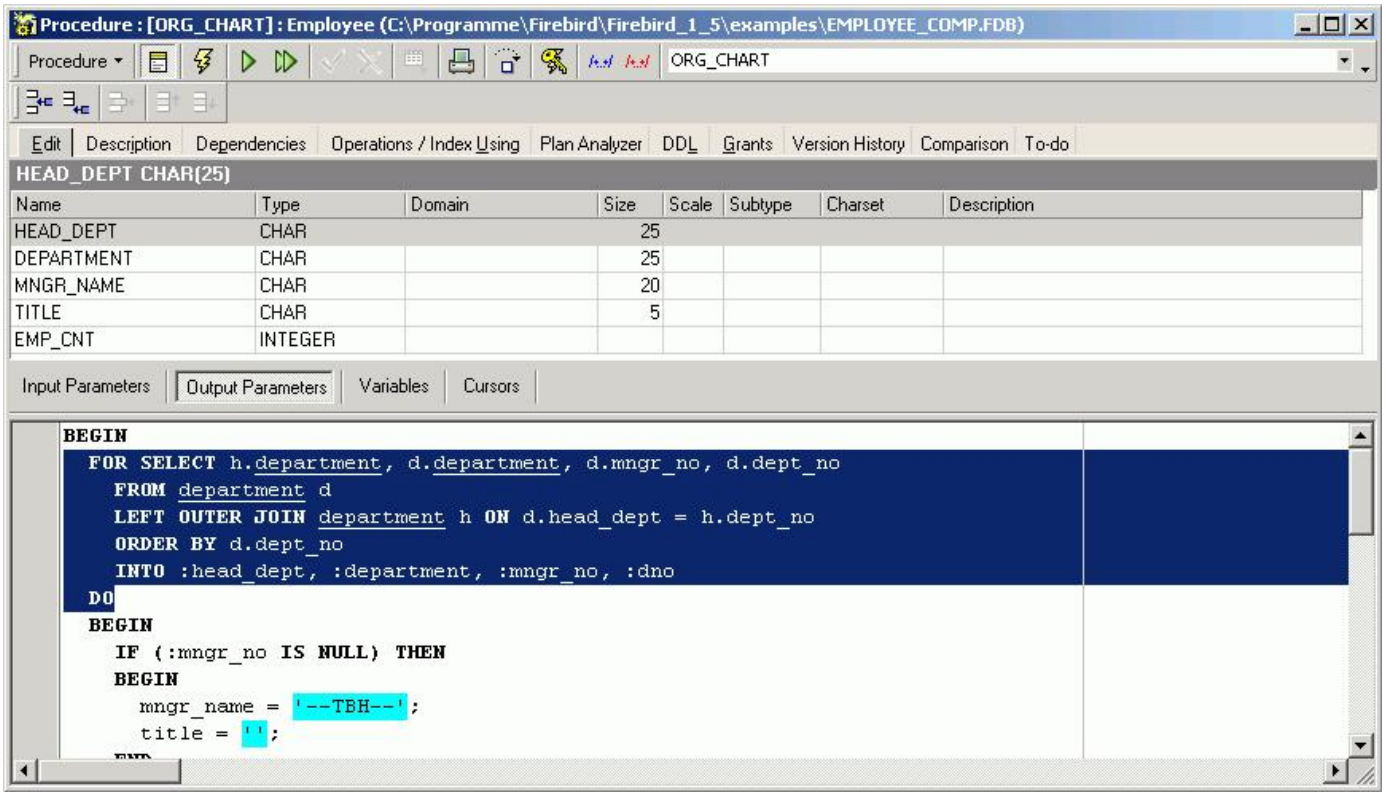
See Table Editor / Dependencies.

Operations/Index Using

This page dissects the procedure into single operations, and examines them to see whether they use a plan (i.e.) or not. The ORG_CHART procedure in the sample EMPLOYEE database displays red-marked entries, which indicates a plan NATURAL (i.e. no indices are used). When an operation is selected, the statement for this operation is displayed in the lower window:



By double-clicking on a selected operation, the SQL panel appears, highlighting the SQL statements for this operation, enabling further analysis and amendments. For example, should perhaps the ORDER BY be altered, or perhaps a different JOIN?

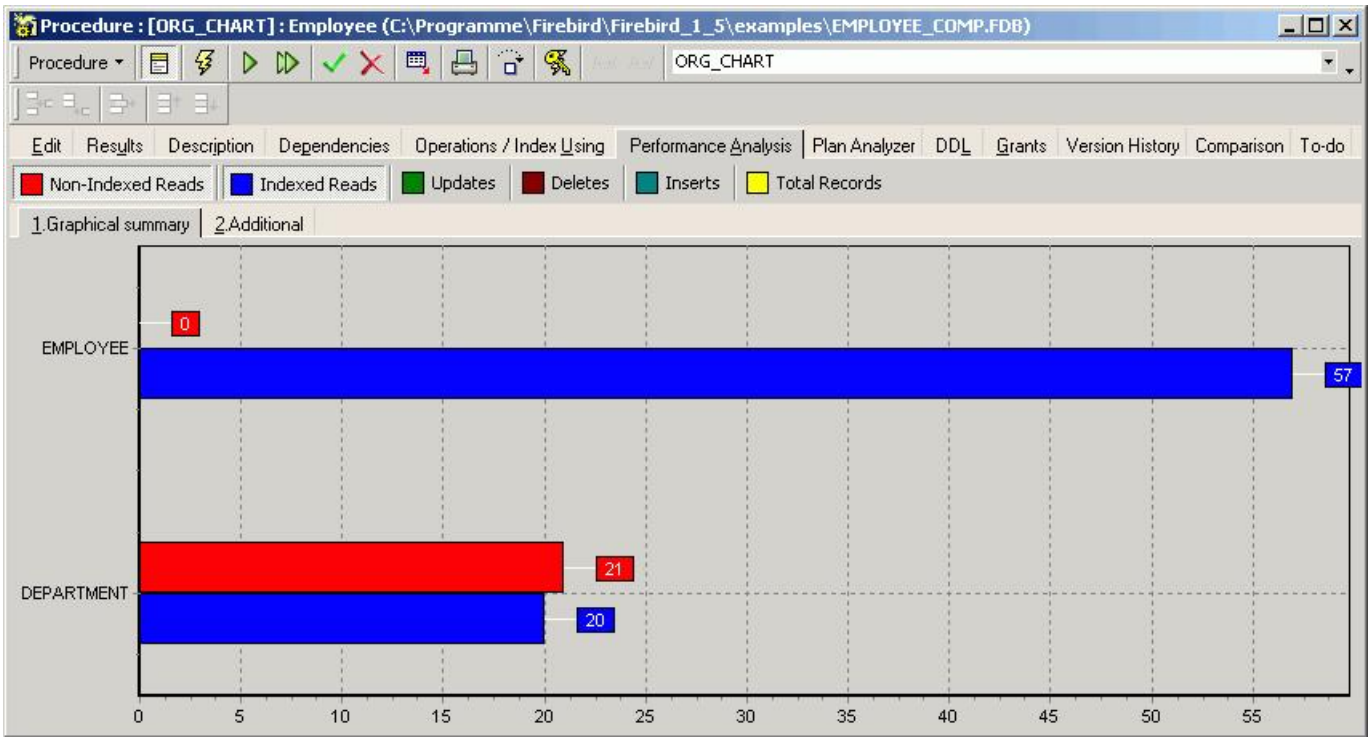


Input and output parameters and variable fields can be displayed by clicking on the buttons in the center of the editor. Alterations may be made directly in the SQL window and subsequently executed and committed.

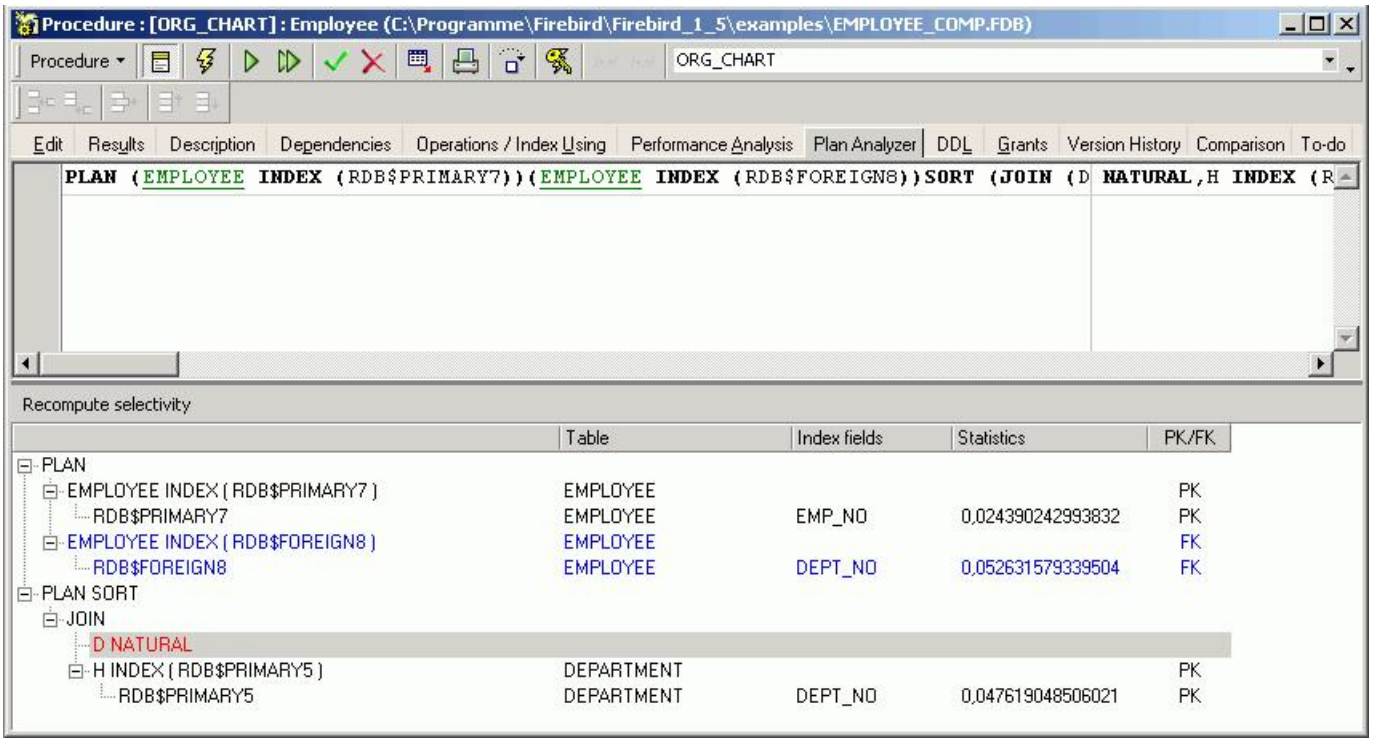
New to IBExpert v. 2.5.0.47 is the SP/Triggers/Views Analyzer in the IBExpert Tools menu. This loads all stored procedures and triggers in the active database, and all NATURAL operations are highlighted.

Performance Analysis

This page only appears once a procedure has been executed. Please refer to SQL Editor / Performance Analysis for details.



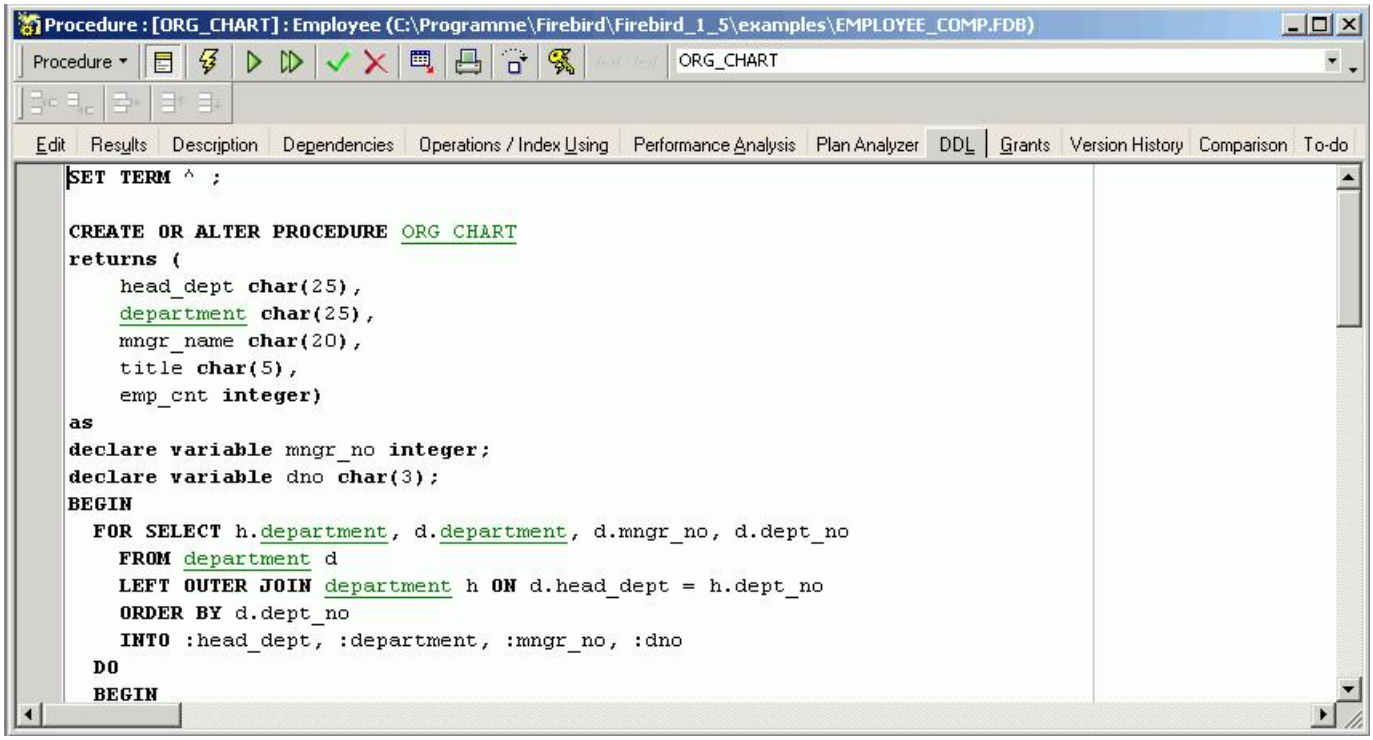
Plan Analyzer



Please refer to SQL Editor / Plan Analyzer.

DDL

The DDL page is new to IBExpert version 2004.6.17. It includes the CREATE PROCEDURE statement, stored procedure and parameter descriptions and GRANT statements.



Grants

Please refer to Table Editor / Grants and autogrant privileges.

Version History

Please refer to View / Version History.

Comparison

Please refer to Table Editor / Comparison.

To-Do

Please refer to Table Editor / To-Do.

Procedure using the `SUBSTRING()` function (`Susbstr` procedure)

Unfortunately Firebird 1.5 does not allow any variable parameters in the `SUBSTRING()` SQL function.

Although there are diverse UDF implementations, for those preferring to use stored procedures, here is an example from Lucas Franzen:

(For those of you who may be wondering what on earth "Donaudampfschiffahrtsgesellschaftskapitän" is, it is the German word for "Donau Steam Navigation Company Captain"!).

Call:

```
SELECT RESULT FROM SP_SUBSTRING
  ( INPUTSTRING, STARTPOS, NO_CHAR_FROM_STARTPOS ).

E.g.: SELECT RESULT FROM SP_SUBSTRING
  ( 'Donaudampfschiffahrtsgesellschaftskapitän', 1, 10 )
--> Donaudampf

E.g.: SELECT RESULT FROM SP_SUBSTRING
  ( 'Donaudampfschiffahrtsgesellschaftskapitän', 35, 8 )
--> kapitän


CREATE PROCEDURE SP_SUBSTRING (
  SRC                              VARCHAR (255),
  START_AT                         INTEGER,
  NLEN                             INTEGER
  )
RETURNS (
  RESULT                         VARCHAR (255)
  )
AS
  declare variable II INTEGER;
  declare variable VGL VARCHAR(255);
  declare variable PFX VARCHAR(255);
```

```
  declare variable C CHAR(1);
BEGIN

  /* Version : 1 */
  /* Author: LUC, 08.01.2003*/
  /* Description: */
  /*            */

 IF ( START_AT <= 0 ) THEN START_AT = 1;
 IF ( START_AT > 255 ) THEN START_AT = 255;

 IF ( NLEN > 255 ) THEN NLEN = 255;
 IF ( NLEN < 1 OR NLEN IS NULL ) THEN NLEN = 1;

 VGL = '';
 RESULT = '';
 PFX = '';

  IF ( START_AT > 1 ) THEN
  BEGIN
    II = 1;
    WHILE ( II < START_AT ) DO
    BEGIN
      PFX = PFX || '_';
      II = II + 1;
    END
  END

  II = START_AT;
  WHILE ( II < NLEN + START_AT ) DO
  BEGIN
    /* WHAT DOES THE STRING LOOK LIKE AT THE CURRENT POSITION, I.E. QUERY THE CURRENT CHARACTER */
   C = ' ';

    IF ( SRC LIKE PFX || ' %' ) THEN C = ' ';
    ELSE IF ( SRC LIKE PFX || 'A%' ) THEN C = 'A';
    ELSE IF ( SRC LIKE PFX || 'B%' ) THEN C = 'B';
    ELSE IF ( SRC LIKE PFX || 'C%' ) THEN C = 'C';
    ELSE IF ( SRC LIKE PFX || 'D%' ) THEN C = 'D';
    ELSE IF ( SRC LIKE PFX || 'E%' ) THEN C = 'E';
    ELSE IF ( SRC LIKE PFX || 'F%' ) THEN C = 'F';
    ELSE IF ( SRC LIKE PFX || 'G%' ) THEN C = 'G';
    ELSE IF ( SRC LIKE PFX || 'H%' ) THEN C = 'H';
    ELSE IF ( SRC LIKE PFX || 'I%' ) THEN C = 'I';
    ELSE IF ( SRC LIKE PFX || 'J%' ) THEN C = 'J';
    ELSE IF ( SRC LIKE PFX || 'K%' ) THEN C = 'K';
    ELSE IF ( SRC LIKE PFX || 'L%' ) THEN C = 'L';
    ELSE IF ( SRC LIKE PFX || 'M%' ) THEN C = 'M';
    ELSE IF ( SRC LIKE PFX || 'N%' ) THEN C = 'N';
    ELSE IF ( SRC LIKE PFX || 'O%' ) THEN C = 'O';
    ELSE IF ( SRC LIKE PFX || 'P%' ) THEN C = 'P';
    ELSE IF ( SRC LIKE PFX || 'Q%' ) THEN C = 'Q';
    ELSE IF ( SRC LIKE PFX || 'R%' ) THEN C = 'R';
    ELSE IF ( SRC LIKE PFX || 'S%' ) THEN C = 'S';
    ELSE IF ( SRC LIKE PFX || 'T%' ) THEN C = 'T';
    ELSE IF ( SRC LIKE PFX || 'U%' ) THEN C = 'U';
    ELSE IF ( SRC LIKE PFX || 'V%' ) THEN C = 'V';
    ELSE IF ( SRC LIKE PFX || 'W%' ) THEN C = 'W';
    ELSE IF ( SRC LIKE PFX || 'X%' ) THEN C = 'X';
    ELSE IF ( SRC LIKE PFX || 'Y%' ) THEN C = 'Y';
    ELSE IF ( SRC LIKE PFX || 'Z%' ) THEN C = 'Z';

    ELSE IF ( SRC LIKE PFX || 'a%' ) THEN C = 'a';
    ELSE IF ( SRC LIKE PFX || 'b%' ) THEN C = 'b';
    ELSE IF ( SRC LIKE PFX || 'c%' ) THEN C = 'c';
    ELSE IF ( SRC LIKE PFX || 'd%' ) THEN C = 'd';
    ELSE IF ( SRC LIKE PFX || 'e%' ) THEN C = 'e';
    ELSE IF ( SRC LIKE PFX || 'f%' ) THEN C = 'f';
    ELSE IF ( SRC LIKE PFX || 'g%' ) THEN C = 'g';
    ELSE IF ( SRC LIKE PFX || 'h%' ) THEN C = 'h';
    ELSE IF ( SRC LIKE PFX || 'i%' ) THEN C = 'i';
    ELSE IF ( SRC LIKE PFX || 'j%' ) THEN C = 'j';
    ELSE IF ( SRC LIKE PFX || 'k%' ) THEN C = 'k';
    ELSE IF ( SRC LIKE PFX || 'l%' ) THEN C = 'l';
    ELSE IF ( SRC LIKE PFX || 'm%' ) THEN C = 'm';
    ELSE IF ( SRC LIKE PFX || 'n%' ) THEN C = 'n';
    ELSE IF ( SRC LIKE PFX || 'o%' ) THEN C = 'o';
    ELSE IF ( SRC LIKE PFX || 'p%' ) THEN C = 'p';
    ELSE IF ( SRC LIKE PFX || 'q%' ) THEN C = 'q';
    ELSE IF ( SRC LIKE PFX || 'r%' ) THEN C = 'r';
    ELSE IF ( SRC LIKE PFX || 's%' ) THEN C = 's';
    ELSE IF ( SRC LIKE PFX || 't%' ) THEN C = 't';
    ELSE IF ( SRC LIKE PFX || 'u%' ) THEN C = 'u';
    ELSE IF ( SRC LIKE PFX || 'v%' ) THEN C = 'v';
    ELSE IF ( SRC LIKE PFX || 'w%' ) THEN C = 'w';
    ELSE IF ( SRC LIKE PFX || 'x%' ) THEN C = 'x';
    ELSE IF ( SRC LIKE PFX || 'y%' ) THEN C = 'y';
    ELSE IF ( SRC LIKE PFX || 'z%' ) THEN C = 'z';

    ELSE IF ( SRC LIKE PFX || '0%' ) THEN C = '0';
    ELSE IF ( SRC LIKE PFX || '1%' ) THEN C = '1';
    ELSE IF ( SRC LIKE PFX || '2%' ) THEN C = '2';
    ELSE IF ( SRC LIKE PFX || '3%' ) THEN C = '3';
```

```
    ELSE IF ( SRC LIKE PFX || '4%' ) THEN C = '4';
    ELSE IF ( SRC LIKE PFX || '5%' ) THEN C = '5';
    ELSE IF ( SRC LIKE PFX || '6%' ) THEN C = '6';
    ELSE IF ( SRC LIKE PFX || '7%' ) THEN C = '7';
    ELSE IF ( SRC LIKE PFX || '8%' ) THEN C = '8';
    ELSE IF ( SRC LIKE PFX || '9%' ) THEN C = '9';

    ELSE IF ( SRC LIKE PFX || 'ä%' ) THEN C = 'ä';
    ELSE IF ( SRC LIKE PFX || 'ö%' ) THEN C = 'ö';
    ELSE IF ( SRC LIKE PFX || 'ü%' ) THEN C = 'ü';
    ELSE IF ( SRC LIKE PFX || 'Ä%' ) THEN C = 'Ä';
    ELSE IF ( SRC LIKE PFX || 'Ö%' ) THEN C = 'Ö';
    ELSE IF ( SRC LIKE PFX || 'Ü%' ) THEN C = 'Ü';
    ELSE IF ( SRC LIKE PFX || 'ß%' ) THEN C = 'ß';

    ELSE IF ( SRC LIKE PFX || '!%' ) THEN C = '!';
    ELSE IF ( SRC LIKE PFX || '"%' ) THEN C = '"';
    ELSE IF ( SRC LIKE PFX || '§%' ) THEN C = '§';
    ELSE IF ( SRC LIKE PFX || '$%' ) THEN C = '$';
    ELSE IF ( SRC LIKE PFX || '&%' ) THEN C = '&';
    ELSE IF ( SRC LIKE PFX || '/%' ) THEN C = '/';
    ELSE IF ( SRC LIKE PFX || '(%' ) THEN C = '(';
    ELSE IF ( SRC LIKE PFX || ')%' ) THEN C = ')';
    ELSE IF ( SRC LIKE PFX || '=%' ) THEN C = '=';

    ELSE IF ( SRC LIKE PFX || '@%' ) THEN C = '@';
    ELSE IF ( SRC LIKE PFX || %' ) THEN C = ';
    ELSE IF ( SRC LIKE PFX || '*%' ) THEN C = '*';
    ELSE IF ( SRC LIKE PFX || '~%' ) THEN C = '~';
    ELSE IF ( SRC LIKE PFX || '#%' ) THEN C = '#';
    ELSE IF ( SRC LIKE PFX || '%' ) THEN C = '´';
    ELSE IF ( SRC LIKE PFX || %' ) THEN C = ';

    ELSE IF ( SRC LIKE PFX || 'Á%' ) THEN C = 'Á';
    ELSE IF ( SRC LIKE PFX || 'É%' ) THEN C = 'É';
    ELSE IF ( SRC LIKE PFX || 'Í%' ) THEN C = 'Í';
    ELSE IF ( SRC LIKE PFX || 'Ó%' ) THEN C = 'Ó';
    ELSE IF ( SRC LIKE PFX || 'Ú%' ) THEN C = 'Ú';
    ELSE IF ( SRC LIKE PFX || 'á%' ) THEN C = 'á';
    ELSE IF ( SRC LIKE PFX || 'é%' ) THEN C = 'é';
    ELSE IF ( SRC LIKE PFX || 'í%' ) THEN C = 'í';
    ELSE IF ( SRC LIKE PFX || 'ó%' ) THEN C = 'ó';
    ELSE IF ( SRC LIKE PFX || 'ú%' ) THEN C = 'ú';

    ELSE IF ( SRC LIKE PFX || 'À%' ) THEN C = 'À';
    ELSE IF ( SRC LIKE PFX || 'È%' ) THEN C = 'È';
    ELSE IF ( SRC LIKE PFX || 'Ì%' ) THEN C = 'Ì';
    ELSE IF ( SRC LIKE PFX || 'Ò%' ) THEN C = 'Ò';
    ELSE IF ( SRC LIKE PFX || 'Ù%' ) THEN C = 'Ù';
    ELSE IF ( SRC LIKE PFX || 'à%' ) THEN C = 'à';
    ELSE IF ( SRC LIKE PFX || 'è%' ) THEN C = 'è';
    ELSE IF ( SRC LIKE PFX || 'ì%' ) THEN C = 'ì';
    ELSE IF ( SRC LIKE PFX || 'ò%' ) THEN C = 'ò';
    ELSE IF ( SRC LIKE PFX || 'ù%' ) THEN C = 'ù';

    ELSE IF ( SRC LIKE PFX || 'Â%' ) THEN C = 'Â';
    ELSE IF ( SRC LIKE PFX || 'Ê%' ) THEN C = 'Ê';
    ELSE IF ( SRC LIKE PFX || 'Î%' ) THEN C = 'Î';
    ELSE IF ( SRC LIKE PFX || 'Ô%' ) THEN C = 'Ô';
    ELSE IF ( SRC LIKE PFX || 'Û%' ) THEN C = 'Û';
    ELSE IF ( SRC LIKE PFX || 'â%' ) THEN C = 'â';
    ELSE IF ( SRC LIKE PFX || 'ê%' ) THEN C = 'ê';
    ELSE IF ( SRC LIKE PFX || 'î%' ) THEN C = 'î';
    ELSE IF ( SRC LIKE PFX || 'ô%' ) THEN C = 'ô';
    ELSE IF ( SRC LIKE PFX || 'û%' ) THEN C = 'û';

    ELSE IF ( SRC LIKE PFX || '{%' ) THEN C = '{';
    ELSE IF ( SRC LIKE PFX || '}%' ) THEN C = '}';
    ELSE IF ( SRC LIKE PFX || '[%' ) THEN C = '[';
    ELSE IF ( SRC LIKE PFX || ']%' ) THEN C = ']';

    RESULT = RESULT || :C;

    PFX = PFX || '_';
    II = II + 1;
    IF ( II > 255 ) THEN
    BEGIN
      SUSPEND;
      EXIT;
    END
  END
   SUSPEND;
END
```

Debug procedure or trigger (IBExpert Debugger)

A stored procedure or trigger can be simply and quickly debugged in IBExpert. (This feature is unfortunately not included in the IBExpert Personal Edition.) IBExpert simulates running the procedure or trigger on the database server by interpreting the procedure and running the commands one at a time. It offers a number of useful functionalities, such as *breakpoints*, *step into*, *trace* or *run to cursor*, you can watch certain parameters, analyze the performance and indices

used, and you can even change values on the fly. If you have Delphi experience you will easily find your way around the Debugger as key strokes etc. are the same.

Simply open the procedure or trigger in the Procedure Editor or Trigger Editor by double-clicking on the procedure/trigger name in the DB Explorer and click the *Debug* icon on the Procedure or Trigger Editor toolbar (or [Shift + Ctrl + D]) to start the *Debugger* window.

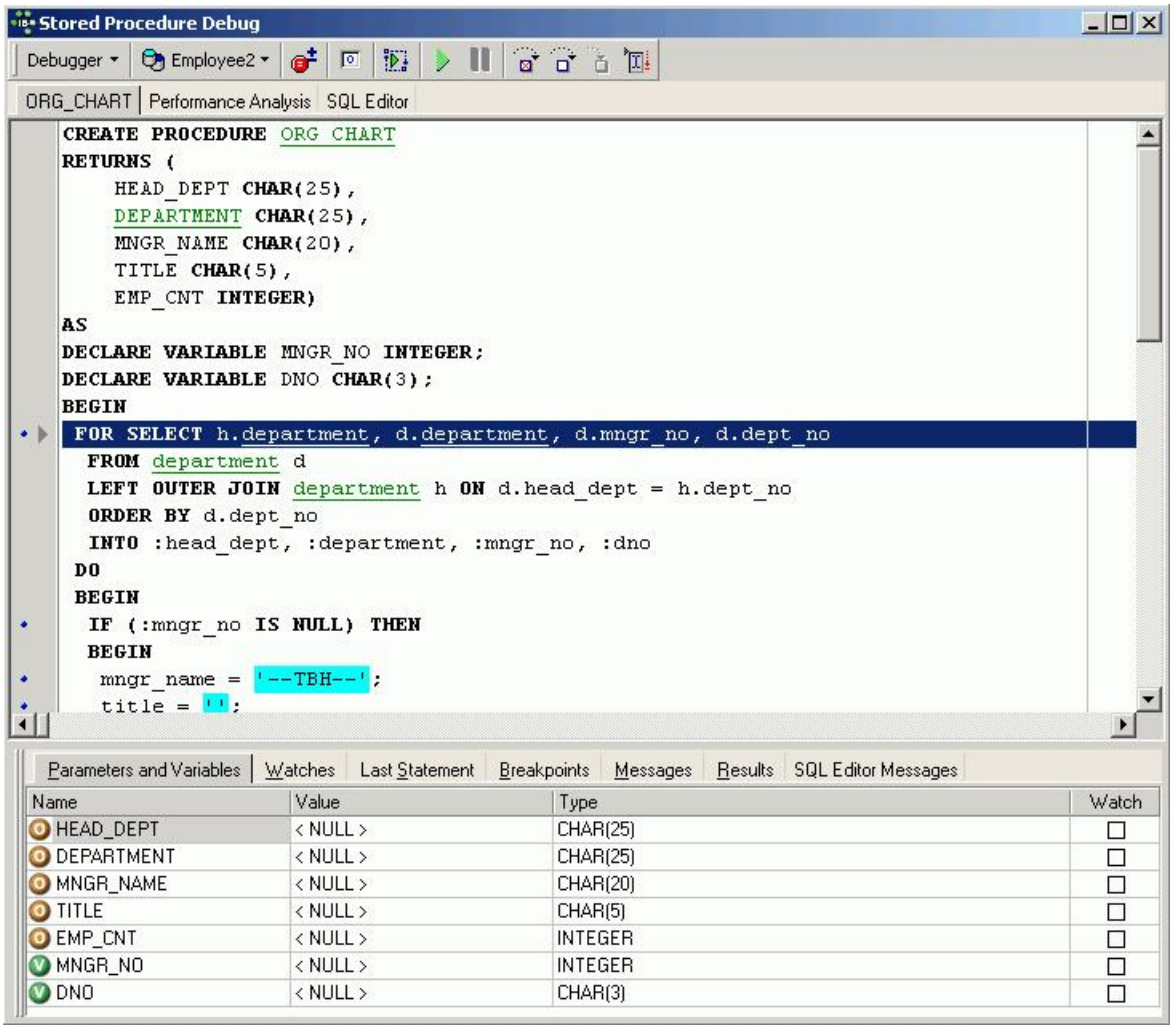
The Debug Procedure/Trigger Editor comprises 3 pages, the *Debug* page (described here), Performance Analysis and the SQL Editor.

The Debugger also supports the following new Firebird 2.0 features in the named IBExpert versions:

- IBExpert version 2005.04.24: `SELECT ... FROM (SELECT ...)`
- IBExpert version 2005.06.07: `IS DISTINCT FROM`
- IBExpert version 2005.09.25: `INSERT ... RETURNING`
- IBExpert version 2005.09.25: Added support for aliases of nested `SELECT`s as in the following example:

```
SELECT * FROM (SELECT RDB$RELATION_NAME,
      RDB$RELATION_ID
      FROM RDB$RELATIONS) AS R
      (RELATION_NAME, RELATION_ID)
```

- IBExpert version 2005.09.25: `TRIM` function
- IBExpert version 2005.09.25: `ROWS` clause
- IBExpert version 2006.10.14: `CROSS JOIN`



The upper half of this dialog displays the SQL text. Since IBExpert version 2006.10.14 the object name (if applicable) is displayed in the Windows bar. The lower area displays a number of tabs:

Parameters and Variables

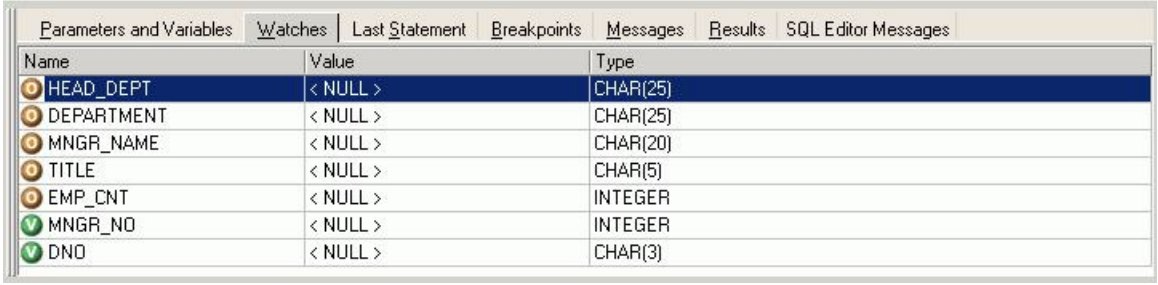
The parameters are listed in a grid. The circular symbols to the left of the name indicate whether the parameters are input (`I`) or output (`O`). Variables logically have the key (`V`). Further information displayed here includes the parameter value, scope and datatype. The *Watch* boxes can be checked, to specify which variables should be observed.

Since IBExpert version 2004.9.12.1 there is the added possibility to initialize parameters/variables using values of any data grid. Just drag and drop a cell value from any data grid onto the corresponding node in the parameters/variables list to initialize the variable with the value of the data cell. It is also possible to initialize multiple variables/parameters by holding the [Ctrl] key when dropping. In this case IBExpert searches for the corresponding parameter/variable (by name) for each field in the data record, and if the parameter/variable is found it will be initialized with the value of the field with the same name.

Since IBExpert version 2004.04.01.1 there is added support for default values of input parameters (Firebird 2).

And IBExpert version 2005.12.04 introduced the possibility to debug universal triggers which use the context variables `INSERTING`/`UPDATING`/`DELETING`. The debugger interprets these variables as regular input parameters with a `BOOLEAN` datatype and they are `FALSE` by default.

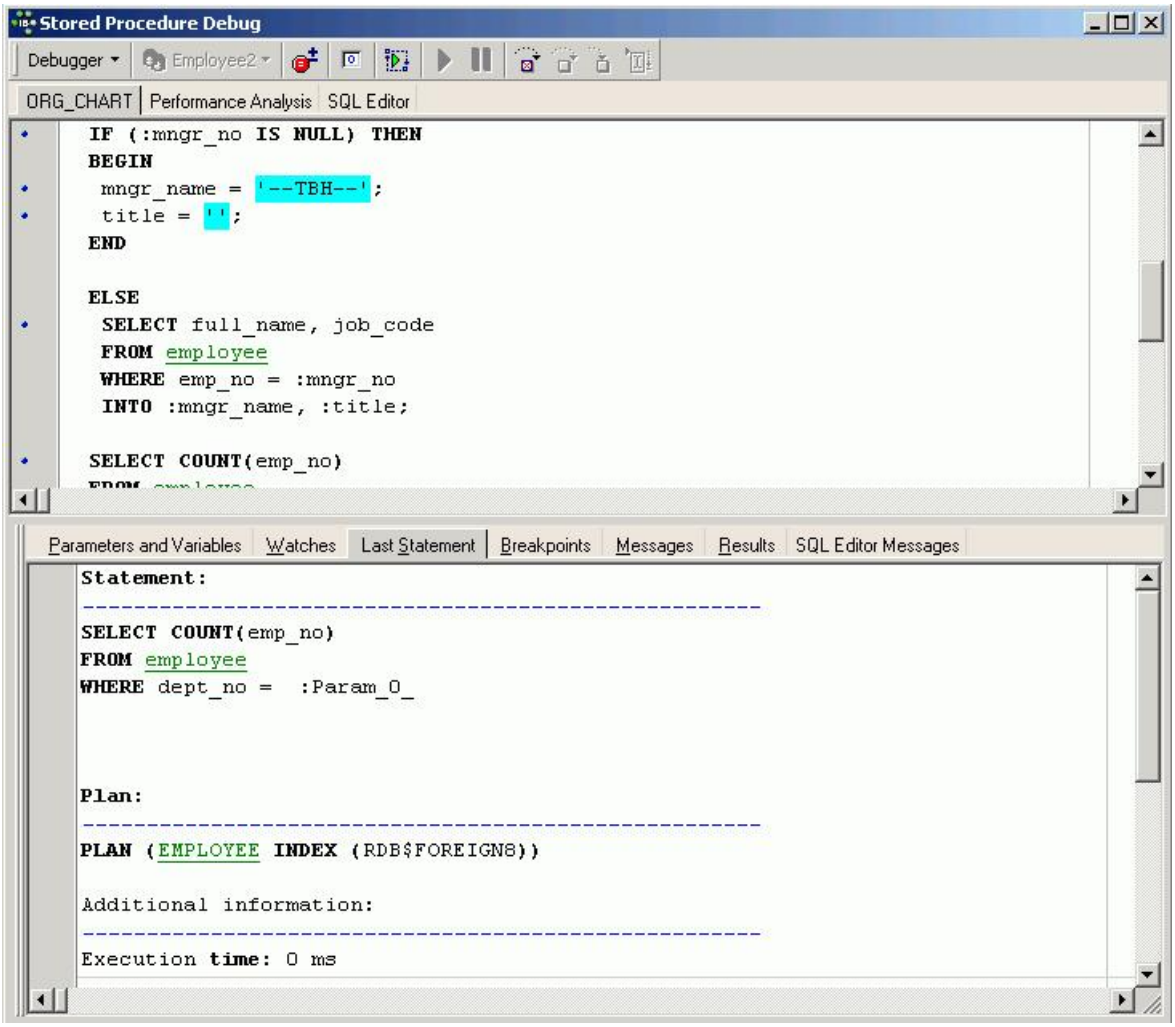
Watches



The *Watches* tab displays those parameters and variables that have been checked for particular observation in the previous window.

Last Statement

Following execution, the last internal statement is displayed here, along with additional information such as execution time:



Breakpoints

This page displays the positions where breakpoints have been specified, using the respective icon in the Debug Procedure toolbar, the [F5] key, or by clicking on the blue points in the SQL left margin.

When the procedure is executed (using the respective icon or [F9]), it always stops automatically at these breakpoints. The procedure can thus be executed step by step, either using [F8] (or the respective toolbar icon) to continue execution step by step (not including the next sublevel), or [F7] (or the respective toolbar icon) to continue step by step including the next sublevel. Please note that this *Trace Into* [F7] function is new to IBExpert version 2004.04.01.1.

Alternatively, if you have a procedure or trigger containing cursors, you can of course use the *Run to Cursor* icon, or [F4], to execute a part of a stored procedure or trigger up to the location of the cursor in the Code Editor.

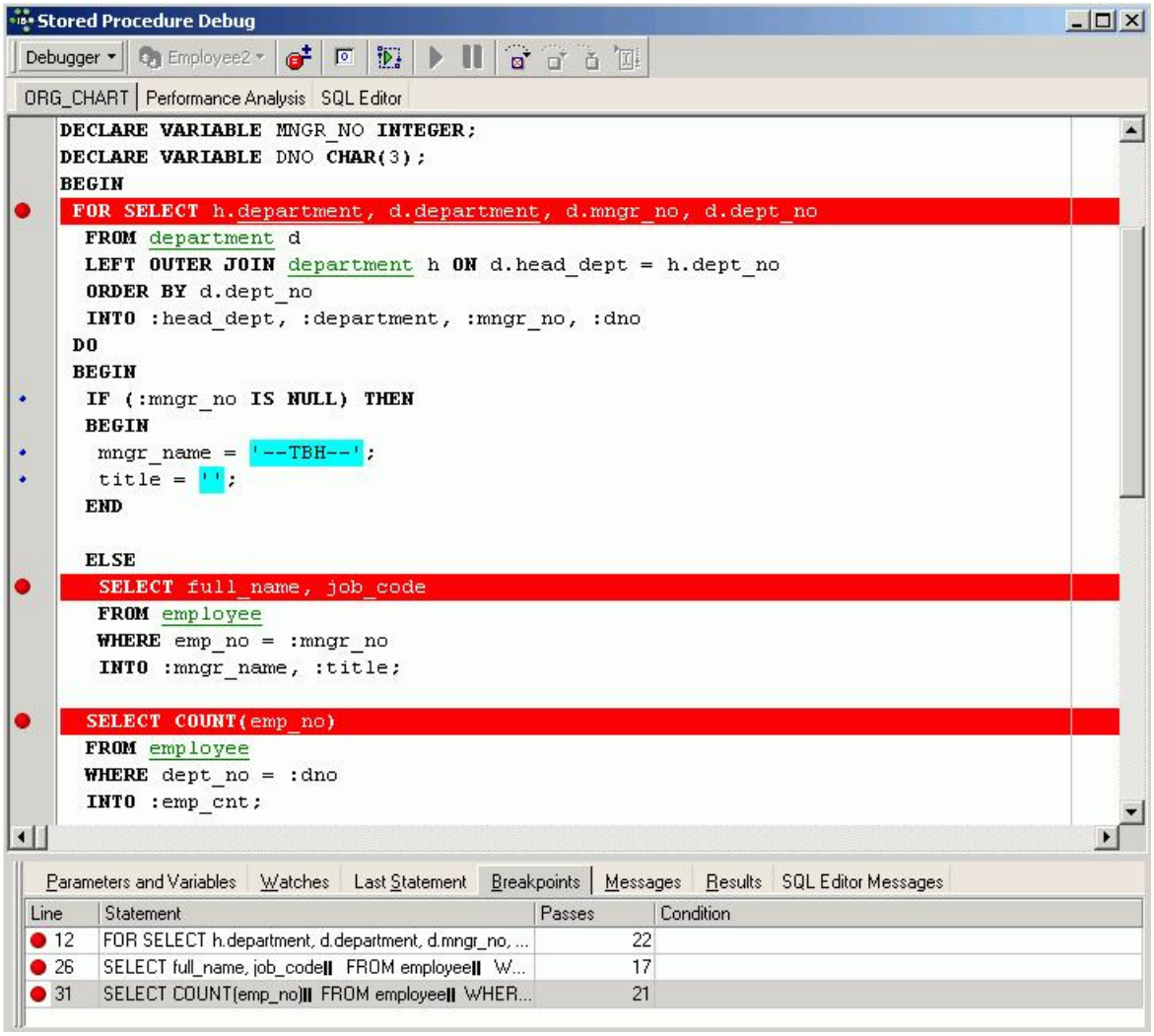
Since IBExpert version 2006.06.05 it is also possible to define breakpoints using comments. To define a breakpoint simply write a special comment line:

```
-- IBE_BREAKPOINT
```

or

```
/* IBE_BREAKPOINT */
```

before the statement where the debug process should be paused.

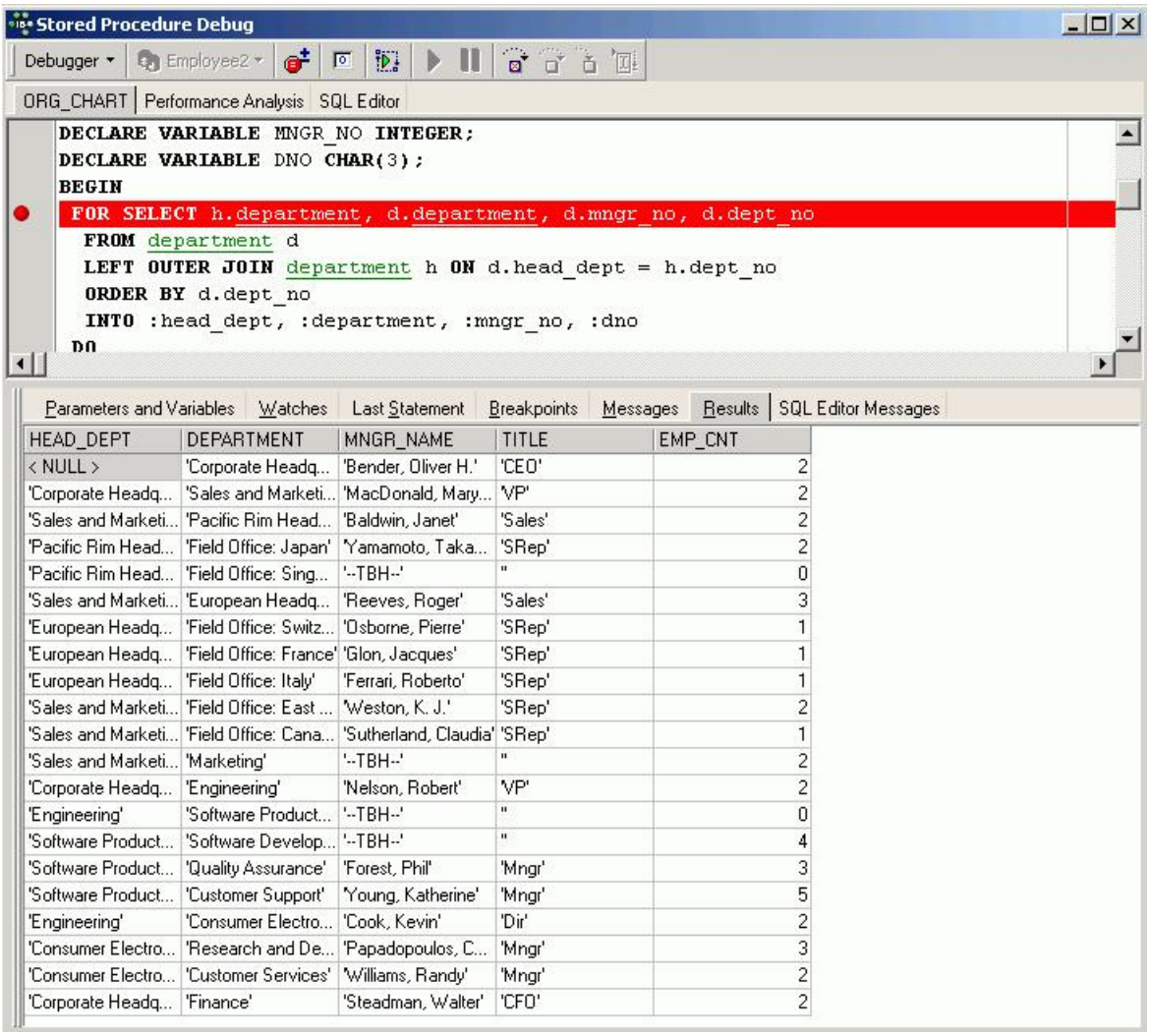


Messages

These indicate the sort of error that has occurred and where, by highlighting the relevant SQL row.

Results

This page only appears if there are output parameters in the procedure.



SQL Editor Messages

These are displayed here when applicable.

When debugging a procedure, first take a look at the values of the parameters and then use [F8] to go through the procedure step by step ([F9] executes fully). After each step, all variable values can be seen. Don't forget to work with breakpoints [F5]. Of course, the Debug Procedure toolbar offers all these operations and more.

Alter procedure

Procedures can be altered directly in the Procedure Editor, started by double-clicking directly on the procedure name in the DB Explorer. Alternatively use the DB Explorer's right mouse-click menu item *Edit Procedure* or key combination [Ctrl + O].

`ALTER PROCEDURE` has exactly the same syntax as `CREATE PROCEDURE`. In fact, when procedures are altered the original procedure definition is replaced. It may seem that `ALTER PROCEDURE` is therefore not necessary, as a procedure could be dropped and then recreated to carry out any changes. However this will not work if the procedure to be changed is called by another procedure. If procedure A calls procedure B, procedure B cannot be dropped because procedure A depends on its existence.

The SQL syntax for this command is:

```
ALTER PROCEDURE <procedure_name>
<revised_input_parameter_list>
RETURNS
<revised_return_parameter_list>
AS
<local_variable_declarations>
BEGIN
<procedure_body>
END
```

A procedure can only be altered by the original creator or by the SYSDBA user.

Drop procedure/delete procedure

A procedure may only be dropped, if it is not being used at the time of deletion. Also it may not be dropped if it is used by other procedures, triggers, views or SELECTS, until this dependency is removed.

The Procedure Editor / Dependencies page displays which database objects use this procedure, and which objects this procedure uses. Most database objects can be dropped directly on the *Dependencies* page or the Dependencies Viewer by using the right-click menu on the selected object, and choosing the menu item *Drop Object* or [Ctrl + Del].

To drop a procedure use the DB Explorer right mouse-click menu item *Drop Procedure...* (or [Ctrl + Del]).

IBExpert asks for confirmation:

before finally dropping the procedure. Once dropped, it cannot be retrieved; the procedure has to be recreated, if a mistake has been made!

Using SQL the syntax is:

```
DROP PROCEDURE <procedure_name>;
```

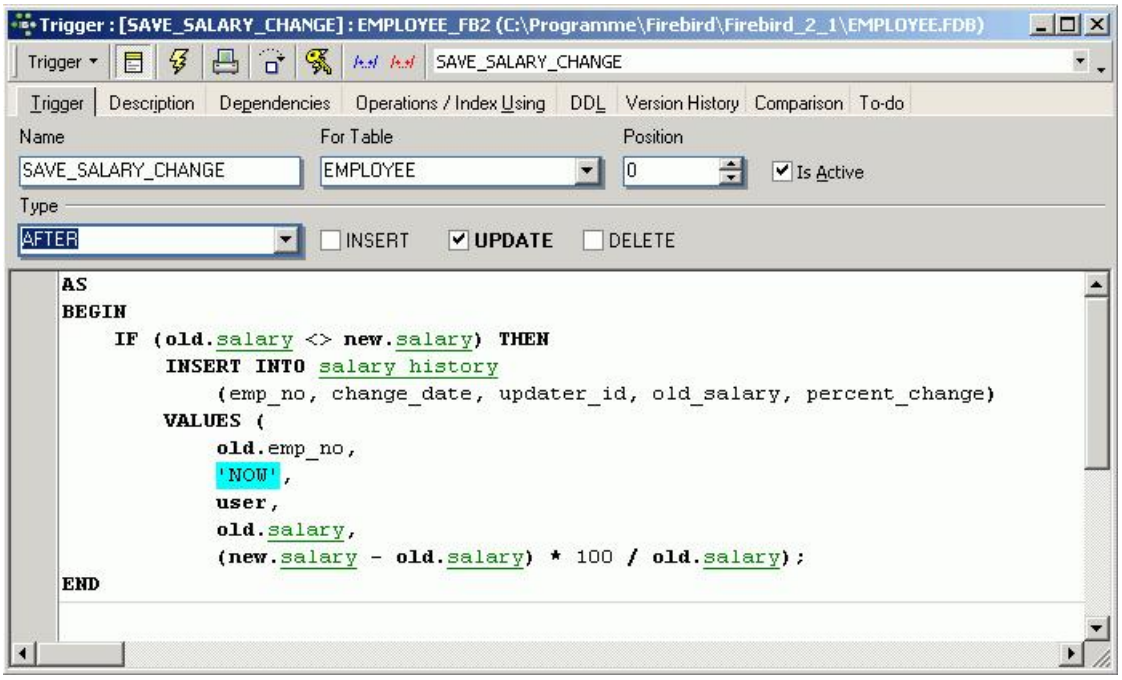
A procedure can only be dropped by its creator or the SYSDBA.

See also:
SELECT
DDL - Data Definition Language
Stored Procedure and Trigger Language
Create Stored Procedure from SELECT
Dependencies Viewer
Firebird for the database expert - Episode 1: Indexes
Writing stored procedures and triggers
Firebird 2 SQL Reference Guide

Trigger

A trigger is an independent series of commands stored as a self-contained program (SQL script) in the database. Triggers are executed automatically in the database when certain events occur. For example, it is possible to check before an insert, whether a primary key already exists or not, and if necessary allocate a value by a generator. These events are database-, table- or row-based.

Triggers are the so-called database police force, as they are vital for database integrity and security by enforcing the rules programmed by the database developer. They can include one or more execute commands. They can also be used as an alarm (= event alerter) that sends an event of a certain name to the InterBase/Firebird *Event Manager*.



Triggers take no input parameters and do not return values.

The sequence in which triggers are specified is determined by the term `TRIGGER POSITION`, and different trigger types can be specified (see below).

They can be created, edited and deleted using the IBExpert DB Explorer right-click menu, from the Table Editor or Field Editor, or directly in the IBExpert SQL Editor.

Since Firebird 1.5 universal triggers (which can be used simultaneously for insert and/or update and/or delete) are available and Firebird 2.1 introduced database triggers (see below for further information).

An example of a trigger:

```
CREATE TRIGGER TEST_TRIG FOR TEST
ACTIVE BEFORE INSERT POSITION 0
AS
begin
```

```
    if (new.id is null) then
       new.id=gen_id (GLOB_ID,1);
end
```

Several triggers can be created for one event. The `POSITION` parameter determines the sequence in which the triggers are executed.

Triggers are almost identical to stored procedures, the main difference being the way they are called. Triggers are called automatically when a change to a row in a [Table | table]] occurs, or certain database actions occur. Most of what is said about stored procedures applies to triggers as well, and they share the same language, PSQL.

Database triggers

Database triggers were implemented in Firebird 2.1. These are user-defined PSQL modules that can be defined to fire in various connection-level and transaction-level events. This allows you to, for example, set up a protocol relatively quickly and easily.

Database trigger types

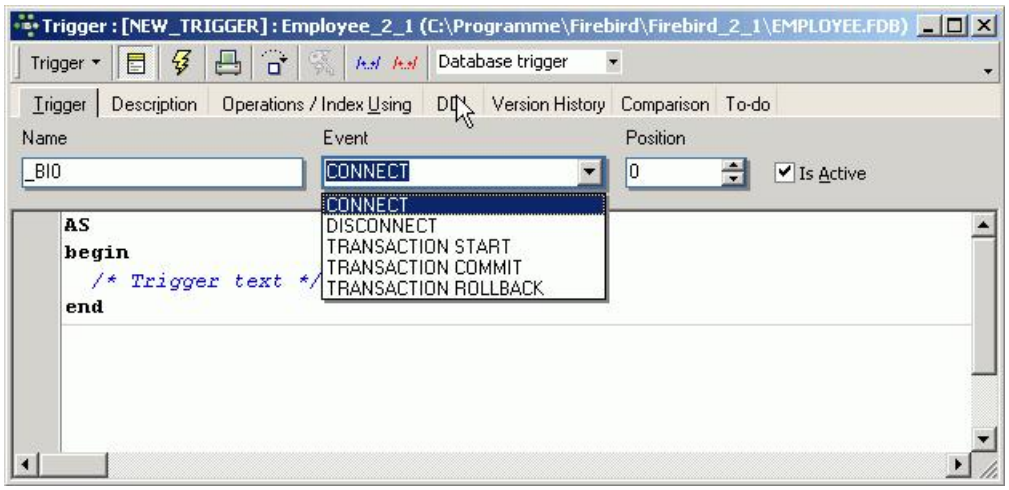
Database-wide triggers can be fired on the following database trigger types:

`CONNECT`	The database connection is established, a transaction begins, triggers are fired - uncaught exceptions rollback the transaction, disconnect the attachment and are returned to the client. Finally the transaction is committed.
`DISCONNECT`	A transaction is started, triggers are fired - uncaught exceptions rollback the transaction, disconnect the attachment and are stopped. The transaction is committed and the attachment disconnected.
`TRANSACTION START`	Triggers are fired in the newly-created user transaction - uncaught exceptions are returned to the client and the transaction is rolled back.
`TRANSACTION COMMIT`	Triggers are fired in the committing transaction - uncaught exceptions rollback the trigger's savepoint, the commit command is aborted and an exception is returned to the client. For two-phase transactions the triggers are fired in `PREPARE` and not in `COMMIT`.
`TRANSACTION ROLLBACK`	Triggers are fired in the rolling-back transaction - changes made will be rolled back together with the transaction, and exceptions are stopped.

Only the SYSDBA or the database owner can:

- define database triggers
- switch them of for a new connection by:
  - `new isc_dpb_no_db_triggers` tag
  - `new -no_dbtriggers` switch in utilities

In IBExpert database triggers can be created, edited and deleted in the same way as table-bound triggers (see *Newtrigger* for details). Simply switch to *Database trigger* in the toolbar, to access the options specific to database triggers:



An example of a database trigger (source *Firebird 2.1 What's New*, by Vladyslav Khorsum):

Example of an `ON CONNECT` trigger

```
isql temp.fdb -user SYSDBA -pass masterkey
Database: temp.fdb, User: SYSDBA
SQL> SET TERM ^ ;
SQL> CREATE EXCEPTION EX_CONNECT 'Forbidden !' ^
SQL> CREATE OR ALTER TRIGGER TRG_CONN ON CONNECT
CON> AS
CON> BEGIN
CON> IF (<bad user>)
CON> THEN EXCEPTION EX_CONNECT USER || ' not allowed !';
CON> END ^
SQL> EXIT ^

isql temp.fdb -user BAD_USER -pass ...
```

```
Statement failed, SQLCODE = -836
exception 217
-EX_CONNECT
-BAD_USER not allowed !
-At trigger 'TRG_CONN' line: 5, col: 3
Use CONNECT or CREATE DATABASE to specify a database
SQL> EXIT;
```

If you encounter problems with an `ON CONNECT` trigger, so that noone can connect to the database any more, use the `-no_dbtriggers` switch in the utilities:

```
isql temp.fdb -user SYSDBA -pass masterkey
-nodbtriggers Database: temp.fdb, User: SYSDBA
SQL> ALTER TRIGGER TRG_CONN INACTIVE;
SQL> EXIT;
```

Database triggers can be quickly and easily defined in IBExpert's Trigger Editor (see below).

Table triggers

Table trigger types

Trigger types refer to the trigger status (`ACTIVE` or `INACTIVE`), the trigger position (`BEFORE` or `AFTER`) and the operation type (`INSERT`, `UPDATE` or `DELETE`).

They are specified following the definition of the table or view name, and before the trigger body.

ACTIVE or INACTIVE

`ACTIVE` or `INACTIVE` is specified at the time a trigger is created. `ACTIVE` is the default if neither of these keywords is specified. An inactive trigger does not execute.

BEFORE or AFTER

A trigger needs to be defined to fire either `BEFORE` or `AFTER` an operation. A `BEFORE INSERT` trigger fires before a new row is actually inserted into the table; an `AFTER INSERT` trigger fires after the row has been inserted.

`BEFORE` triggers are generally used for two purposes:

1. They can be used to determine whether the operation should proceed, i.e. certain parameters can be tested to determine whether the row should be inserted, updated or deleted or not. If not, an exception can be raised and the transaction rolled back.

2. `BEFORE` triggers can also be used to determine whether there are linked rows that might be affected by the operation. For example, a trigger might be used to automatically reassign sales before deleting a sales employee.

`AFTER` triggers are generally used to update columns in linked tables that depend on the row being inserted, updated or deleted for their values. For example, the `PERCENT_CHANGE` column in the `SALARY_HISTORY` table is maintained using an `AFTER UPDATE` trigger on the `EMPLOYEE` table.

To summarize: Use `BEFORE` until all data manipulation operations have been completed. The `EMPLOYEE` database trigger `SET_CUST_NO` is an example of a `BEFORE INSERT`, as a new customer number is generated before the data set has been inserted.

When manipulation of the table data should have been concluded before checking or altering other data, then use an `AFTER` trigger. The `EMPLOYEE` database trigger `SAVE_SALARY_CHANGE` is an example of `AFTER UPDATE` trigger, as the changes to the data have already been completed, before the trigger fires.

INSERT, UPDATE, DELETE

A trigger must be defined to fire on one of the keywords `INSERT`, `UPDATE` or `DELETE`.

1. An `INSERT` trigger fires before or after a row is inserted into the table.
2. An `UPDATE` trigger fires when a row is modified in the table.
3. A `DELETE` trigger fires when a row is deleted from the table.

If the same trigger needs to fire on more than one operation, a universal trigger needs to be defined. Before Firebird 1.5 triggers were restricted to either insert or update or delete actions, but now only one trigger needs to be created for all of these. For example:

```
AS
   BEGIN
      if (new.bez<>'')
      then new.bez=upper(new.bez);
   END
```

The `' ' UPPER` applies to `INSERT` and `UPDATE` operations.

Please note that special characters, such as German umlauts, are not recognized and altered to upper case, as the character is treated technically as a special character, and not an alphabetical letter.

For further information regarding `NEW` variables, please refer to `NEW` and `OLD` context variables.

NEW and OLD context variables

In triggers (but not in stored procedures), InterBase/Firebird provides two context variables that maintain information about the row being inserted, updated or deleted:

1. `OLD.columnName` refers to the current or previous values in a row being updated or deleted. It is not relevant for `INSERT` triggers.
2. `NEW.columnName` refers to the new values in a row being inserted or updated. It is not relevant for `DELETE` triggers.

Using the `OLD.` and `NEW.` values you can easily create history records, calculate the amount or percentage of change in a numeric value, find records in another table that match either the `OLD.` or `NEW.` value or do pretty well anything else you can think of. Please note that `NEW.` variables can be modified in a `BEFORE` trigger; since the introduction of Firebird 2.0 it is not so easy to alter them in an `AFTER` trigger. `OLD.` variables cannot be modified.

It is possible to read to or write from these trigger variables.

New to Firebird 2.0: Restrictions on assignment to context variables in triggers

- Assignments to the `OLD` context variables are now prohibited for every kind of trigger.
- Assignments to `NEW` context variables in `AFTER`-triggers are also prohibited.

Tip: If you receive an unexpected error `Cannot update a read-only column` then violation of one of these restrictions will be the source of the exception.

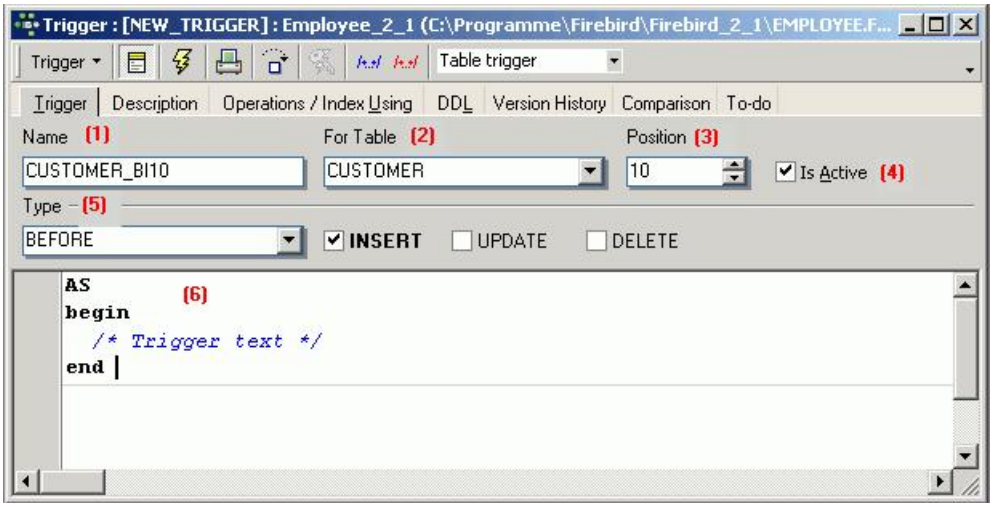
New trigger

There are numerous ways to create a trigger in IBExpert.

1. Using the IBExpert Database menu item, *New Trigger* or the respective icon on the New Database Object toolbar.

2. From the DB Explorer by right-clicking on the highlighted trigger branch of the relevant connected database (or key combination [Ctrl + N]).

Both these options open the Trigger Editor:

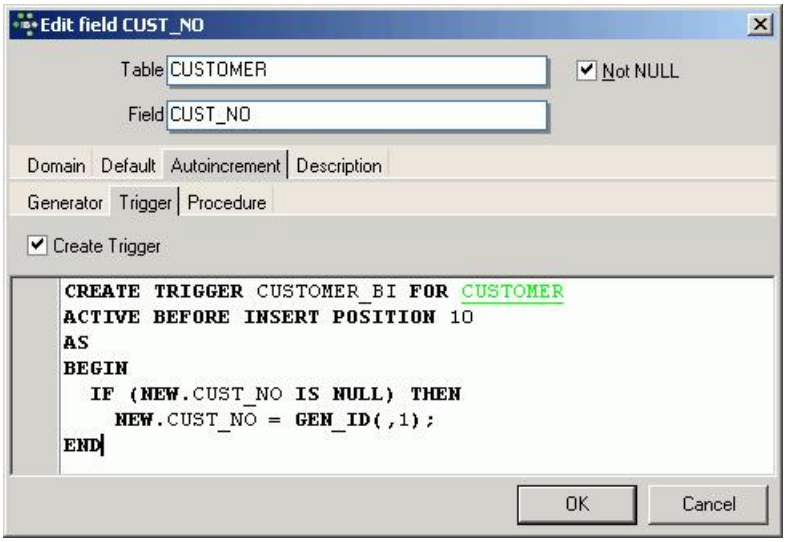


The Trigger Editor's first page allows the following to be specified simply and quickly, with the aid of pull-down lists, provided the lazy mode has been switched on:

- **(1) Name:** the trigger name can be altered as wished, if you do not wish to keep the default name. As with all database objects it is important to make rule about , which will aid you and other developers in the years to come to easily recognize objects, where they belong and their relationship to other objects. The illustration above depicts a `BEFORE INSERT` trigger. The name is composed of the table name, `BI` is the abbreviation for *Before Insert* and 10 denotes the specified position.
- **(2) For Table:** select the table or view name from the drop-down list.
- **(3) Position:** 255 positions are allowed per table, (starting at 0, up to 254). Several triggers on a table can also have the same firing position if it is irrelevant which one is fired first. As the positions do not have to be consecutive numbers it is wise to develop a convention, beginning let's say with 50, and numbering in 10 or 20 intervals. That way, you can insert and position new triggers at any time, without having to alter all your existing triggers to adjust the firing position. It's extremely important to layer the execution order of your triggers for logical reasons. For example, The `before insert` logging trigger on a table needs to know the data set's primary key, so the `before insert` primary key trigger needs to be fired first.
- **(4) Is Active:** check the box active/inactive as appropiate.
- **(5) Type:** specify trigger type as `BEFORE` or `AFTER`, and check the action(s) `INSERT`, `UPATE` and/or `DELETE` as wished. Checking all three manipulation options automatically generates a universal trigger.
- **(6) Trigger body:** The trigger body can be completed in the SQL window.

3. A trigger can also be created in the Table Editor or View Editor, on the *Triggers* page by selecting the desired `BEFORE/AFTER` operation and using the mouse right-click menu item *New Trigger*. This opens the *New Trigger Editor* shown above.

4. Or in the Field Editor on the Autoincrement page. For example, a trigger text for a new generator can be simply and quickly created using the *Edit Field / Autoinc*, *Create Generator* and then *Create Trigger*.



For those preferring direct SQL input, the `CREATE TRIGGER` statement has the following syntax:

```
CREATE TRIGGER <trigger_name>
FOR <table_name>
<keywords_for_trigger_type>
AS
<local_variable_declarations>
BEGIN
<body_of_trigger>
END
```

The trigger name needs to be unique within the database, and follow the InterBase/Firebird naming conventions used for columns, tables, views and procedures.

Triggers can only be defined for a single database, table or updatable view. Triggers that should apply to multiple tables need to be called using a stored procedure. This can be done simply by creating a stored procedure which refers to the trigger. Please refer to the *Using procedures to create and drop triggers* chapter in the *Firebird Development using IBExpert* documentation.

Triggers fire when a row-based operation takes place on the named table or view.

Local variable declarations

Triggers use the same extensions to SQL that InterBase/Firebird provides for stored procedures. Therefore, the following statements are also valid for triggers:

- `DECLARE VARIABLE`
- `BEGIN … END`
- `SELECT … INTO : variable_list`
- `Variable = Expression`
- `/* comments */`
- `EXECUTE PROCEDURE`
- `FOR select DO …`
- `IF condition THEN … ELSE …`
- `WHILE condition DO …`

As with stored procedures, the `CREATE TRIGGER` statement includes SQL statements that are conceptually nested inside this statement. In order for InterBase/Firebird to correctly parse and interpret a trigger, the database software needs a way to terminate the `CREATE TRIGGER` that is different from the way the statements inside the `CREATE TRIGGER` are terminated. This can be done using the `SET TERM` statement.

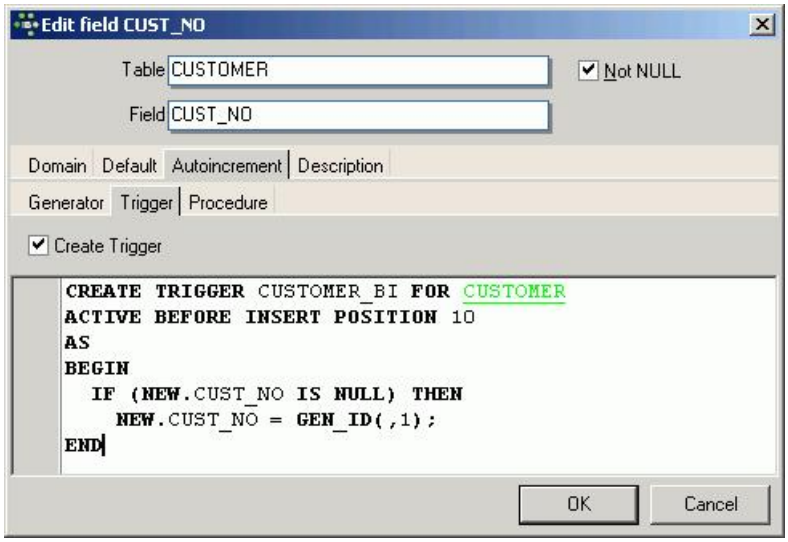
Since IBExpert version 2005.03.12 there is added support for following Firebird 2 features:

- `DECLARE <cursor_name> CURSOR FOR ...`
- `OPEN <cursor_name>`
- `FETCH <cursor_name> INTO ...`
- `CLOSE <cursor_name>`
- `LEAVE <label>`
- `NEXT VALUE FOR <generator>`

Don't forget to finally compile the new trigger using the respective toolbar icon or [F9], and, if desired, autogrant privileges, again using the respective toolbar icon or key combination [Ctrl + F8].

Create a trigger for a generator

Generally a generator is used to determine unique identification numbers for primary keys. A `BEFORE INSERT` trigger can be defined for this to generate a new ID, increasing the current value using the `GEN_ID()` function, and automatically entering it in the respective table field.



The above illustrates the Field Editor, started from the Table Editor.

Create a trigger for a view

It is possible to create a trigger for a view directly in the View Editor on the Trigger page. This is particularly interesting for read-only views. For example, `BEFORE INSERT`, `insert into Table1 new_fields` and `table2 new_data` for fields. `BEFORE UPDATES` and `BEFORE DELETE` triggers should also be added, in order to distribute the data manipulation made in the view into the respective base tables.

Trigger Editor

The Trigger Editor can be started using the IBExpert Database menu item, *New Trigger*; from the DB Explorer, using the right mouse-click menu or double-clicking on an existing trigger, or alternatively directly from the View or Triggers page.

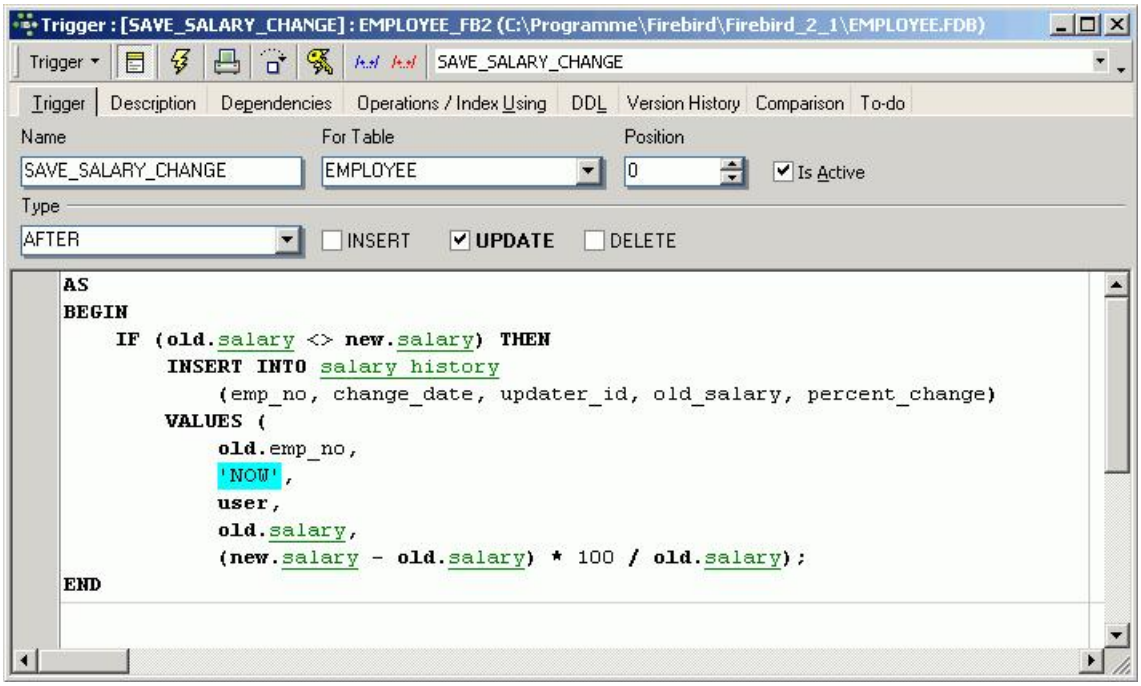
Please refer to New Trigger when creating a trigger for the first time.

The Trigger Editor has its own toolbar (see Trigger Editor toolbar) and offers the following options:

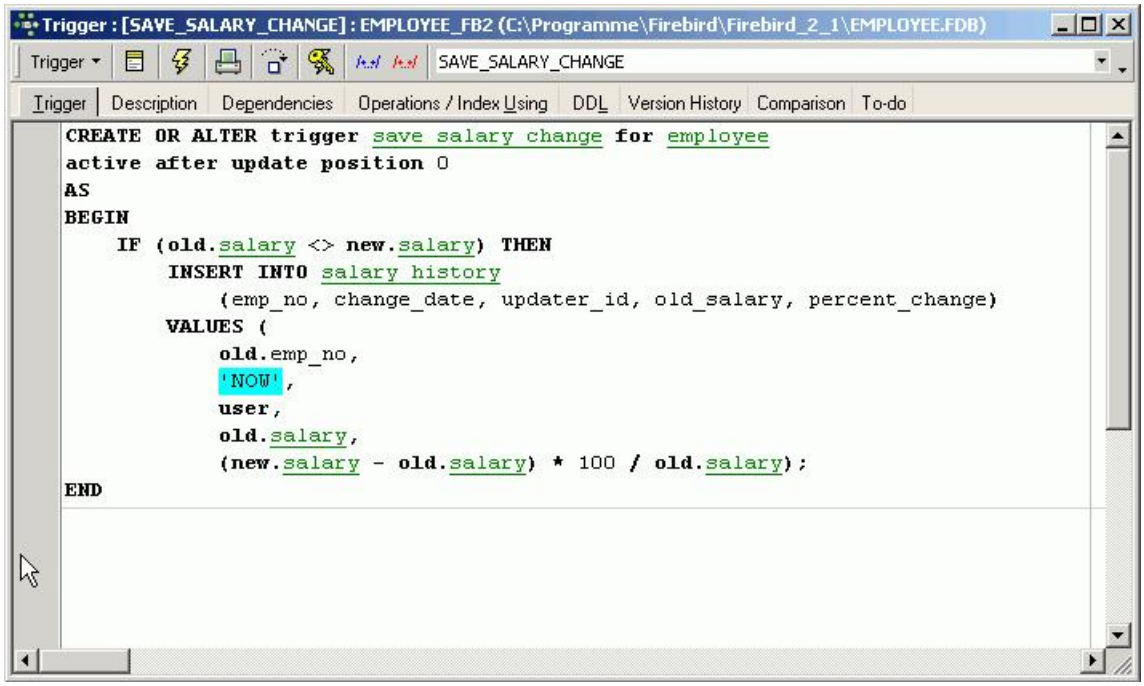
- Trigger
- Description
- Dependencies
- Operations/Index Using
- DDL
- Version History
- Comparison
- To-Do

Trigger page

The Trigger Editor's first page allows the trigger name, table or view name, position, active/inactive, and trigger type to be specified simply and quickly, with the aid of pull-down lists, provided the lazy mode has been switched on:



If this is switched off, all information needs to be specified in the SQL window:



The SQL window provides a template for both standard (for the whole trigger) and lazy mode, where the trigger body can be input. These templates can be altered if wished, using the IBExpert menu item Options / General Templates / New Trigger.

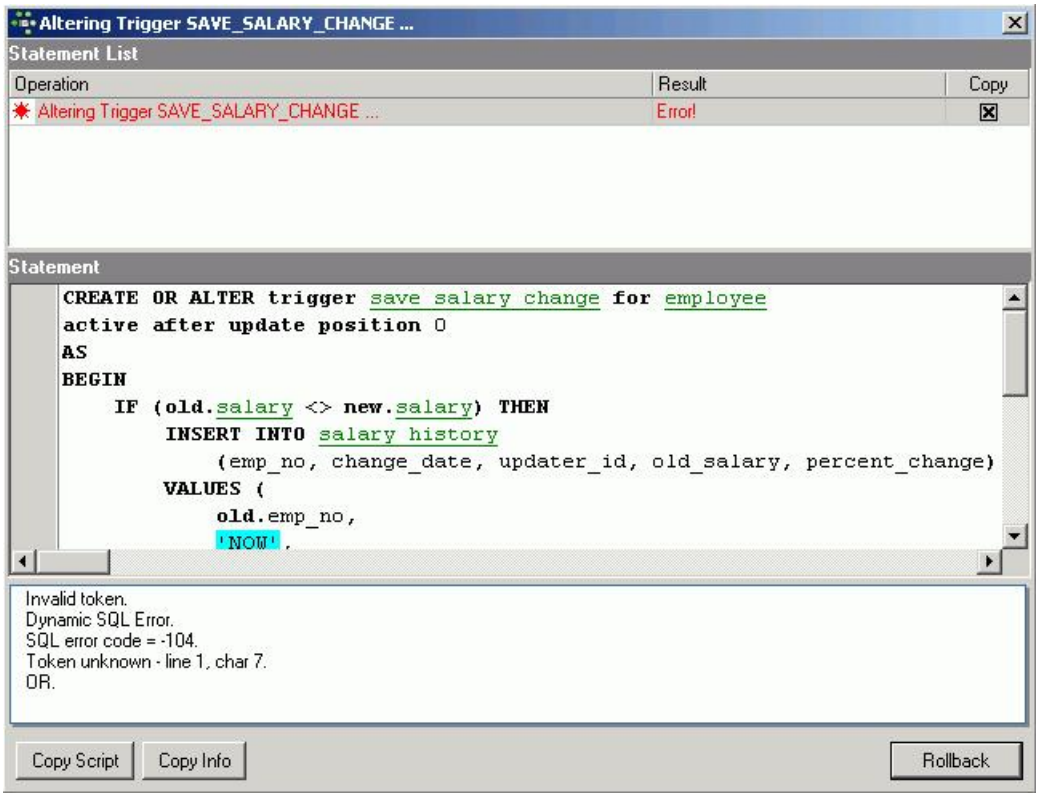
As with all SQL input windows, the SQL Editor Menu can be called using the right mouse button. The keyboard shortcuts available in the SQL Editor are also available here. These options may be used to perform a number of actions, for example:

- Comment or Uncomment code using the right-click context-sensitive menu.
- indent a marked block of code with [Ctrl + Shift + I] and move back with [Ctrl + Shift + U].

Since IBExpert version 2005.04.24 the Debugger also supports the new Firebird 2.0 feature: `SELECT ... FROM (SELECT ...)`

Since IBExpert version 2005.12.04 the Code Completion list now displays cursor names when one is declared within procedure or trigger (Firebird 2).

When the trigger or trigger alterations are complete, it can be compiled using the respective icon or [Ctrl + F9]. If errors are found, click *YES* when the *Compile Anyway* query appears, to produce an SQL error script (below the trigger text), to detect the error source.



If the problem is more complicated, the options *Copy Script* or *Copy Info* can be used before finally rolling back the trigger.

The Trigger Editor also has its own *Debug Trigger* icon. For more information regarding this, please refer to Debug Procedure or Trigger.

Description

Please refer to Table Editor / Description.

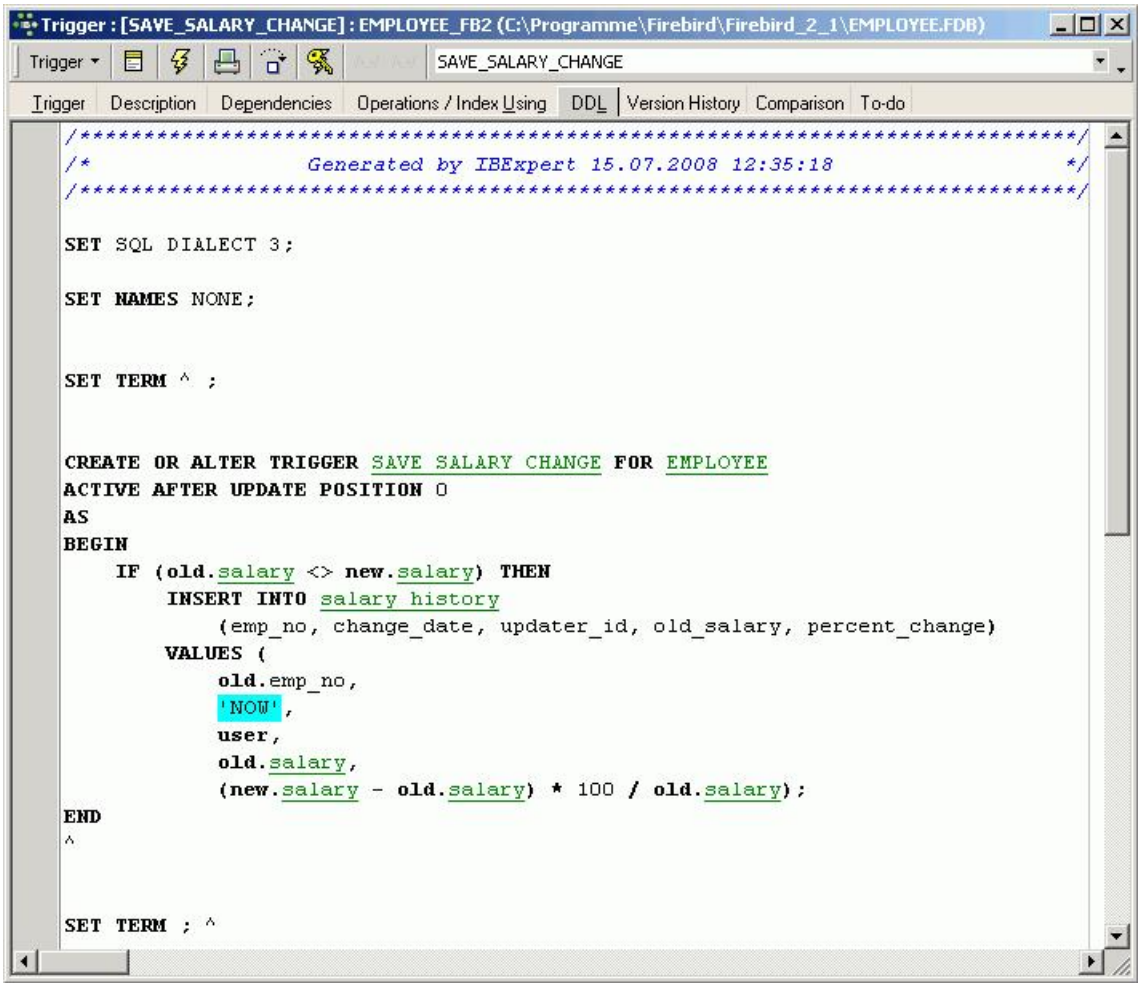
Dependencies

Please refer to Table Editor / Dependencies.

Operations/Index Using

Please refer to Procedure Editor / Operations / Index Using.

DDL



Please refer to Table Editor / DDL.

Version History

Please refer to View Editor / Version History.

Comparison

Please refer to Table Editor / Comparison.

To-do

Please refer to Table Editor / To-do.

Comment Trigger Body/Uncomment Trigger Body

It certain situations it may be necessary to disable certain commands or parts of trigger code. It is possible to do this temporarily, without it being necessary to delete these commands. Simply select the rows concerned in the SQL Editor, and select either the editor toolbar icons:

the right mouse button menu item, *Comment Selected*, or key combination [Ctrl + Alt + .]. This alters command rows to comments. The commented text can be reinstated as SQL text by using *Uncomment Trigger Body* icon (above), the right mouse button menu item *Uncomment Selected*, or [Ctrl+ Alt + .].

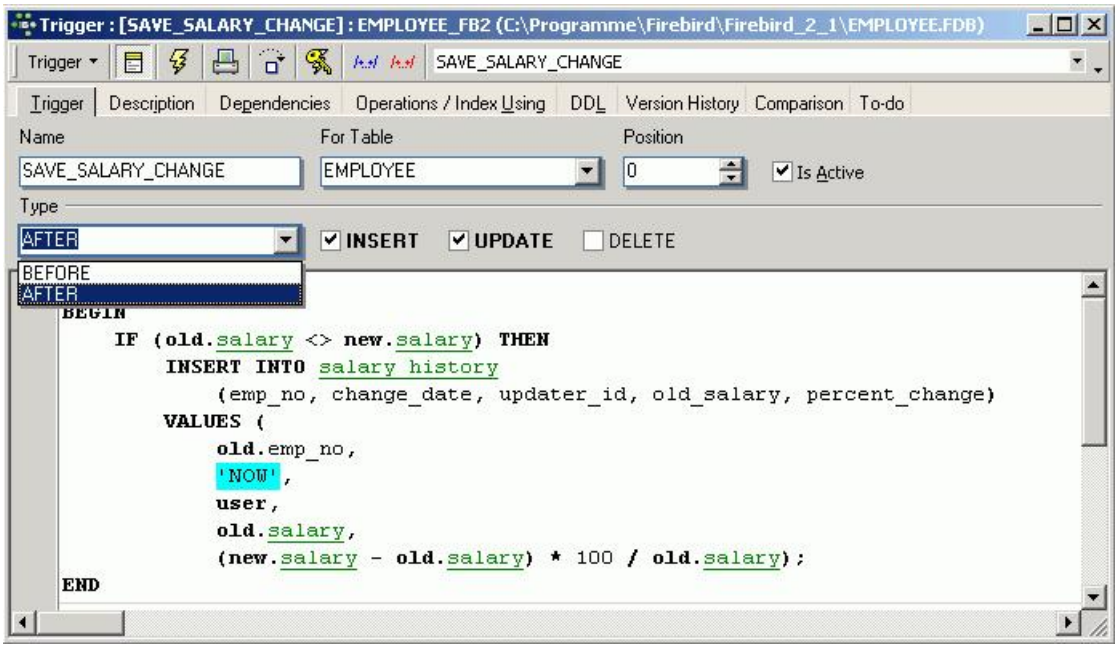
It can not only be used to add comments and documentary notes to more complex stored procedures and triggers; but also to factor out selected parts of code during the testing phase, or even for customer applications, where certain features are not currently needed but may be required at a future date. The code can be reinstated by simply uncommenting as and when required.

Alter trigger

Both the trigger header and the trigger body may be altered. The trigger header may be activated or deactivated, or its position changed (in relation to other triggers).

If the trigger body needs to be altered, there is no need to make any alterations to the header, unless you wish to of course! Although in this case, it would probably make more sense to drop the trigger and create a new one. Any amendments to the trigger body override the original contents.

Triggers can easily be altered in the DB Explorer's Trigger Editor, opened either by double-clicking on the trigger name, or right-clicking and selecting *Edit Trigger* [Ctrl + O]. The header information can be changed as wished using the pull-down lists to alter position, active/non-active and type:



(Image shows lazy mode). The body text may be altered in the SQL panel as wished.

Finally the revised trigger needs to be compiled and committed, for the alterations to become effective.

The SQL syntax for alterations to the trigger header is as follows:

```
ALTER TRIGGER <trigger_name> INACTIVE | ACTIVE

ALTER TRIGGER <trigger_name> POSITION n
```

where `n` is the new position number. Or to alter the trigger body:

```
ALTER TRIGGER <trigger_name>
AS
BEGIN
   <new_trigger_body>
END
```

A trigger can only be altered by the database owner or by the SYSDBA.

Recreate trigger

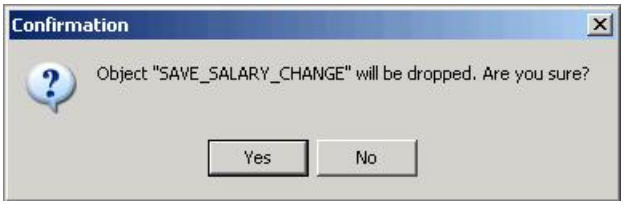
New to Firebird 2.0: The DDL statement `RECREATE TRIGGER` is now available in DDL. Semantics are the same as for other `RECREATE` statements.

See also:
`RECREATE TRIGGER`

Drop trigger/delete trigger

A trigger can only be dropped if other users are not performing any changes to any tables which may relate to the specified trigger, at the time of deletion. In IBExpert, a trigger can be dropped from the DB Explorer by selecting the trigger to be deleted and using the right-click menu item *Drop Trigger* or [Ctrl + Del].

IBExpert asks for confirmation



before finally dropping.

For those preferring to use SQL, the syntax is as follows:

```
DROP TRIGGER <trigger_name>
```

An alternative solution to dropping triggers is to alter them to the `INACTIVE` status. That way they are left in the database, but disabled from firing, just in case they might be needed after all at a later date.

A trigger can only be dropped by the database owner or the SYSDBA.

See also:
Stored Procedure and Trigger Language
Writing stored procedures and triggers
Using procedures to create and drop triggers
Comments
Lazy Mode
Generator
View
Debug Procedure
Firebird for the database expert - Episode 1: Indexes
Dependencies Viewer
Stored Procedure/Triggers/Views Analyzer
`IBE$VERSION_HISTORY` system table

Generator (FB2: Sequence)

Generators are automatic sequential counters, spanning the whole database. They are necessary because all operations in InterBase/Firebird are subject to transaction control.

A generator is a database object and is part of the database's metadata. It is a sequential number, incorporating a whole-numbered 64 bit value integer since InterBase 6/Firebird (in earlier versions a 32 bit value integer), that can automatically be inserted into a column. It is often used to ensure a unique value in an internal primary key.

Generators are the only transaction-independent part of InterBase/Firebird. For each operation a new number is generated, regardless whether this transaction is ultimately committed or rolled back (this consequently leads to "missing numbers"). Therefore generators are best suited for automatic internal sequential numbering for internal primary keys.

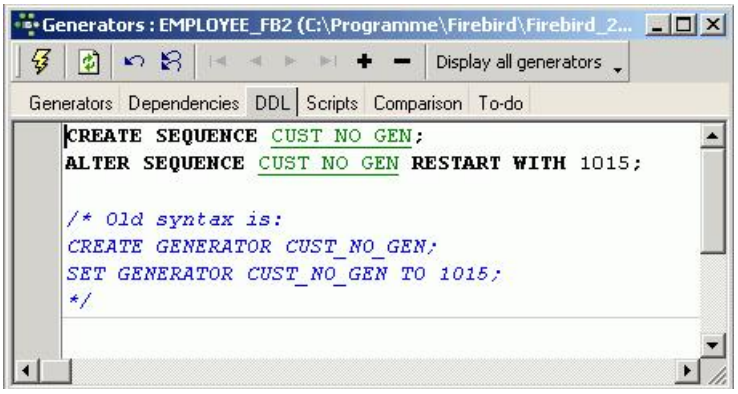
`SEQUENCE` was introduced in Firebird 2.0. It is the SQL-99-compliant synonym for `GENERATOR`. `SEQUENCE` is a syntax term described in the SQL specification, whereas `GENERATOR` is a legacy InterBase syntax term.

It is recommended Firebird 2.0 users use the standard `SEQUENCE` syntax:

- `CREATE SEQUENCE`
- `NEXT VALUE FOR`
- `ALTER SEQUENCE`
- `DROP SEQUENCE`

A sequence generator is a mechanism for generating successive exact numeric values, one at a time. A sequence generator is a named schema object. In dialect 3 it is a `BIGINT`, in dialect 1 it is an `INTEGER`. It is often used to implement guaranteed unique IDs for records, to construct columns that behave like `AUTOINC` fields found in other RDBMSs. Further information regarding `SEQUENCE` can be found in the Firebird 2.0.4 Release Notes.

For legacy reasons, IBExpert will still continue to use the term `Generator` alongside the term `SEQUENCE`.



Generators can be created either directly in the SQL Editor or using the DB Explorer (refer to New Generator for details).

Generally a generator is used to determine unique identification numbers for primary keys. A trigger can be defined for this, which increases the current value using the `GEN_ID()` function, and automatically enters it in the respective table]field. Please refer to create a trigger for a generator for more information. A generator can also be called from a stored procedure or an application.

A database can contain any number of generators. Although up until the most recent InterBase version 7.x the number of generators was limited to one data page. One generator uses 8 bytes, which means approximately 115 generators fit onto one page (at 1K). This limitation has been solved in the InterBase 7.x version.

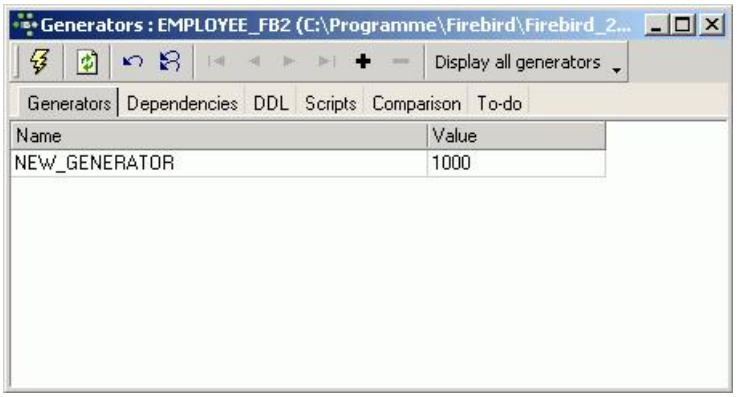
The current generator value of existing generators is not stored in a table but on its own system data pages, as the table contents are subject to transactional changes. The generator value is also secured when backing up.

Generators are database objects and are part of the database's metadata, and can be created, modified and dropped as all other InterBase/Firebird objects in the IBExpert DB Explorer.

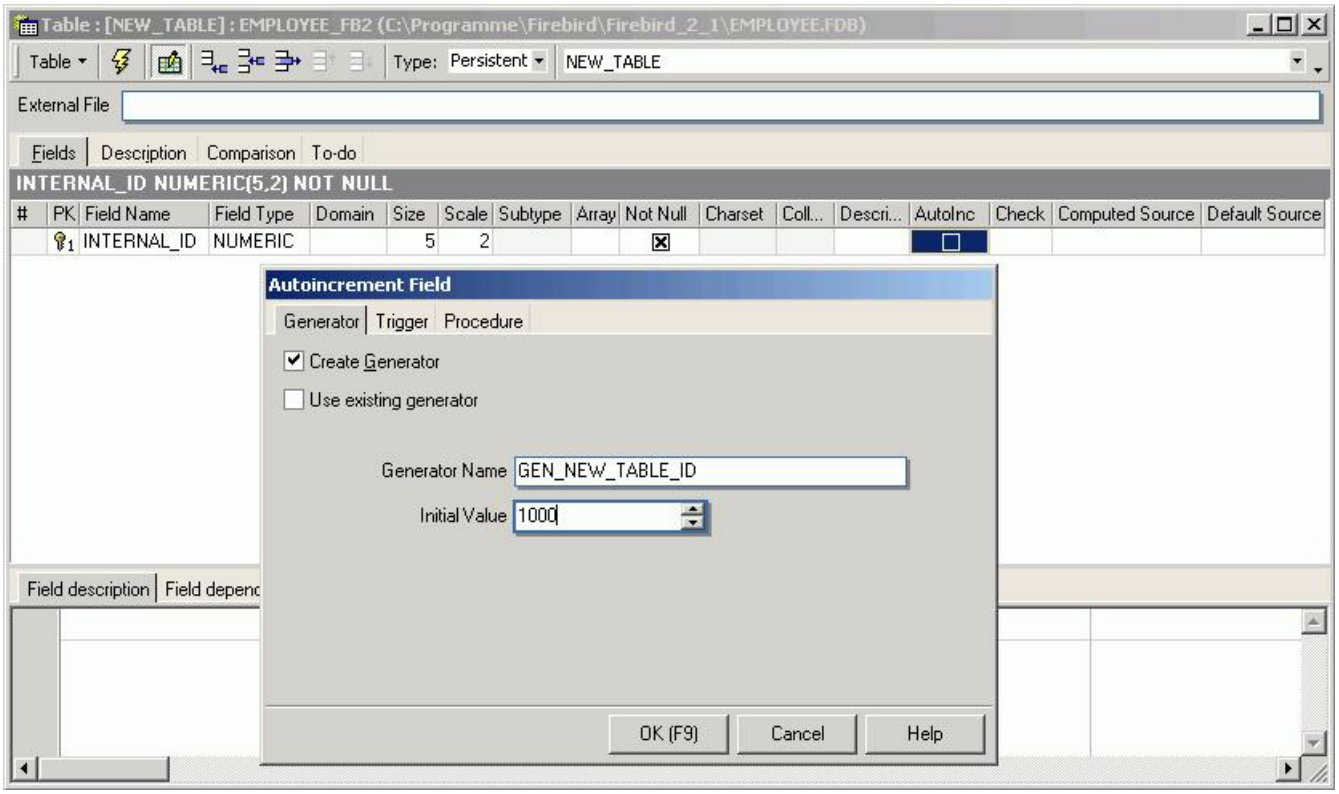
New generator

A new generator can be created in a connected database in a number of ways:

1. By using the menu item Database / New Generator, the respective icon in the New Database Object toolbar, or using the DB Explorer right mouse button (or key combination [Ctrl + N]), when the generator heading of the relevant connected database is highlighted, to start the *New Generator Editor*:



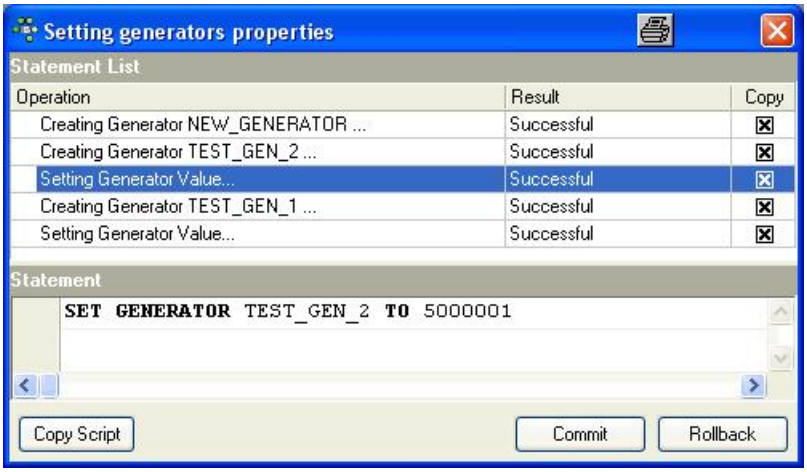
2. Alternatively, a new generator can be created in the DB Explorer on the Fields page by double-clicking (or using the space bar when inserting a new field) to check the *Autoinc* box:



3. Or in the under Autoincrement (started by double-clicking on an existing `INTEGER` or `SMALLINT` field in the Table Editor).

4. Or directly in the IBExpert SQL Editor, and then saved as a generator.

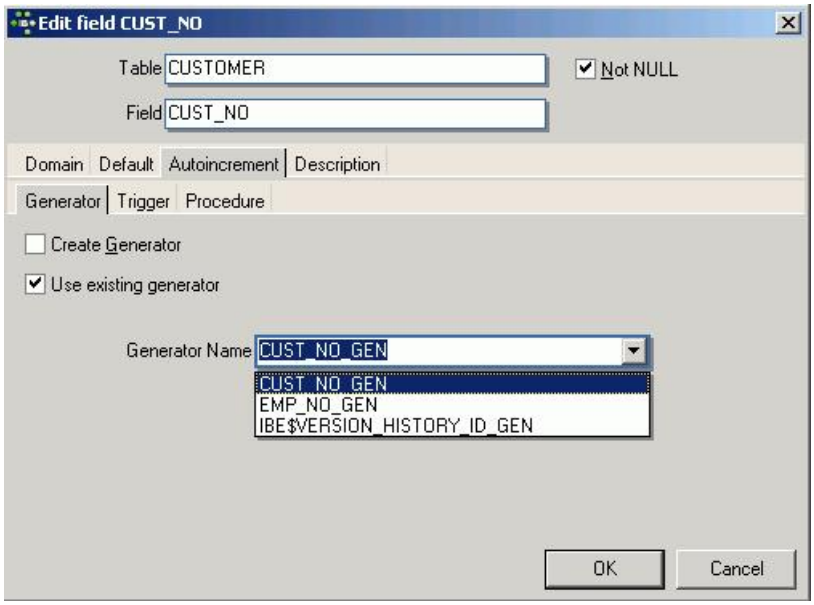
Using the the new generator name simply needs to be specified along with the initial generator value. Several generators can be created in the Generator Editor and compiled simultaneously:



Using the *Display all Generators* button on the Generator Editor toolbar, all generators for the database can be listed and an existing generator selected. (For internal numbering purposes, the same generator may be used on several fields, for example all internal primary key IDs, within the database.)

Using the *Autoinc* page in the Table and Field Editors, the *Create Generator* box simply needs to be checked, and the name and starting value defined.

It is also possible to select an existing generator for the specified field here (simply click *Use Existing Generator* and select from the pull-down list):



For those preferring direct SQL input, the syntax is as follows:

```
CREATE GENERATOR <Generator_Name>;
```

This statement also sets the initial generator value to zero. To establish a different starting value, use the `SET GENERATOR` statement, for example:

```
SET GENERATOR <Generator_Name> TO n;
```

where `n` is the initial generator value. `SET GENERATOR` can also be used to reset an existing generator's value. This however requires care, as usually the column(s) that receives the generator value is/are defined to be unique. For example, you would not normally reset customer IDs except under unusual and controlled circumstances.

To increment the generator use the `STEP_VALUE` parameter (can be positive or negative):

```
GEN_ID(<Generator_Name>, STEP_VALUE)
```

If this parameter is not used, the default `STEP_VALUE` with an increment of 1 applies.

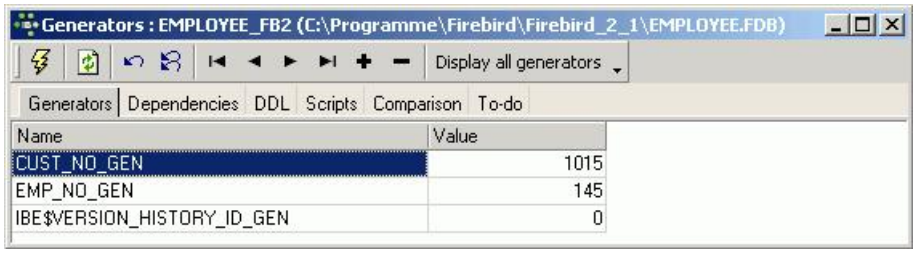
Generator Editor

The Generator Editor can be started using the Database / *New Generator* menu item; from the DB Explorer, using the right mouse-click menu or double-clicking on an existing generator; or directly from the Field or Table Editor / *Autoincrement*.

Please refer to New Generator when creating a generator for the first time.

The Generator Editor has its own toolbar (see Generator Editor toolbar) and offers the following options:

- Generators page
- Dependencies
- DDL
- Scripts
- Comparison
- To-Do



Generators page

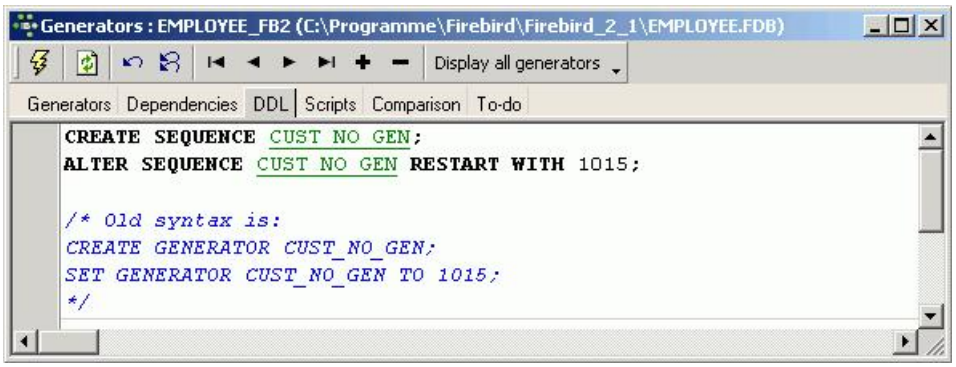
Here it is possible to create new generators, select an existing generator, and alter a generator. Please refer to New Generator or Alter Generator for details.

Dependencies

Please refer to Table Editor / Dependencies.

DDL

Please refer to Table Editor / DDL.



Scripts

Creating - displays the `CREATE GENERATOR` statement for the generator selected on the Generators page. If all generators are displayed on the *Generator* page (*Display All Generators* button), all corresponding `CREATE` statements appear on this page.

Setting Values - displays the `SET GENERATOR` statement for the generator selected on the *Generators* page. Again, if all generators are displayed on the *Generator* page (*Display All Generators* button), all `SET` statements appear on this page.

Full - displays the full SQL text for the generator selected on the *Generators* page (or all generators).

Please note that the *Scripts* page is for display only. It is not possible to make any amendments on this page.

Comparison

Please refer to Table Editor / Comparison.

To-Do

Please refer to Table Editor / To-Do.

Alter generator

A generator may be altered to specify a new value. The value of a generator can be changed as often as wished.

This can be performed in IBExpert using the DB Explorer's Generator Editor, opened either by double-clicking on the generator name, or right-clicking and selecting *Edit Generator* [Ctrl + O]. Simply enter the new figure in the *Value* column, compile and commit.

The SQL syntax for altering a generator is as follows:

```
SET GENERATOR <generator_name> TO n
```

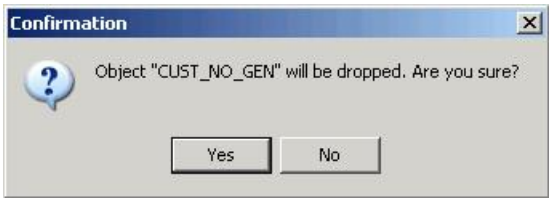
where `n` is the new value. This new value is immediately effective.

Please refer to the `SET GENERATOR` statement for further information.

Drop generator/delete generator

In IBExpert, a generator can be dropped from the DB Explorer by selecting the generator to be deleted and using the '-' icon on the Generator Editor toolbar or [Shift + Del].

IBExpert asks for confirmation and displays the SQL statement:



before finally dropping when the statement is committed.

For those preferring to use SQL, the syntax is as follows:

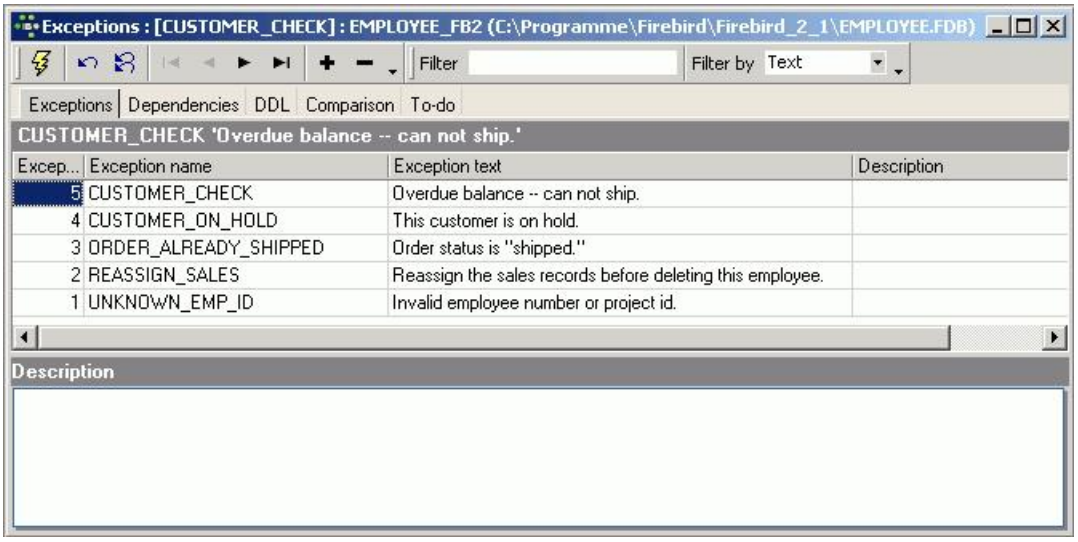
```
DROP GENERATOR <generator_name>;
```

See also:
CREATE SEQUENCE
FB 2.0.4. Release Notes: CREATE SEQUENCE
Firebird for the database expert - Episode 2: Page Types
SET GENERATOR Create a trigger for a generator

Exception

Exceptions are user-defined named error messages, written specifically for a database and stored in that database for use in stored procedures and triggers.

If it is ascertained in a trigger that the value in a table is incorrect, the exception is fired. This leads to a rollback of the total transaction that the client application is attempting to commit. Exceptions can be interleaved.

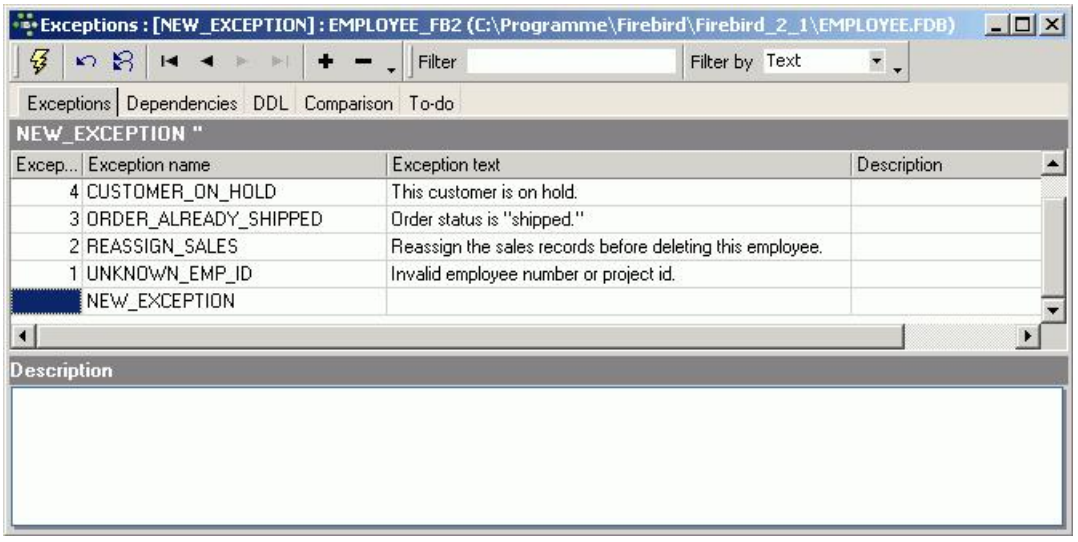


They can be shared among the different modules of an application, and even among different applications sharing a database. They provide a simple way to standardize the handling of preprogrammed input errors. Exceptions are typically used to implement program logic, for example, you do not wish a user to sell an item in stock, which has already been reserved by another user for their customer.

Exceptions are database objects and are part of the database's metadata, and can be created, modified and dropped as all other InterBase/Firebird objects in the IBExpert DB Explorer.

New exception/Exception Editor

A new exception can be created in a connected database either by using the menu item Database / New Exception, the respective icon in the New Database Object toolbar, or using the DB Explorer right-click menu (or key combination [Ctrl + N]), when the exception heading of the relevant connected database is highlighted. A *NewException* dialog appears, with its own toolbar:



Alternatively, a new exception can be created directly in the IBExpert SQL Editor, using the following statement:

```
CREATE EXCEPTION <Exception_Name>
"Exception_Text";
```

The *Exception Editor* can be opened directly from the DB Explorer by double-clicking on any existing exception name. It can also be started directly from any procedure or trigger containing an exception, simply by double-clicking on the exception name in the SQL text on the Procedure Editor's Edit page, or the Trigger Editor's Triggers page.

Exceptions page

The new exception name can be added to the list displaying all exceptions for the active database, and the exception text message entered. Please be careful when using special characters! Especially when using older versions of InterBase, it is preferable to abstain from using any special characters. With the newer versions, there should not be any problems, provided the correct character set has been specified. The exception ID is automatically assigned by the database, when the exception is committed.

After creating the exception, it then needs to be incorporated into a stored procedure or a trigger, to determine under what conditions and when the exception is to appear. Please refer to Raising an Exception for details.

Dependencies

Please refer to Table Editor / Dependencies.

DDL

Please refer to Table Editor / DDL.

Comparison

Please refer to Table Editor / Comparison.

To-Do

Please refer to Table Editor / To-Do.

Raising an exception

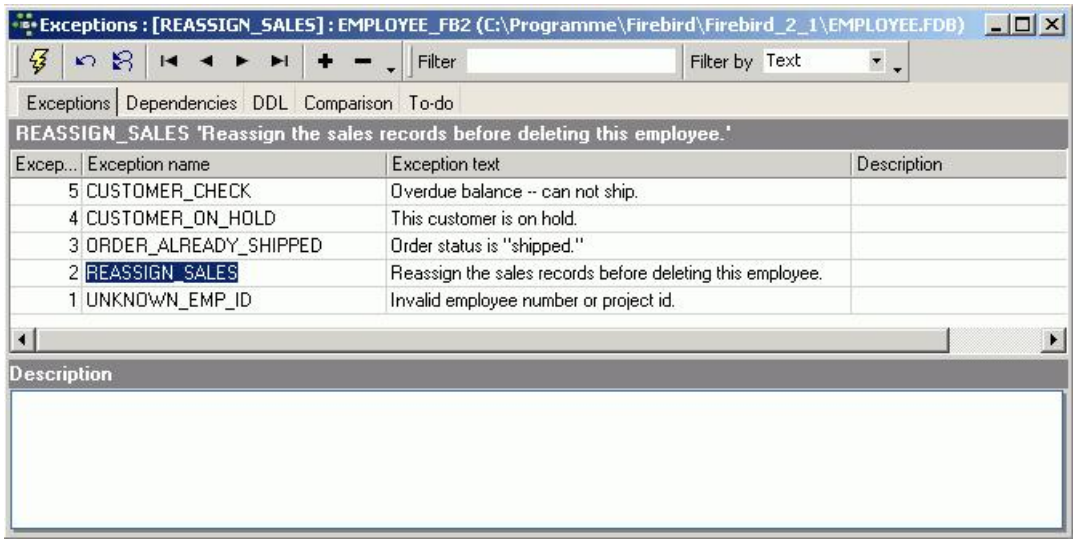
The `EXCEPTION` statement is used to notify a calling application of an exception. The calling application can be a trigger, a stored procedure, or another program. To raise an exception in a trigger or stored procedure use the `EXCEPTION` keyword:

```
EXCEPTION <Exception_Name>;
```

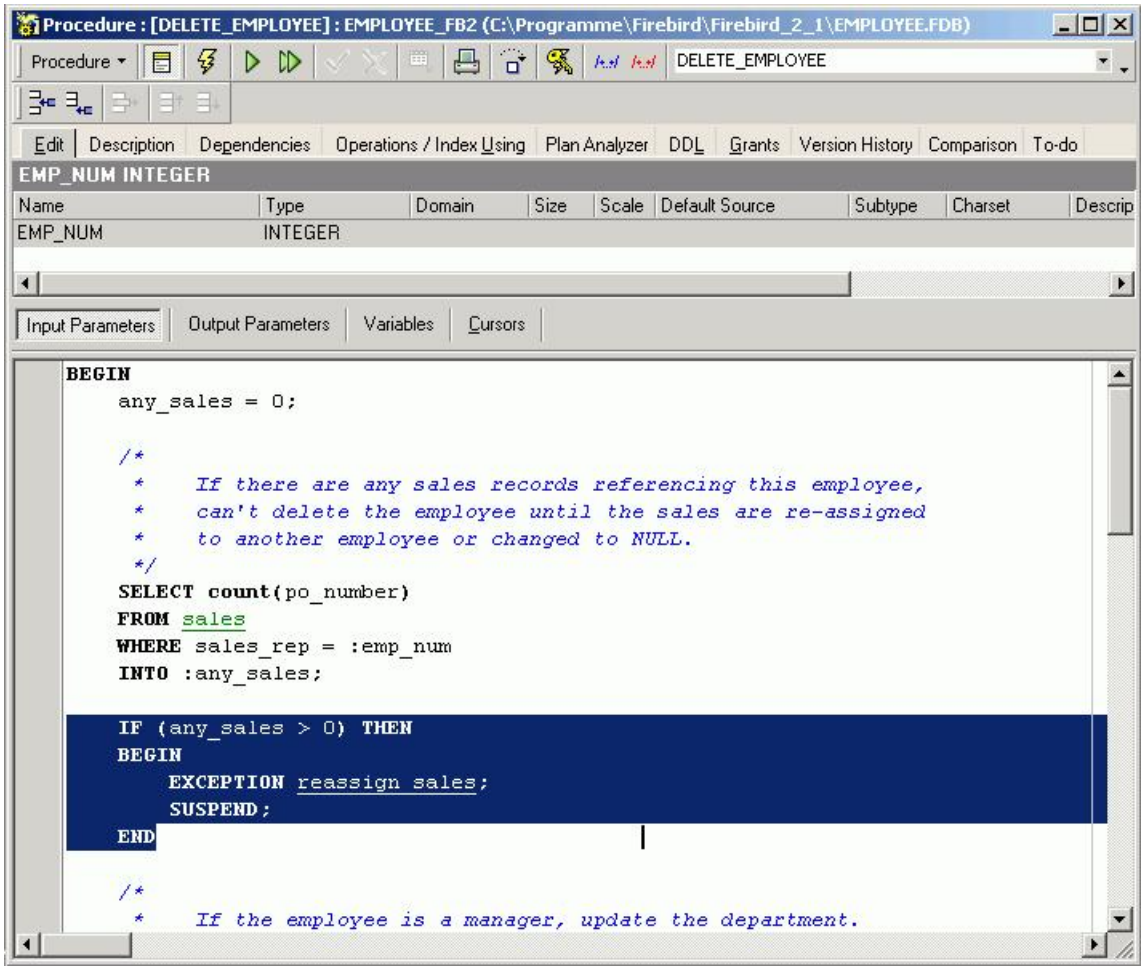
When an exception is raised, the following takes place:

1. The exception terminates the trigger or procedure.
2. Any statements in the trigger or stored procedure that follow the `EXCEPTION` statement are not executed. In the case of a `BEFORE` trigger the update that fired the trigger is aborted.
3. The trigger or procedure returns an error message to the calling application.

An example of an exception raised in a procedure can be found in the `EMPLOYEE` database. The exception `REASSIGN_SALES` was first created:



and then incorporated into the DELETE_EMPLOYEE procedure:



Alter exception

Exceptions can be altered directly in the *Exceptions Editor*, started by double-clicking directly on the exception name in the DB Explorer. Alternatively use the DB Explorer's right mouse-click menu item *Edit Exception* or key combination [Ctrl + O].

The Exception Editor appears, where changes to the exception name and exception text can be made as wished. Changes to exception texts may be made even if other objects depend on them, however not the exception name.

The SQL syntax is:

```
ALTER EXCEPTION <exception_name>
'New Exception Text';
```

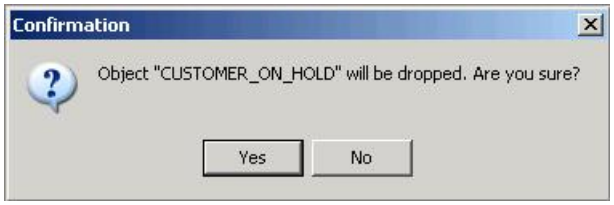
An exception can only be altered by the original creator or by the SYSDBA user.

A number of new syntaxes for changing exceptions was introducted in Firebird 2.0. Please refer to Firebird 2.0.4 Release Notes: New syntaxes for changing exceptions for further information.

Drop exception/delete exception

An exception may not be dropped if it is used by other procedures or triggers, until the dependency is removed. Any such dependencies are listed on the Exception Editor / Dependencies page, where they can be directly removed, if wished.

To drop an exception use the DB Explorer right mouse-click menu item *Drop Exception...* or [Ctrl + Del]. IBExpert asks for confirmation:



before finally dropping the exception. Once dropped, it cannot be retrieved.

Using SQL the syntax is:

```
DROP EXCEPTION <exception_name>;
```

An exception can only be dropped by its creator, the database owner or the SYSDBA.

See also:
Stored Procedure
Trigger
Stored procedure and trigger language
Dependencies Viewer

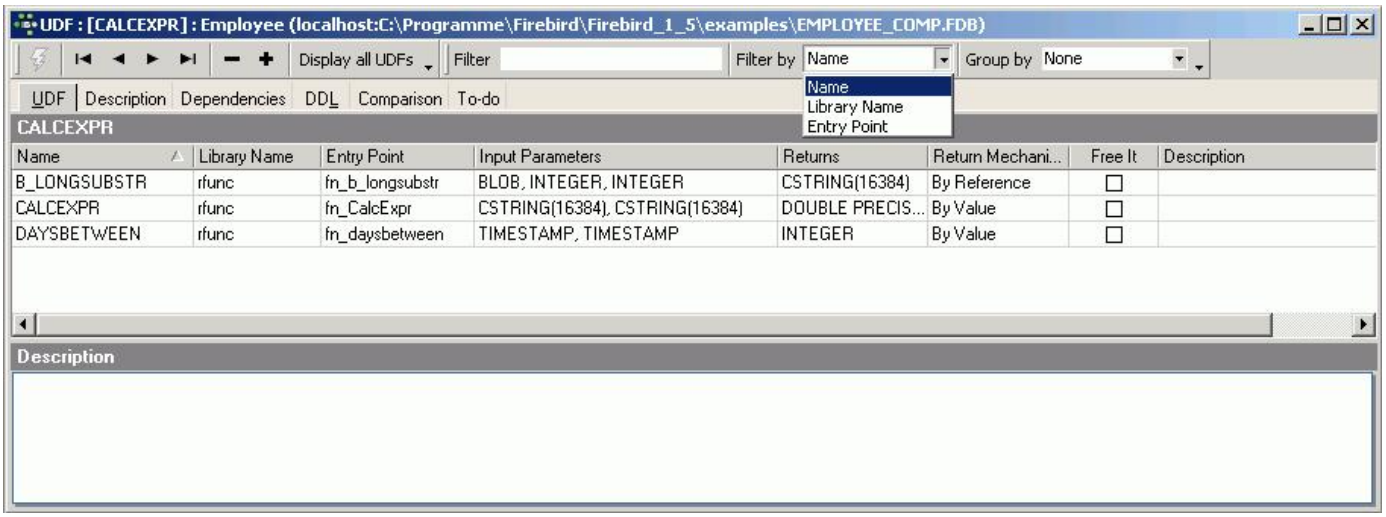
User-defined function

A user-defined function (UDF) is used to perform tasks that Firebird/InterBase can't. It can be described as an external database function written entirely in another language, such as C++ or Pascal, to perform data manipulation tasks not directly supported by InterBase/Firebird.

UDFs can be called from InterBase/Firebird and executed on the server. These functions can exist on their own or be collected into libraries. UDFs offer the possibility to create your own functions (such as `SUBSTR`) and integrate them in the database itself. Each UDF is arranged as a function, belonging to a `DLL` (Linux: `.so`). Thus one dynamically loaded library consists of at least one function.

UDFs can be incorporated into the database using the IBExpert DB Explorer, IBExpert SQL Editor, or IBExpert Script Executive.

UDF Editor



The IBExpert UDF Editor displays those UDFs inserted into the list, by double-clicking on the UDF name in the DB Explorer, or alternatively using the navigation icons in the editor toolbar to insert single or all UDFs. The grid display can also be filtered or grouped if wished. The grid displays key information, including *name*, *library*, *entry point*, *input parameters*, *returns*, *return mechanism* (pull-down list of options), whether *freed* (checkbox), and *description*. IBExpert version 2006.06.05 introduced support for the Firebird 2.0 `NULL` clause. Further information is displayed on the *Description*, *Dependencies*, *DDL*, *Comparison* and *To-Do* pages.

UDF definitions are database dependent and not server dependent, i.e. they need to be registered for each database individually. Since InterBase 6/Firebird, the libraries need to be stored in the InterBase/Firebird UDF folder. This is not critical when working with older InterBase versions.

Please refer to the `DECLARE EXTERNAL FUNCTION` statement for details of incorporating UDFs in InterBase/Firebird.

It is important to note that the majority of UDFs, when used in a `WHERE` condition, prevent indices being used during execution.

New to Firebird 2.0: The following is a summary of the major changes, the details of which can be found in the Firebird 2.0.4 Release Notes in the External functions (UDFs) chapter:

- Ability to signal SQL `NULL` via a null pointer
- UDF library diagnostic messages improved
- UDFs added and changed
  - `IB_UDF_rand()` VS `IB_UDF_srand()`
  - `IB_UDF_lower`
- General UDF changes
  - Build changes

An ideal example of a UDF library is RFunc (written in C++) containing over 80 UDFs (although some of these are only applicable for older InterBase versions or for different SQL dialects). It is available for both Windows and Linux platforms in English and Russian and can be downloaded free of charge from http://www.ibexpert.com/download/udf/. FreeUDFLib is an example of a UDF library written in Delphi, and can also be downloaded from this link.

For further functions please refer to IBEBlock Functions and the Firebird documentation: *Firebird built-in Functions*.

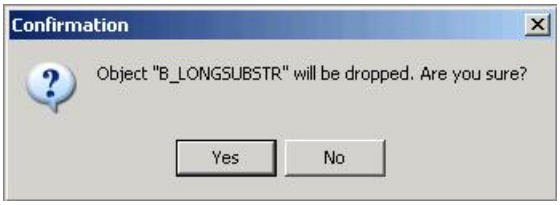
Drop external function/drop UDF

The `DROP EXTERNAL FUNCTION` command removes the declaration of the UDF, specified by an additional parameter, from the database.

The dropped function can no longer be reached by the database, as the relevant reference to the UDF library is deleted. However the UDF still exists in the UDF library, so that it can still be used by other databases.

In IBExpert, a UDF can be dropped from the DB Explorer by selecting the UDF to be deleted and using the right-click menu item Drop UDF or [Ctrl + Del].

IBExpert asks for confirmation



before finally dropping.

The SQL syntax is:

```
DROP EXTERNAL FUNCTION <external_function_name>
```

The declaration of a UDF can only be dropped by the database owner or the SYSDBA.

RFunc

RFunc is a UDF library containing over 80 UDFs (although some of these are only applicable for older InterBase versions or for different SQL dialects). It is available for both Windows and Linux platforms in English and Russian. It can be downloaded free of charge from http://www.ibexpert.com/download/udf/. The most up-to-date version of this library can found at http://rfunc.sourceforge.net/.

It represents a set of user's (UDF) string, bit, numerical functions, and can also be used for operations with `DATES` and `TIME` and blobs. Also contains `PARSER`, i.e. calculator of expressions.

InterBase 4.2, 5.x, 6.x, 7.0 (Windows 9x, NT, 2000) and InterBase 5.x, 6.x, 7.0 (Linux) or Firebird are supported. The library is written in C++ and is delivered with source codes.

RFunc installation

The ZIP-file should be selected (Windows or Linux; English or Russian) and downloaded.

Windows installation

1. The `RFUNC.DLL` file needs to be copied into a folder:

- Variant 1: `<IB path>IB_path\bin` (for IB6: `IB_path\UDF`), where `IB_path` is the path to a folder, in which InterBase/Firebird is installed (recommended).
- Variant 2: `Windows\System` (for Windows 9x) or `WinNT\System32` (Windows NT, 2k).

2. only for IB 5.x: `copy ib_util.dll` file from `<IB path>\Lib` to `\Bin`.

If several versions of InterBase servers are installed on one computer, it is necessary to use the RFunc library appropriate to the installed client IB (`GDS32.DLL`).

It is recommended before starting the InterBase/Firebird server to substitute `GDS32.DLL` appropriate to the version of the server.

Linux installation

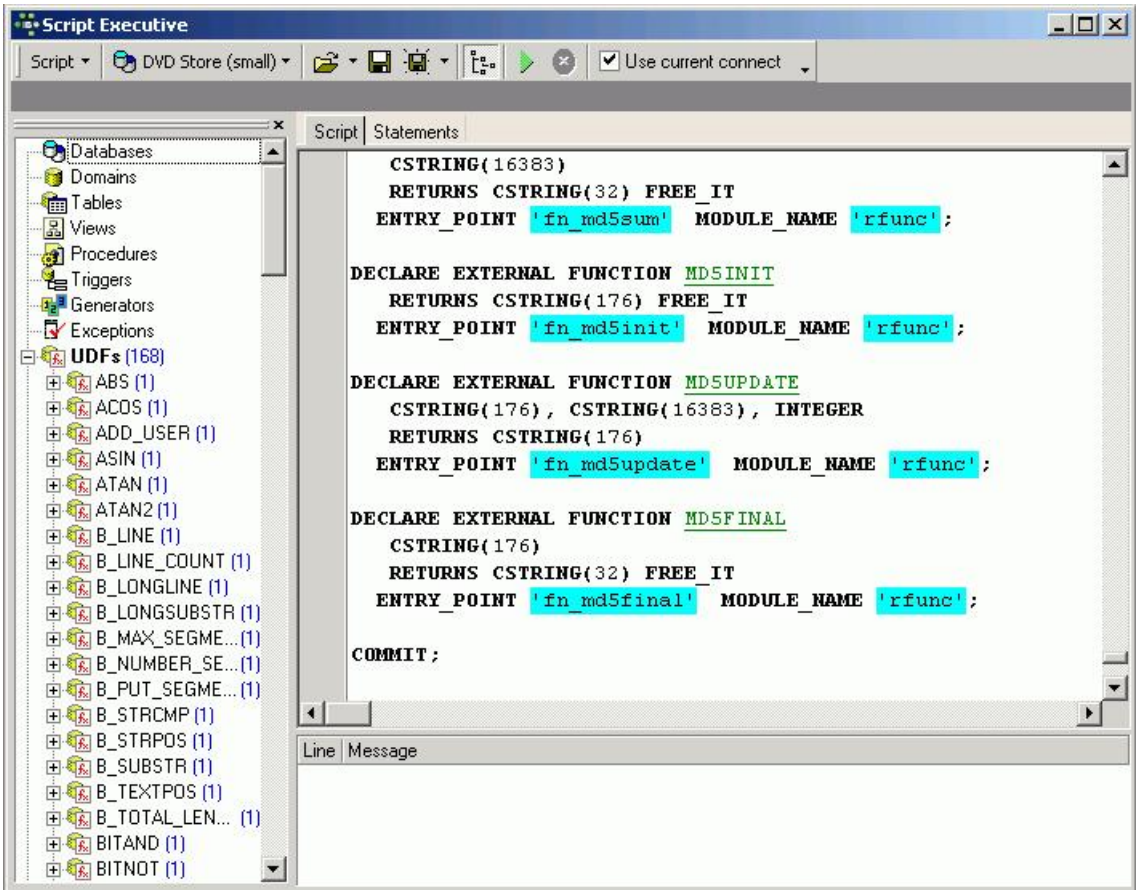
IB 5.x:

- Variant 1: Copy the RFunc file into directory `/usr/lib`.
- Variant 2: Copy the RFunc file into any directory, for example, `/home/rFunc`. Create the reference to the library by using the `\ln -s /home/rFunc/rfunc /usr/lib/rfunc\` command. The user should own the right to create references in the directory `/usr/lib`.

InterBase 6-7 und Firebird (Windows und Linux):

Copy the RFunc file into directory `\UDF`.

The `rfuncx.sql` (`x` = InterBase version; use `rfunc6.sql` for all Firebird versions) script, found in the `UDF\sql` directory, should then be copied into the IBExpert Script Executive (found in the Tools menu), and executed [F9]. A database connection must exist, as UDF libraries need to be registered for each database (i.e. they are database-dependent and not server-dependent).



It is then necessary to disconnect and reconnect to the database so that the full list of RFunc UDFs can be viewed in the DB Explorer under the DB object branch *UDF*.

FreeUDFLib

FreeUDFLib is a free UDF library (October 1998) containing many useful UDFs for use with InterBase 4.2 and 5.0 under the Win32 platforms (unfortunately no UNIX support with this). It is written entirely in Delphi and all source code is provided.

It can be downloaded free of charge from http://www.ibexpert.com/download/udf/.

Everything in this release is completely free. However, it's not a `PUBLIC DOMAIN`. Please refer to the `license.txt`, included in the `ZIP` file for more information on licensing.

FreeUDFLib installation

After unzipping `FreeUDFLib.zip`, copy `FreeUDFLib.dll` to the InterBase/Firebird `bin` or `udf` directory, for example: `C:\Program Files\InterBase Corp\InterBase\bin`, `C:\Program Files\Borland\InterBase\udf\bin` or `C:\Program Files\Firebird\udf\bin`.

The `ext_funcs.sql` script should then be copied into the IBExpert Script Executive (found in the Tools menu), and executed using [F9]. A database connection must exist, as UDF libraries need to be registered for each database (i.e. they are database-dependent and not server-dependent). If necessary, use the Script Executive menu item *Add CONNECT statement* to connect to the desired database, before executing.

It is then necessary to disconnect and reconnect to the database so that the full list of FreeUDF external functions can be viewed in the DB Explorer under the DB object branch *UDF*.

FreeAdhocUDF

The latest published version of FreeAdhocMin (a "minimal" version without source code) was released on February 9, 2007. It is available for Linux and Windows and can be downloaded at: http://www.ibexpert.com/download/udf/.

It includes several minor bug fixes, new functions and almost complete implementation of RFunc. Altogether a total of 333 functions! Full documentation of the individual functions can be found at: http://www.udf.adhoc-data.de/documentation_english/dok_eng_inhalt.html.

The FreeAdhocUDFs are based upon:

- FreeUDFLib (in Delphi, 1998 from Gregory Deatz)
- FreeUDFLibC (ported to C, 1999 from Gregory Deatz)

and are compatible to

- FreeUDFLib from AvERP (in Delphi, with some enhancements) - complete
- GrUDF (in Delphi and Kylix 2004 from Torsten Grundke and Gerd Kroll) - complete
- RFunc (in C++ from Polaris Software, last version 2003-11-27) - nearly complete

The FreeAdhocUDFs are programmed by Peter M., Georg Horn and Christoph Theuring.

The FreeAdhocUDFs return the same values in Windows and Linux. They also return the same values from InterBase 5.6 to InterBase2007 and Firebird 1.0 to Firebird 2.0.

The FreeAdhocUDFs are published under the GPL and everyone may use them, even in commercial projects (see license). The FreeAdhocUDFs are copyright adhoc dataservice GmbH, Virneburg/Eifel, Germany.

The FreeAdhocUDFs are distributed under the License on an "AS IS" basis, *WITHOUT WARRANTY OF ANY KIND*, either express or implied.

FreeAdhocUDFmin installation

The newest version of FreeAdhocMin (a "minimal" version without source code) was released on February 9, 2007. It is available for Linux and Windows and can be downloaded at: http://www.ibexpert.com/download/udf/.

Download the `ZIP` file to the hard drive. Select the file you wish to install (Windows/Linux, InterBase/Firebird). and copy to the InterBase/Firebird `bin` or `udf` directory, for example: `C:\Program Files\InterBase Corp\InterBase\bin`, `C:\Program Files\Borland\InterBase\udf\bin` or `C:\Program Files\Firebird\udf\bin`.

Copy the required SQL text (found in the `Install` directory) into the IBExpert]Script Executive (found in the Tools menu), and execute using [F9]. A database connection must exist, as UDF libraries need to be registered for each database (i.e. they are database-dependent and not server-dependent). If necessary, use the Script Executive menu item *Add CONNECT statement* to connect to the desired database, or click the *Use current connect* checkbox before executing.

It is then necessary to disconnect and reconnect to the database so that the full list of FreeAdhocUDF external functions can be viewed in the DB Explorer under the DB object branch *UDF*.

Because of the sheer quantity of UDFs, there is no longer a single SQL script but a number of `DECLARE` SQL scripts for the individual UDF types, so that users only need to install those UDFs that they require.

There are also a number of different versions for some functions, e.g. for `SUBSTR` - for reasons of compatibility; which means different `DECLARES` compatible to:

- FreeUDFLib
- FreeUDFLibC
- FrUDF or AvERPUDF (for those working from AvERP)
- RFunc

In such cases it is necessary to "uncomment" the relevant script parts.

For those RFunc users who have recently transferred to FreeAdhocUDF, there is an special script, so that the application containing the script does not need to be altered.

Important: please refer to *Necessary update for existing databases* before updating to FreeAdhocUDFmin.

The most recent news, information and documentation can be found at: http://www.udf.adhoc-data.de/index_eng.html.

FreeAdhocUDF complete installation

Installation of versions published on or before November 30, 2006: Unzip the file, select the required `FreeAdhocUDF.dll` or `FreeAdhocUDF.so` and copy to the InterBase/Firebird bin or udf directory, for example: `C:\Program Files\InterBase Corp\InterBase\bin`, `C:\Program Files\Borland\InterBase\udf\bin` or `C:\Program Files\Firebird\udf\bin`.

The `FreeAdhocUDF_declarations_all_dialect1.sql` or the `FreeAdhocUDF_declarations_all_dialect3.sql` script should then be copied into the IBExpert Script Executive (found in the Tools menu), and executed using [F9]. A database connection must exist, as UDF libraries need to be registered for each database (i.e. they are database-dependent and not server-dependent). If necessary, use the Script Executive menu item *Add CONNECT statement* to connect to the desired database, or click the *Use current connect* checkbox before executing.

It is then necessary to disconnect and reconnect to the database so that the full list of FreeAdhocUDF external functions can be viewed in the DB Explorer under the DB object branch *UDF*.

Necessary update for existing databases

Because it's possible in FireBird 2.0 for UDFs to return `<null>` instead of `0` or `empty string`, the code of most functions must be changed basically to use this mechanism. So a `RETURN INTEGER BY VALUE` becomes a `RETURN INTEGER FREE_IT`.

To use the functions in the new version with an old declaration, which includes a `BY VALUE`, returned for example in `SELECT F_HOUR ('30.11.2006 15:00:00') FROM RDB$DATABASE` plus something nonsensical - a number with 8 digits - but not 15. It is therefore necessary in all cases to replace the old declarations with the new ones - even you do not want to use Firebird 2.0 or do not want to use the `<null>` option.

If you have declared your FreeAdhocUDF with a version prior to version `adhoc200612xx` you have to delete the UDFs and redeclare them or - if you can't delete them because of existing dependencies - you have to use the `UPDATE` script `adhoc200612update_xxx.sql`! You'll find them at http://www.ibexpert.com/download/udf/.

For UUID-functions there is no `UPDATE` script, you have to delete the declarations in the database and then rebuild them with the new `DECLARE` script.

For the function F_TRUNCATE the entry point had to be altered from truncate to `f_truncate` to avoid conflicts with a standard function in C. This is also part of the UPDATE script.

The current DECLARE scripts are up to date (March 2007).

See also:
Aggregate Functions Conversion Functions DECLARE EXTERNAL FUNCTION (incorporating a new UDF library) Threaded Server and UDFs

Blob filter

Blob filters are routines for blobs. They translate blob data from one type to another, i.e. they allow the contents of blob subtype `X` to be displayed as subtype `Y` or vice versa. These filters are ideal tools for certain binary operations such as the compression and translation of blobs, depending upon the application requirements.

A blob filter is technically similar to a UDF (user-defined function). It hangs itself in the background onto the database engine, and is used for example to compress the blob, or to specify the format such `GIF` or `JPG` (dependent upon use with Windows or Apple Mac). The blob filter mechanism relies on knowing what the various subtypes are, to provide its functionality.

Blob filters are written in the same way that UDFs are written, and are generally part of standard libraries, just as UDFs are.

Declaring a blob filter

A blob filter needs to be explicitly declared in the database before it is used. This is done using the keyword `DECLARE FILTER`. First it is necessary to connect to the database using the blob filter, and then issue the statement. The syntax of `DECLARE FILTER` is as follows:

```
DECLARE FILTER <IB/FB_Filter_Name>
<Parameter_List>
INPUT TYPE <Type>
OUPUT TYPE <Type>
ENTRY_POINT <External_Function_Name>
MODULE_NAME <Library_Name>;
```

New to Firebird 2.0: Declare BLOB subtypes by known descriptive identifiers

Previously, the only allowed syntax for declaring a blob filter was that above. Since Firebird 2.0 there is an alternative new syntax:

```
DECLARE FILTER <name>
  INPUT_TYPE <mnemonic>
  OUTPUT_TYPE <mnemonic>
  ENTRY_POINT <function_in_library>
  MODULE_NAME <library_name>;
```

where `<mnemonic>` refers to a subtype identifier known to the engine.

Initially they are binary, text and others mostly for internal usage, but it is possible to write a new `mnemonic` in `rdb$types` and use it, since it is parsed only at declaration time. The engine keeps the numerical value. Please don't forget that only *negative* subtype values are meant to be defined by users.

To view the predefined types, do

```
select RDB$TYPE, RDB$TYPE_NAME, RDB$SYSTEM_FLAG
  from rdb$types
  where rdb$field_name = 'RDB$FIELD_SUB_TYPE';

RDB$TYPE RDB$TYPE_NAME RDB$SYSTEM_FLAG
========= =========================== =================
0 BINARY 1
1 TEXT 1
2 BLR 1
3 ACL 1
4 RANGES 1
5 SUMMARY 1
6 FORMAT 1
7 TRANSACTION_DESCRIPTION 1
8 EXTERNAL_FILE_DESCRIPTION 1
```

Examples can be found at: Declare BLOB subtypes by known descriptive identifiers.

Calling a blob filter

In the same way as UDFs, blob filters can be called from InterBase/Firebird code whenever an InterBase/Firebird built-in function call is used. In order to use the blob filter, invoke the `FILTER` statement when declaring a cursor. Then, whenever InterBase/Firebird uses the cursor, the blob filter is automatically invoked.

See also:
BLOB
Firebird for the database expert - Episode 2: PageTypes
User-Defined Function (UDF)

Role

A role is a named group of privileges. It simplifies granting user rights as multiple users can be granted the same role. For example, in a large sales department, all those clerks involved in processing incoming orders could belong to a role `Order Processing`.

Should it become necessary to alter the rights of these users, only the role has to be changed.

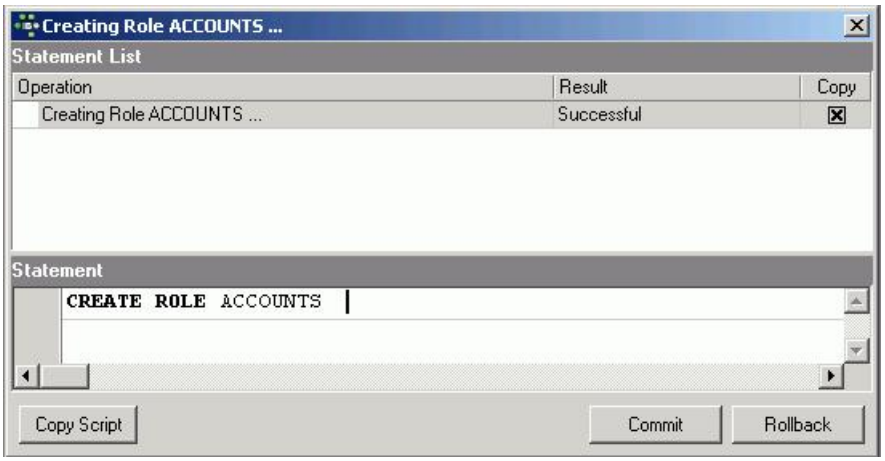
New role

A new role can be created in a connected database, either by using the IBExpert menu item Database / New Role, the respective icon in the New Database Object toolbar, or using the DB Explorer right-click menu (or key combination [Ctrl + N]), when the role heading of the relevant connected database is highlighted.

A *NewRole* dialog appears:

Simply enter the new role name, and click *OK* to compile and commit.

Note: when a role with the name SYSDBA is created, no other users (not even the SYSDBA) can access the database.



For those preferring SQL input, the syntax is as follows:

```
CREATE ROLE <Role_Name>;
```

After successfully creating one or more new roles, privileges need to be granted to the role name(s). Please refer to Grant Manager, found in the IBExpert Tools Menu, and the GRANT statement for further information.

Alter role

Users and rights may be altered for a role using the IBExpert Grant Manager. This can be started either directly from the DB Explorer by either double-clicking on a role name, using the right-click menu item *Edit Role...* or the key combination [Ctrl + O], or using the IBExpert Tools menu item, Grant Manager. Please refer to Grant Manager for details.

Drop role/delete role

To drop a role use the DB Explorer right mouse-click menu item *Drop Role...* (or [Ctrl + Del].

IBExpert asks for confirmation:

before finally dropping the role. Once dropped, it cannot be retrieved.

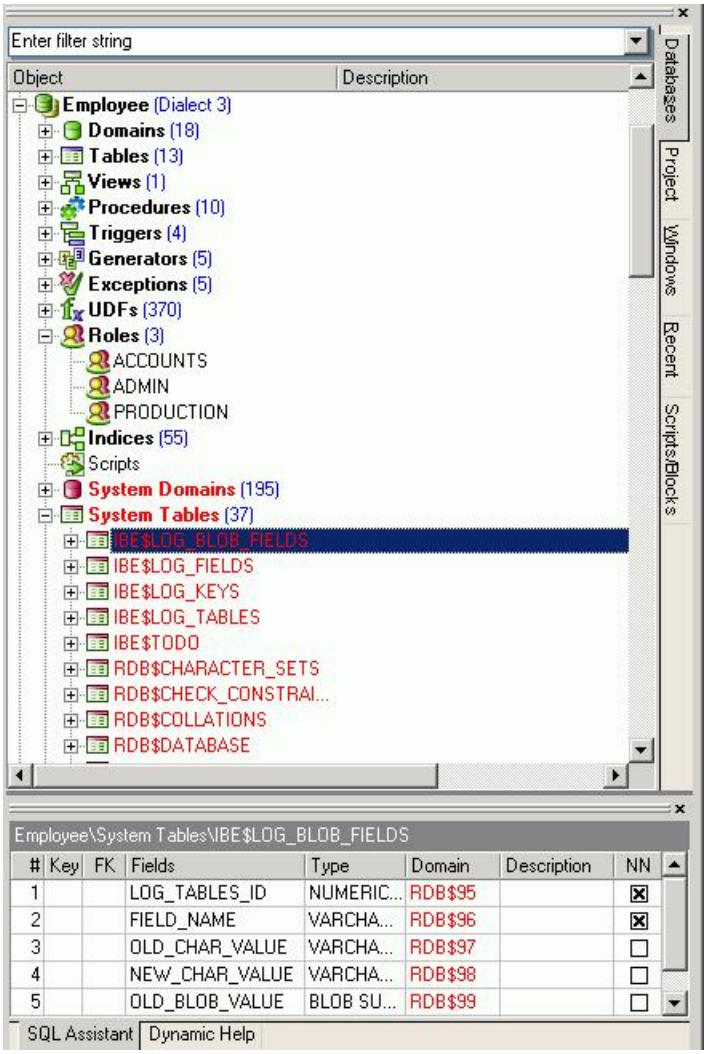
Using SQL the syntax is:

```
DROP ROLE <Role_Name>;
```

See also:
Grant Manager
User Manager
Server Security ISC4.GDB / SECURITY.FDB

System objects

InterBase/Firebird generates system database objects, and stores its own specific system information about the database objects in system tables. System objects are displayed in the DB Explorer in red, if the system options have been flagged in the Register Database dialog (called using the right mouse button Additional/DB Explorer).

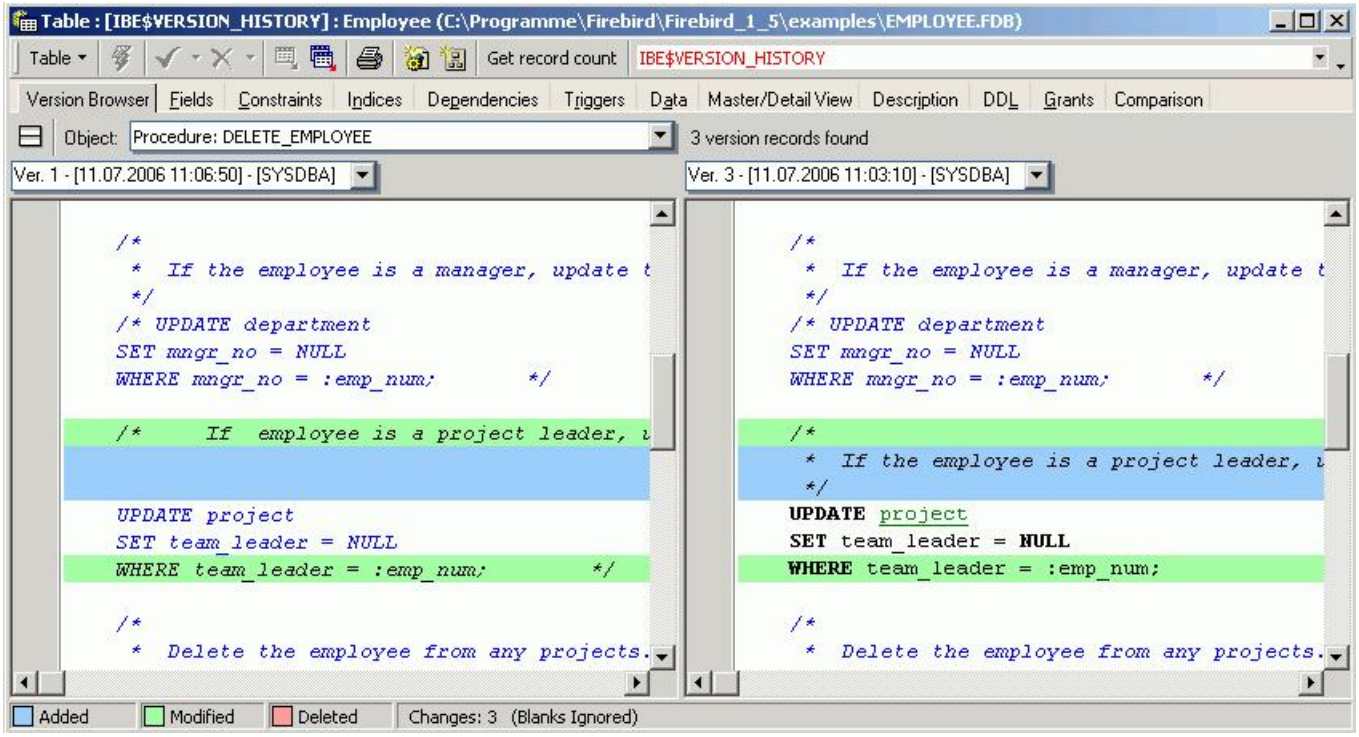


Firebird system objects contain the prefix `RDB$`; IBExpert system objects contain the prefix `IBE$`.

A newly created database is almost 0,5 MB large. This is due to the system tables that are automatically generated by InterBase/Firebird when a database is created.

`IBE$VERSION_HISTORY` system table

A special browser was introduced in IBExpert version 2006.06.05, implemented for the `IBE$VERSION_HISTORY` table. When `IBE$VERSION_HISTORY` is opened in the Table Editor, a new *Version Browser* page is automatically opened:



Select the database object and the versions you wish to compare. Text and code is highlighted according to whether it has been added, modified or deleted.

See also:
Verson History

Text Editor / SQL Code Editor

All Object Editors and SQL Editors include text/SQL input windows. Please refer to the individual subjects, for further information. For example:

- SQL Editor / Edit page
- Plan Analyzer
- SQL Editor / Logs
- Description page
- Debugger
- DDL page
- SQL Monitor
- Stored Procedure

Objects and fields can be simply and quickly dragged and dropped from the DB Explorer and SQL Assistant into the *Edit* page. Since version 2004.2.26.1 this has been greatly improved. When an object node(s) is dragged from the DB Explorer or SQL Assistant, IBExpert offers various versions of text to be inserted into the code editor.

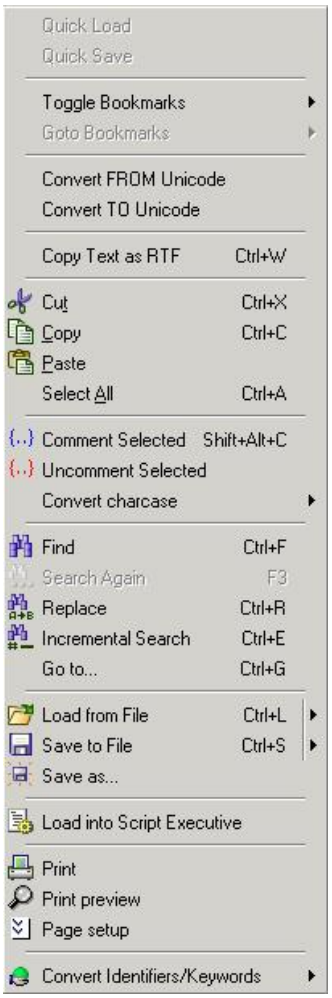
Since version 2006.08.12 IBExpert offers you the following choices when dragging a database node from the DB Explorer tree into any code editor: `CONNECT` statement, `CREATE DATABASE` statement, IBEBlock with the `ibec_CreateConnection` function.

A Code Insight system is included to simplify command input and database objects are underlined for easy recognition.

Hyperlinks allow you to quickly reference database objects if necessary.

There are a number of options available to customize the appearance of the code in the Text Editor. Please refer to the IBExpert Options menu item, Editor Options, to view and specify all options available. For example, it is possible to customize the highlighting of variables. Use the IBExpert menu item Options / Editor Options / Color to select color and font style for variables.

The Text Editor/Code Editor has its own comprehensive right-click context-sensitive menu, the contents of which are described in detail in the SQL Editor / SQL Editor Menu and IBExpert Edit menu. IBExpert version 2007.07.18 introduced the possiblity to convert text from/to unicode. If here is no text selected, the entire content of the code editor will be converted.



As with all working areas in IBExpert there are also a number of key combination shortcuts available here in the Text Editor. To view all short cuts or specify your own, use the Localizing Form (a complete list of all shortcuts and operations), opened using [Ctrl + Shift + Alt + L]. For example, a selected block of text can be simply and easily indented using [Ctrl + Shift + U] (decrease indentation using [Ctrl + Shift + I]).

New to IBExpert version 2005.09.25: Highlighting of paired brackets has been added. This option can be customized using the IBExpert Options menu item, Editor Options / Color.

New features added in IBExpert version 2006.10.14 include the visual representation and highlighting of paired `BEGIN/END` and `CASE/END` clauses, and the implementation of round brackets on a code editor gutter.

Printing from the database object editors

Print Table

Please refer to the IBExpert Edit Menu item Print and the Table Editor menu item Printing Options.

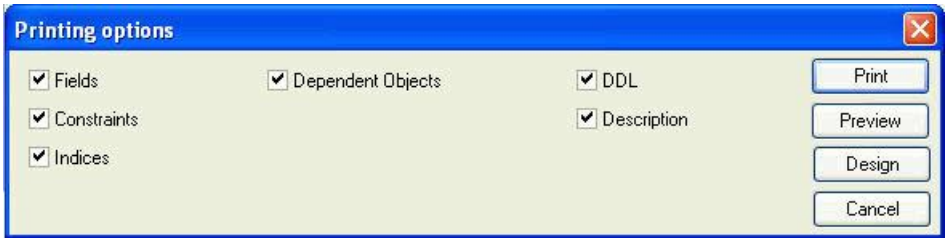
Print Preview and Print Design

Please refer to the IBExpert Report Manager for further information.

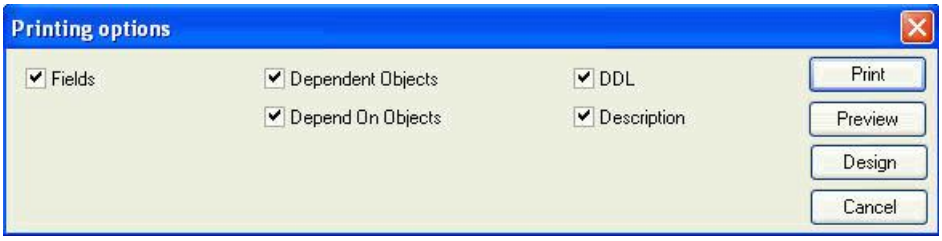
Printing Options

The *Printing Options* dialog can be started using the *Print Table Metadata* icon or [Shift + Ctrl + P].

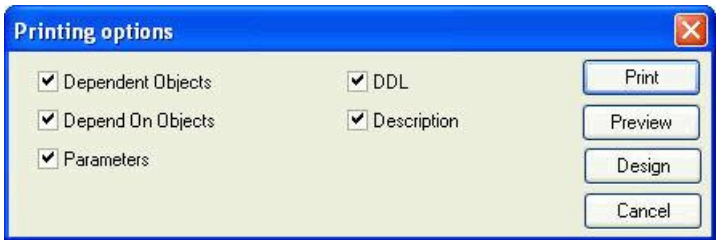
The *Printing Options* dialog offers different options depending upon which Editor it is started from. For example, when started from the Table Editor:



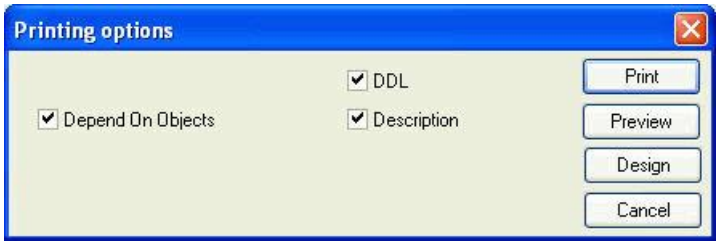
the View Editor:



the Procedure Editor:



the Trigger Editor:



These options include the following:

- Fields
- Constraints
- Indices
- Dependent Objects
- Depend On Objects
- Parameters
- DDL

- Description

Simply check as wished, and then click *Preview* (to view the report as it will be printed - refer to Report Manager for further information), *Design* (to customize the report - again refer to Report Manager for further information) or *Print* to proceed to the standard Windows *Print* dialog.

IBExpert Edit menu

The IBExpert Edit menu offers typical manipulation options found in the majority of windows applications. It includes:

- Load and save to file
- Cut, Copy and Paste
- Find, Search Again and Replace
- Incremental Search
- Print Preview, Print and Page Setup

Load from File / Save to File

These first two items in the IBExpert Edit menu can also be called using the SQL Editor right-click menu (available in the SQL and Object Editors) or the key combinations [Ctrl + L] or [Ctrl + S] respectively. These items can also be found in the Edit toolbar. They allow SQL scripts etc. to be loaded or saved to file.

Cut / Copy / Paste / Select All

These three items can be found in the IBExpert Edit menu and SQL Editor right-click menu (available in the SQL and Object Editors). They can also be executed using the key combinations:

- *Cut* [Ctrl + X]
- *Copy* [Ctrl + C]
- *Paste* [Ctrl + V]

These items can also be found in the Edit toolbar. They allow selected (i.e. marked) text to be cut or copied into the clipboard, and then pasted - either directly in IBExpert or in other applications, such as Windows Editor, Word etc.

The menu item *Select All* [Ctrl + A] selects a complete text (e.g. SQL script).

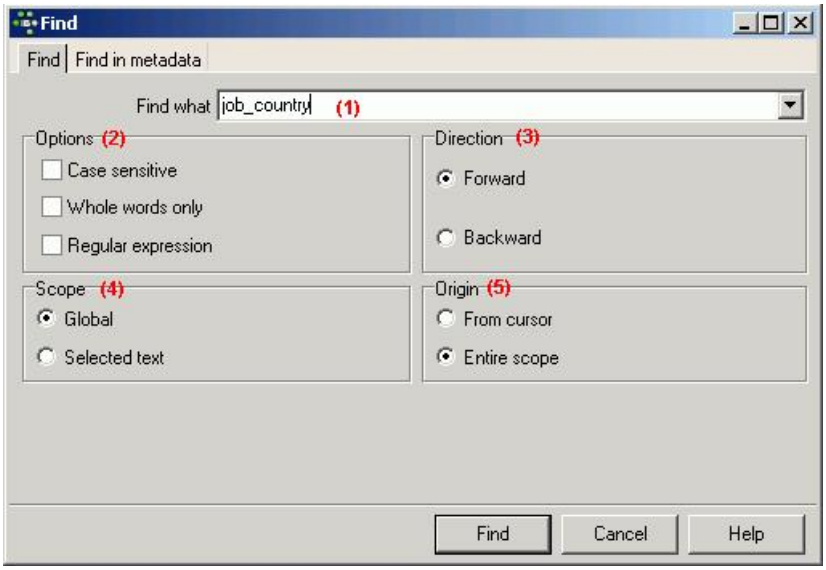
Find / Search Again / Replace

These three items can be found in the IBExpert Edit menu and SQL Editor right-click menu (available in the SQL and Object Editors). They can be executed using the key combinations:

- *Find* [Ctrl + F]
- *Search Again* [F3]
- *Replace* [Ctrl + R]

or the respective icons in the Edit toolbar.

They are useful for finding individual words/digits or word/digit strings in longer texts or metadata. The *Find* dialog offers a number of options:



Find page:

(1) Find What: the *Find* dialog automatically offers the word, where the cursor is currently standing, or a selected text. This can be altered as wished. *Previous Find* criteria can be selected using the pull-down list.

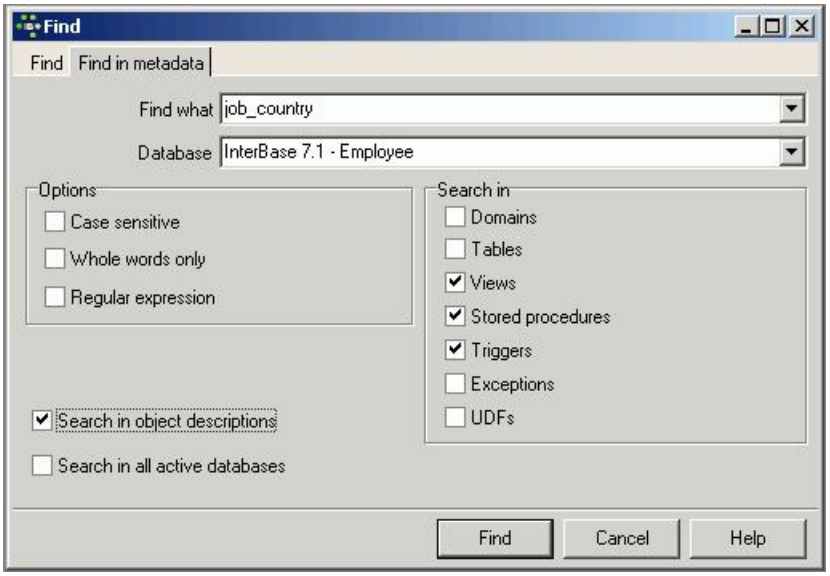
(2) Options: This includes *Case Sensitive*, *Whole Words Only* and *Regular Expressions* (e.g. *,?).

(3) Direction: i.e. forwards or backwards.

(4) Scope: i.e. global or just the selected text.

(5) Origin: From cursor (searches from the cursor position onwards), or entire scope (complete text).

The *Find in Metadata* page offers alternative options:

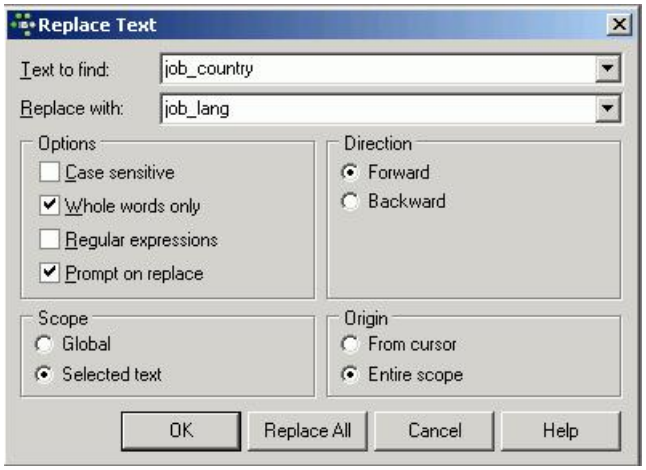


These include database selection (or even a *Search in all active databases* option using the checkbox at the bottom of the dialog) and, in addition to the options offered on the *Find* page, a check list of the database object categories to be searched.

New to version 2004.08.05.1 is the checkbox option to search for text or text strings within database object descriptions.

Replace:

The *Replace* dialog is similar to the *Find* page :



with the following additions:

Replace with:

Enter the word(s)/number(s) that are to replace the searched for text. Previous *Replace* entries can be selected using the pull-down list.

The *Options* check list contains the additional check *Prompt on Replace* (default), allowing the user to check that the found word/number string is correctly replaced.

Incremental Search

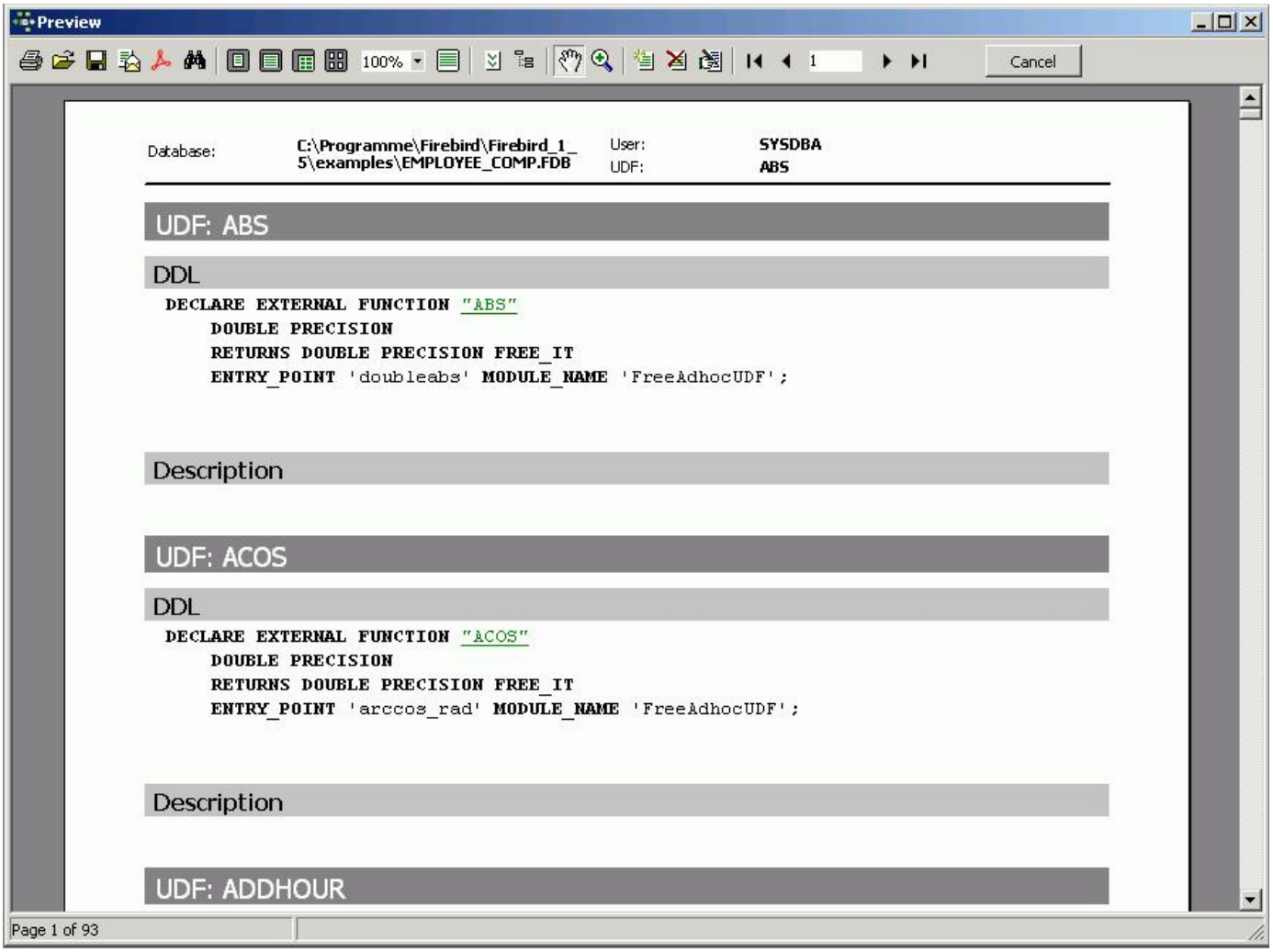
The *Incremental Search* [Ctrl + F] allows a simple search for individual entries by simply marking the desired column header, clicking the right mouse button menu item *Incremental Search* [Ctrl + F] and then typing the relevant digits/letters, until the required dataset(s) is/are found. Alternatively, the [Ctrl + Enter] keys can be used to search for the next occurrence of a substring.

This menu item can also be found in the context-sensitive menus in the Table Editor / Data page and in all editors containing an SQL Editor window and right-click SQL Editor Menu.

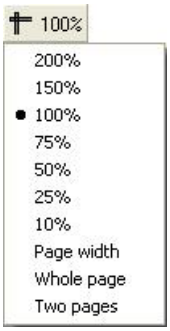
Print Preview

This item can be found in the IBExpert Edit menu and SQL Editor right-click menu (available in the SQL and Object Editors).

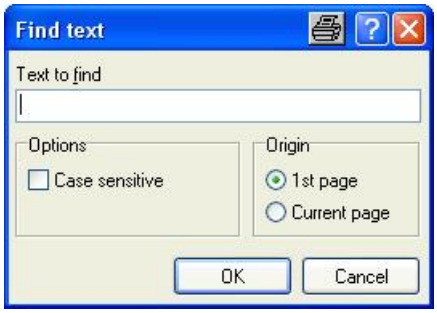
The *Print Preview* dialog is part of the Fast Report Manager and, when opened, displays the current script/report. It offers a number of options:



It is possible to specify the view scale, using the respective icon or the right-click menu:



Further options include opening a report/script, saving it, printing the report/script previewed, and even searching for text within the script:



The last icon in the Print Preview toolbar allows the Print Preview window to be closed.

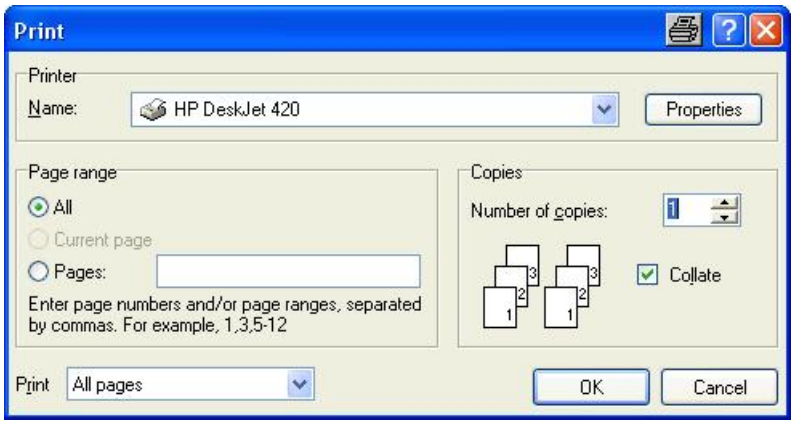
The right-click menu, in addition to scale specification, also offers options to add a page (for example, for a front cover or introduction) or delete one, and also to edit the page previewed, by opening the *Report Designer*.

The *Report Designer* (part of the Report Manager) can also be automatically opened by double-clicking on the report, enabling the user to make alterations to the layout as wished.

Print

This item can be found in the IBExpert Edit menu and SQL Editor right-click menu (available in the SQL and Object Editors), and as an icon on the relevant toolbars, for printing scripts, reports or database object metadata.

It opens a standard Windows Print dialog:



including the usual options such as printer specification (and properties), page range and number of copies.

Page Setup

This item can be found in the IBExpert Edit menu and SQL Editor right-click menu (available in the SQL and Object Editors).

It opens a standard *Windows Page Setup* dialog, where the following options can be specified:

- Paper size
- Source (i.e. printer tray specification)
- Portrait or landscape
- Margins

as well as a *Printer* button to specify the printer.

Convert Identifiers/Keywords

The menu item, *Convert Identifiers/Keywords*, can be found in the IBExpert Edit menu or in the right-click Text Editor/Code Editor menu. It offers the following options to alter the appearance of the SQL characters:

1. **Convert keywords:** allows all keywords (i.e. statements, commands etc.) in the current SQL script to be converted completely to lower or upper case.

2. **Convert identifiers:** allows all identifiers (i.e. object names, field names etc.) in the current SQL script to be converted completely to lower or upper case.

IBExpert Grid menu

The IBExpert Grid menu item is new to version 2003.11.6.1. It includes the following:

- Apply Best Fit
- Save Grid Data
- Copy Current Record to Clipboard
- Copy All to Clipboard

It is of course necessary to be in an active grid (e.g. Table Editor / Data page, View Editor / Data page, SQL Editor / Results page etc.) for any of these menu items to be effective!

Apply Best Fit

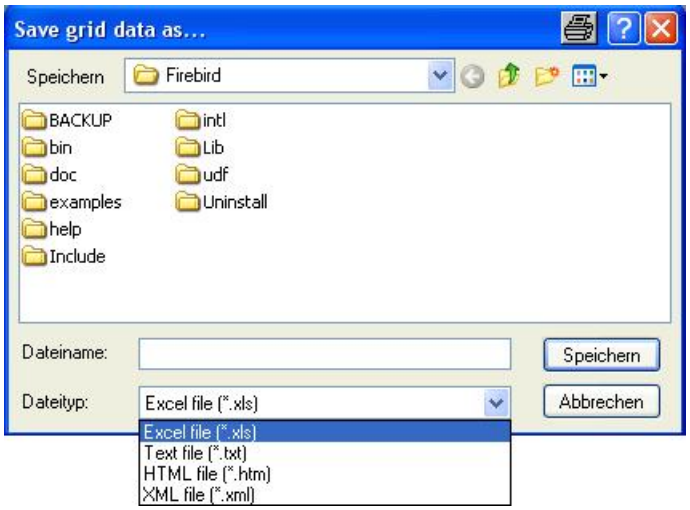
The IBExpert menu item *Apply Best Fit* is new to IBExpert version 2003.11.6.1 and can be started from the Grid menu, or using the key combination [Ctrl + (NumBlock +)].

This automatically adjusts all grid columns to the ideal width.

Save Grid Data

The IBExpert menu item *Save Grid Data as* is new to IBExpert version 2003.11.6.1 and can be started from the Grid menu, or using the key combination [Shift + Ctrl + S].

It opens the *Save Grid Data As...* dialog:



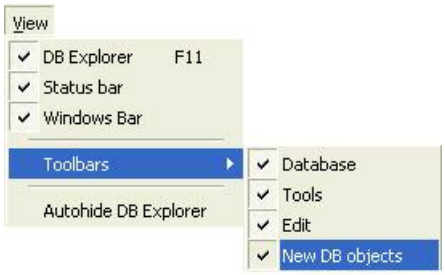
It is possible to save grid data into `TXT`, `XLS`, `HTML` or `XML` formats. This works only with dataset grids (field and index grids in the Table Editor, the parameters/variables grid in the Stored Procedure Editor while working in lazy mode), and doesn't work with SQL Assistant lists, the constraint list in the Table Editor etc.

Copy Current Record to Clipboard/Copy All to Clipboard

The IBExpert menu items *Copy Current Record to Clipboard and Copy All to Clipboard* are new to IBExpert version 2003.11.6.1 They can be started from the Grid menu, and used to copy either one selected record or all records (including column captions) in an active grid to clipboard. The values are delimited with the tab character.

IBExpert View menu

The IBExpert View menu allows the developer to specify which, of certain options, he wishes to have displayed on screen. This eliminates superfluous or unnecessary items on screen. The options available can be seen in the following illustration:



The options DB Explorer, status bar and windows bar can be blended in and out simply by clicking on the check box (alternatively using the space bar). The menu item Toolbar is subdivided into the four main standard toolbars: Database toolbar, Tools toolbar, Edit toolbar, and New DB Objects toolbar.

Autohide DB Explorer is a further alternative to quickly blend the DB Explorer in and out as wished (alternatively use the [F11] key). This option namely enables the DB Explorer to disappear automatically when any editor is opened - allowing a larger working area. It is blended back into view simply by holding the mouse over the left-hand side of the IBExpert main window.

IBExpert Options menu

The IBExpert Options menu enables you to organize your IBExpert working environment as you wish. It includes the following options:

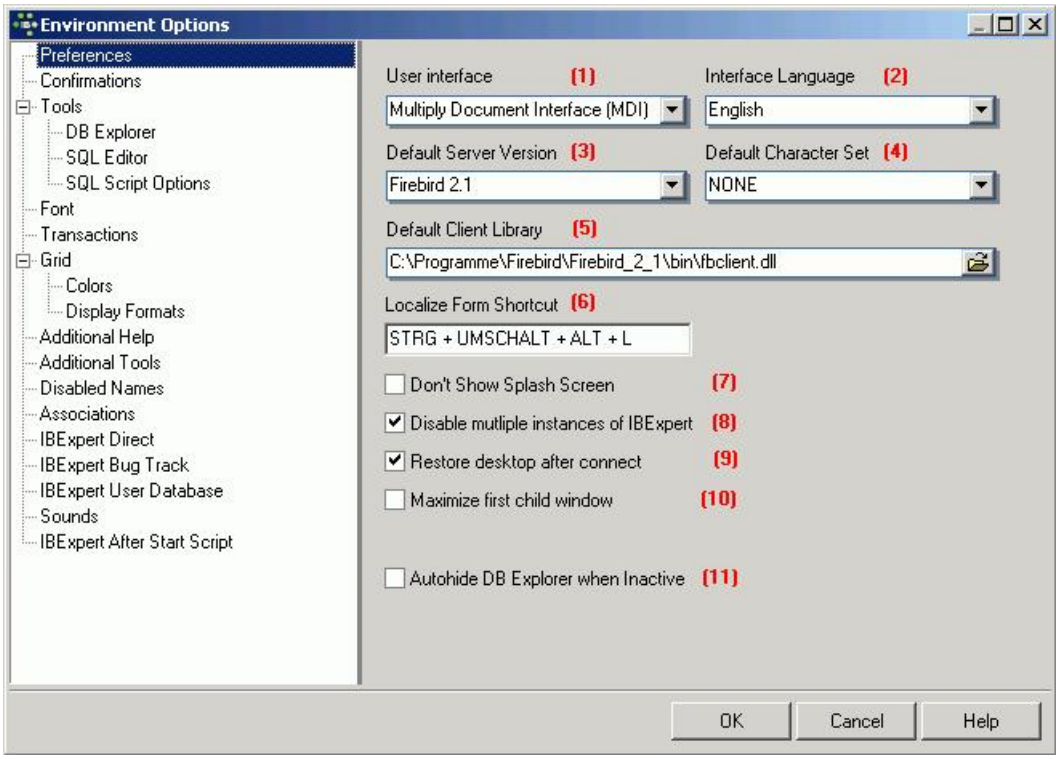
- Environment Options
- Editor Options
- Visual Options
- Keyboard Templates
- General Templates
- Object Editor Options

Environment Options

Environment Options can be found in the IBExpert Options menu. It enables the user to organize his IBExpert working environment as he wishes. It is possible, for example, to set certain defaults for editors and specific menu items, alter colors or the system font, etc.

Preferences

The *Preferences* window allows the user to specify certain general preferences or defaults.



These include:

(1) User interface

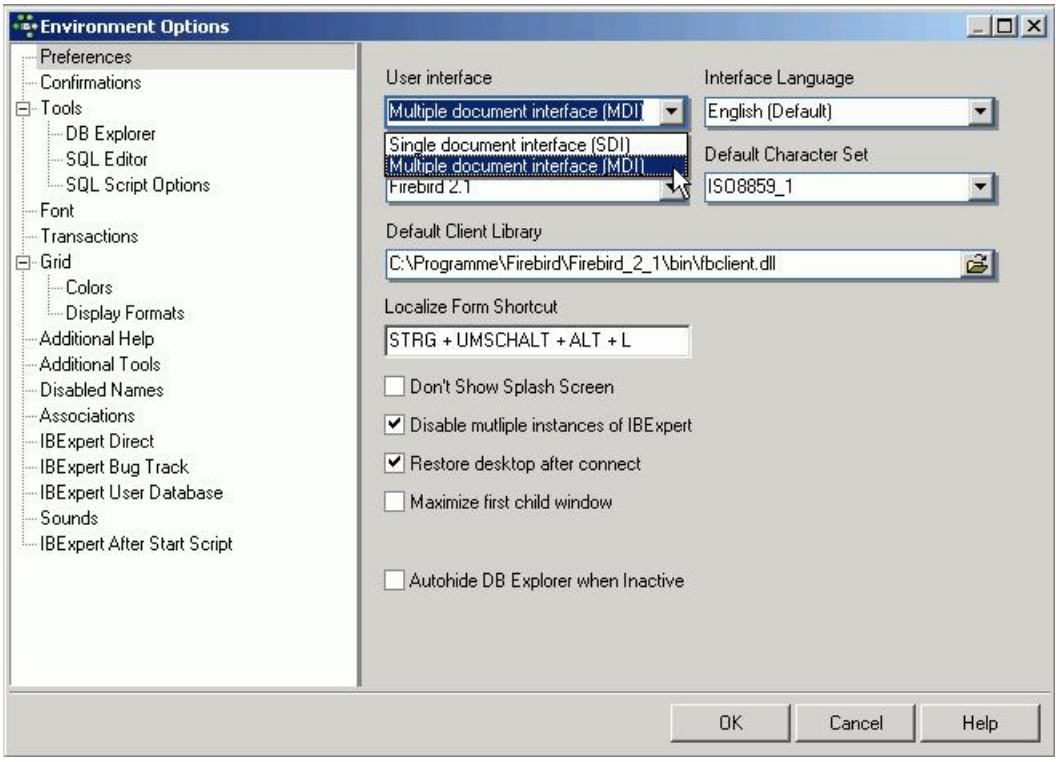
The pull-down list offers the options MDI or SDI (please see below for details). Note that changes to the user interface only take effect after IBExpert has been restarted.

The user interface is the connection between the machine and the user, i.e. the way the software is presented to the user on-screen. The user interface enables the user to use the program and manipulate data.

Under the IBExpert Options menu item, Environment Options, the user interface can be defined as SDI (Single Document Interface) or MDI (Multiple Document Interface).

MDI (Multiple Document Interface)

MDI is the abbreviation for Multiple Document Interface. It can be specified in the IBExpert menu item Options / Environment Options.



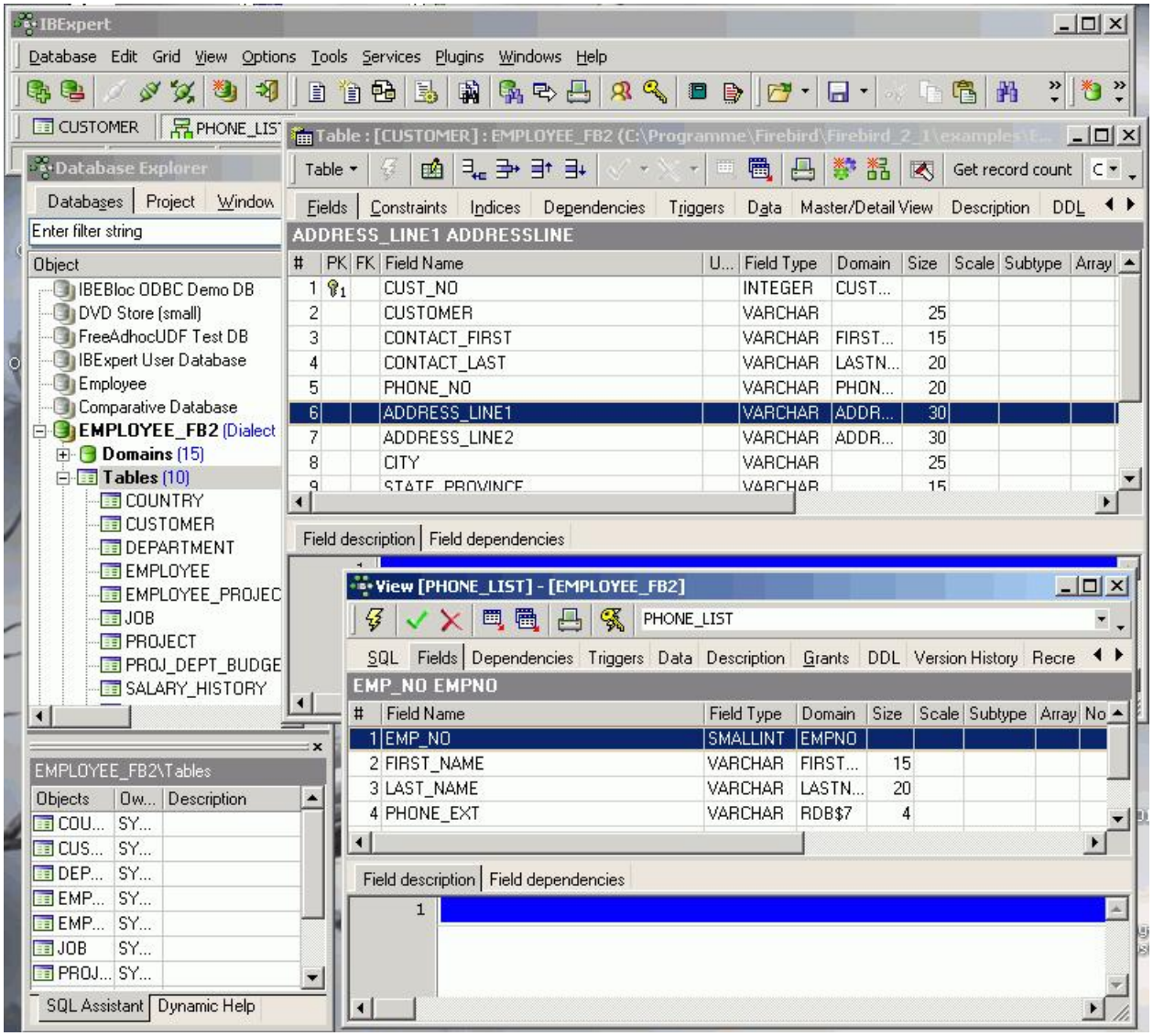
This is the recommended interface, as all windows are contained within one main Window, similar to MS applications. There is one document per window. For all additional objects or documents, the Windows operating system opens an additional window.

The status bar can be seen at the bottom of the screen.

When changing the interface from SDI to MDI and vice versa, IBExpert needs to be restarted for the alterations to take effect.

SDI (Single Document Interface)

SDI is the abbreviation for Single Document Interface.



The windows are spread freely and somewhat haphazardly over the screen, similar to Delphi. The status bar is part of the upper menu and toolbar panel.

Careful: it is possible to accidentally move a window totally out of view!

When altering the user interface from SDI to MDI and vice versa, IBExpert needs to be restarted for the change to take effect.

(2) Interface language

The default language is English. The pull-down list offers the following alternative languages:

- Czech
- Dutch
- English
- French
- German
- Hungarian
- Italian
- Japanese
- Polish
- Portuguese
- Romanian
- Russian
- Spanish

Should you not be able to see the full list of languages in the drop-down list, either delete the `ibexpert.lng` file or rename the `english.lng` file, found in the IBExpert Languages directory, to `ibexpert.lng`, and place this in the main IBExpert directory.

(3) Default server version

If the same database version is used for all projects, it is advisable to set a default version here. This saves having to enter the database server version every time a database is registered. The pull-down list offers the following database versions:

- Unknown (default)
- Firebird 1.0

- Firebird 1.5
- Firebird 2.0
- Firebird 2.1
- InterBase 5.x
- InterBase 6.1
- InterBase 6.5
- InterBase 7.0
- InterBase 7.1
- InterBase 7.5
- InterBase 2007
- Yaffil 1.0

(4) Default character set

The default character set is the character set defined when creating the database, and applicable for all areas of the database unless overridden by the domain or field definition. It controls not only the available characters that can be stored and displayed, but also the collation order. If not specified, the parameter defaults to NONE, i.e. values are stored exactly as typed.

Please refer to the *Create Database* chapter for further information.

The following character sets are currently available:

- ASCII
- BIG_5
- CYRL
- DOS437
- DOS850
- DOS852
- DOS857
- DOS860
- DOS861
- DOS863
- DOS865
- EUCJ_0208
- GB_2312
- ISO8859_1
- ISO8859_2
- KSC_5601
- NEXT
- NONE
- OCTETS
- SJIS0208
- UNICODE_FSS
- UTF8
- WINI1250
- WINI1251
- WINI1252
- WINI1253
- WINI1254

(5) Default client library

The `GDS32.DLL` is dependent upon the database server. Firebird has, in addition to this, its own library, `FBCLIENT.DLL`. The `GDS32.DLL` is however also included for compatibility reasons. When working with Firebird, or different InterBase/Firebird server versions, the DLL can be selected here, as wished; simply click the *Open File* icon to the right of this field, to select the library required.

Since IBExpert version 2006.01.29, the Script Executive always uses this default client library unless it is overriden using the `SET CLIENTLIB` command directly in the Script Executive editor.

(6) Localize form shortcut

Here you can specify your own shortcut for opening the Localizing Form, if you do not wish to use the default [Ctrl + Shift + Alt + L]. The Localizing Form displays all functions and the respective key combinations, which can also be customized. Please refer to Localizing Form for further information.

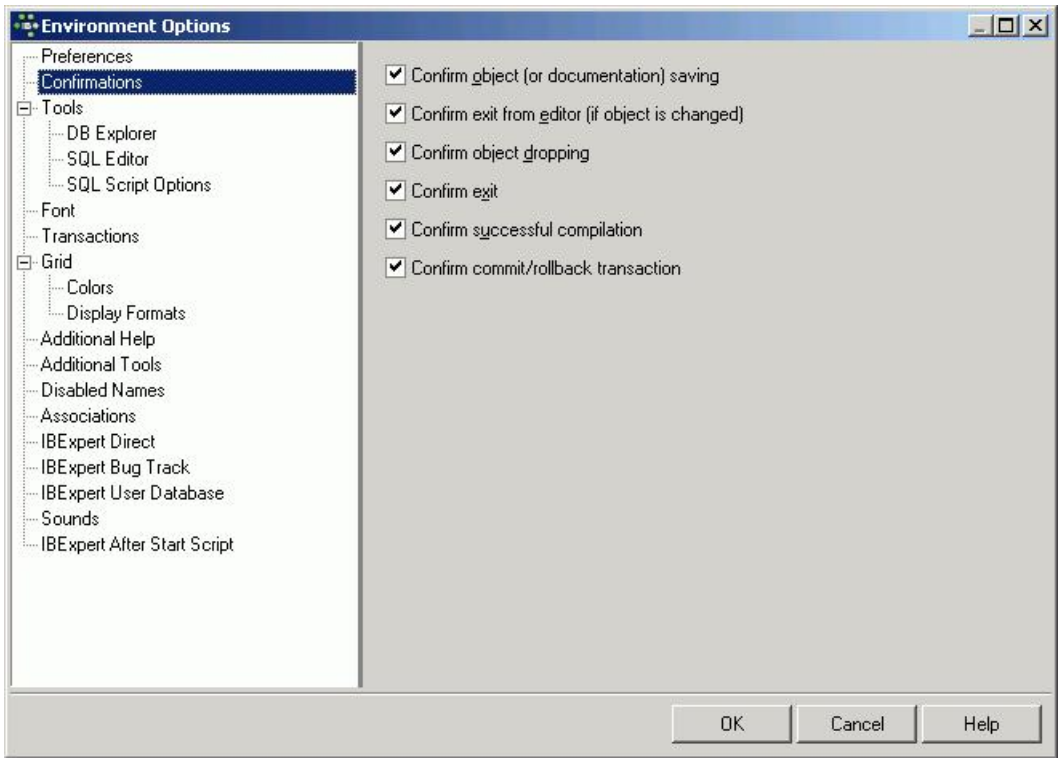
Check options

The following features can be checked or unchecked as wished:

(7) Don't Show Splash Screen: disables the IBExpert Splash Screen displayed whilst IBExpert is being loaded.
(8) Disable multiple instances of IBExpert: when checked this option ensures that IBExpert is only opened once.
(9) Restore desktop after connect: if this option is checked, IBExpert will restore all those forms left open as the last connection was ended, when it reconnects to the database.
(10) Maximize first child window: the first Editor/window opened is automatically expanded to fill the maximum screen area. This option is only available in the MDI version.
(11) Autohide DB Explorer when inactive: this option hides the DB Explorer automatically, if it is not focused. In other words, when the mouse is held over the left area, the DB Explorer appears; when the mouse is removed to begin work in an editor or child window, the DB Explorer is blended out, offering a larger work area.

Confirmations

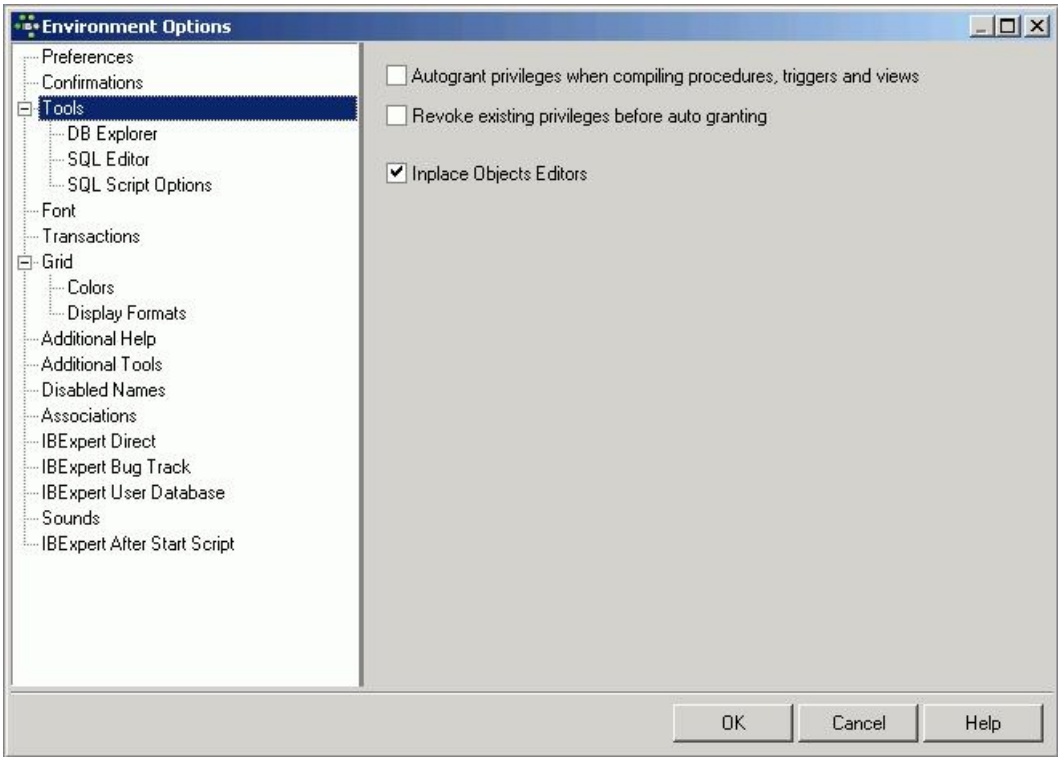
Some users find it annoying to be constantly asked for confirmation, whether or not they really want to carry out an operation. This window allows the user to specify, which confirmations he considers wise.



The following options are available:

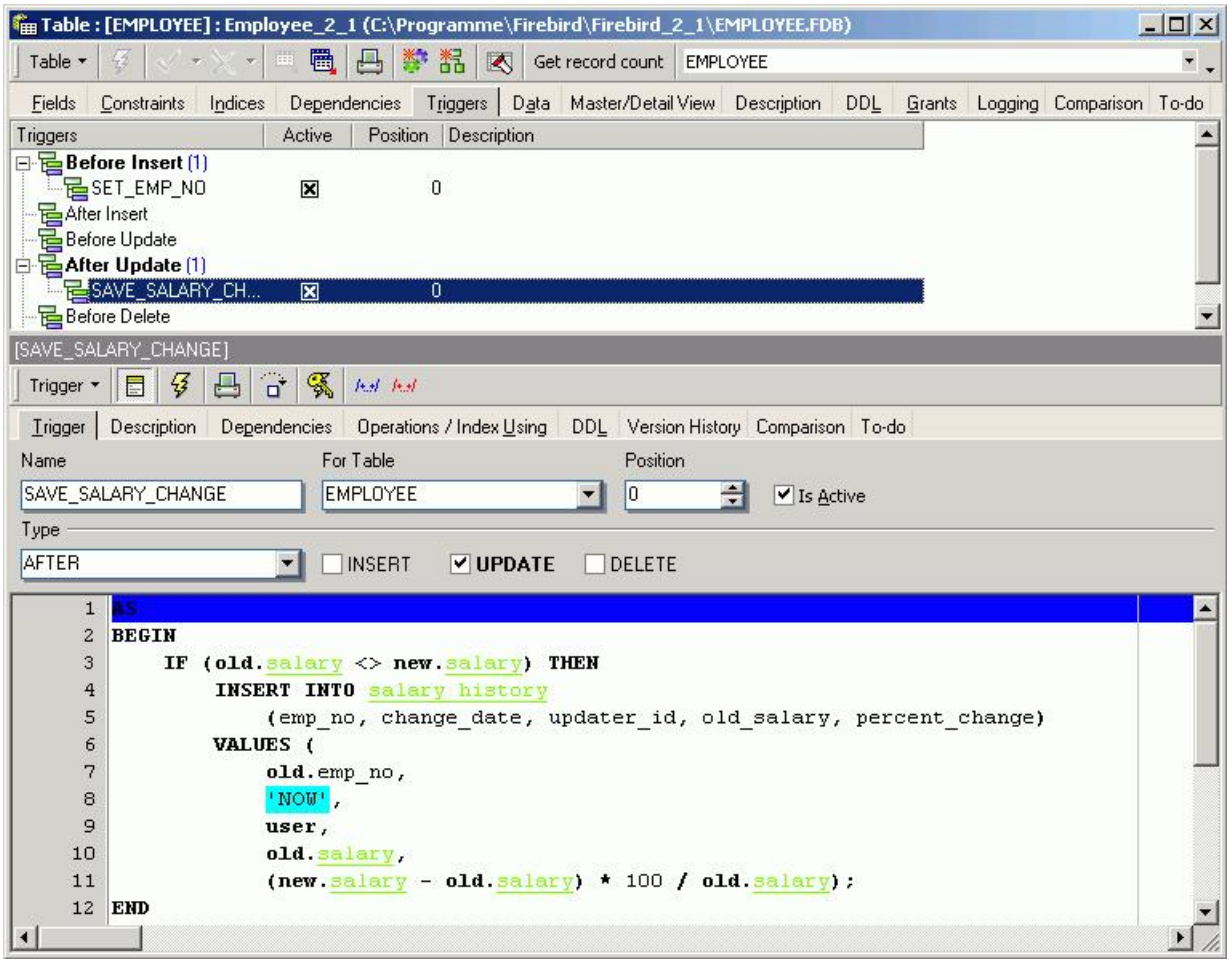
- **Confirm object (or documentation) saving:** if this option is checked, IBExpert will request confirmation before saving object modifications or descriptions.
- **Confirm exit from editor (if object is changed):** if this option is checked, IBExpert will request confirmation, if alterations have been made, before exiting from an object editor.
- **Confirm object dropping (recommended):** if this option is checked, IBExpert will request confirmation before dropping any database object.
- **Confirm exit:** if this option is checked, IBExpert will request confirmation before closing IBExpert.
- **Confirm successful compilation:** (recommended) if this option is checked, IBExpert displays a dialog, showing whether compilation was successful or not.
- **Confirm commit/rollback transaction:** (recommended) this option determines whether a message box appears, asking for confirmation when a user commits or rolls back active transactions in the SQL Editor, Table Editor, View Editor or Stored Procedure Editor.

Tools



The *Tools* page allows the user to specify the following for all tools if wished:

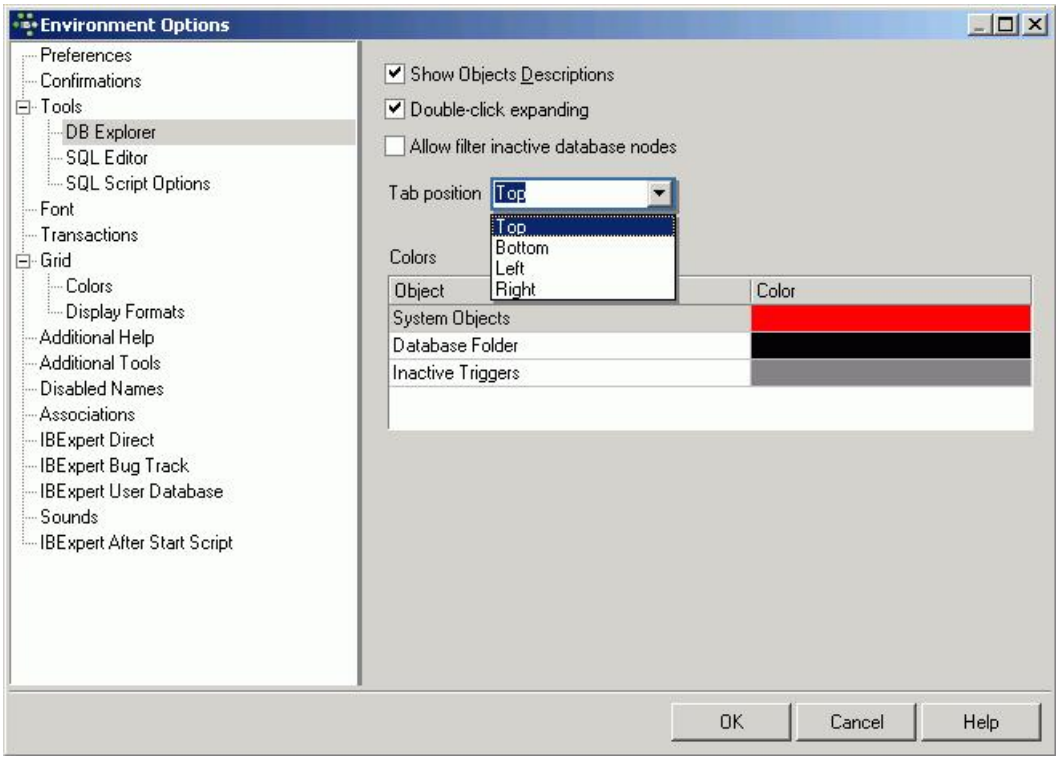
- **Autogrant privileges when compiling procedures, triggers and views:** this saves the repetitive task of autogranting privileges on the *Grants* page of the object editors each time a new procedure, trigger or view is created, and prevents the problems which inevitably arise should the assignment of rights be forgotten.
- **Revoke existing privileges:** if this option is enabled, an object's (stored procedure, trigger, view) existing privileges will be deleted before granting it new privileges.
- **Inplace Objects Editors:** this item applies to the so-called editors within editors.



For example, the Table Editor is active and a trigger is selected on the Trigger page: if this option is not checked, an SQL Editor window appears automatically in the lower part of the Table Editor, displaying the trigger code, but not allowing any changes to be made. When this option is however checked,

a simple click on a trigger automatically opens the Trigger Editor in this lower area, enabling work to be done on it, without having to leave the Table Editor and opening the Trigger Editor.

DB Explorer

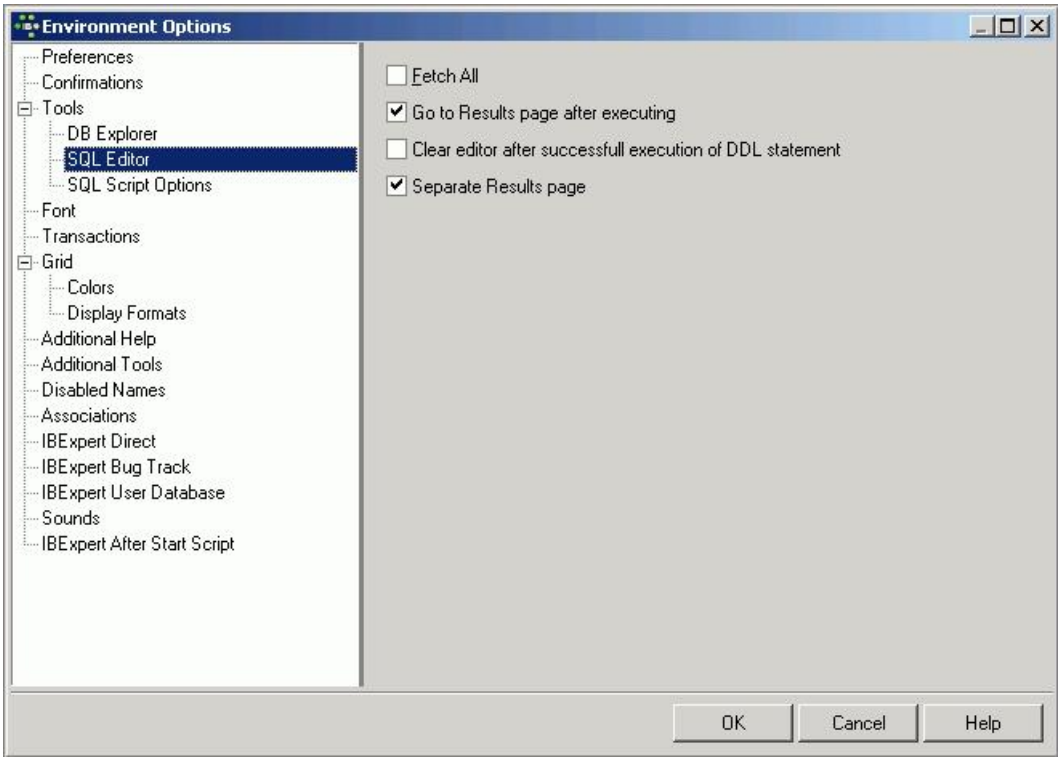


Here it is possible to specify whether database object descriptions should be displayed or not (this only makes sense if object descriptions are entered by the user), and whether double-click expanding (for the DB Explorer tree) is desired. Further options include a check-box option to allow filtering of inactive database nodes and, since IBExpert version 2007.07.18, the tab position of the Database Explorer pages can be also defined here.

Furthermore, colors may be specified for the following:

- system objects
- database folders
- inactive triggers

SQL Editor

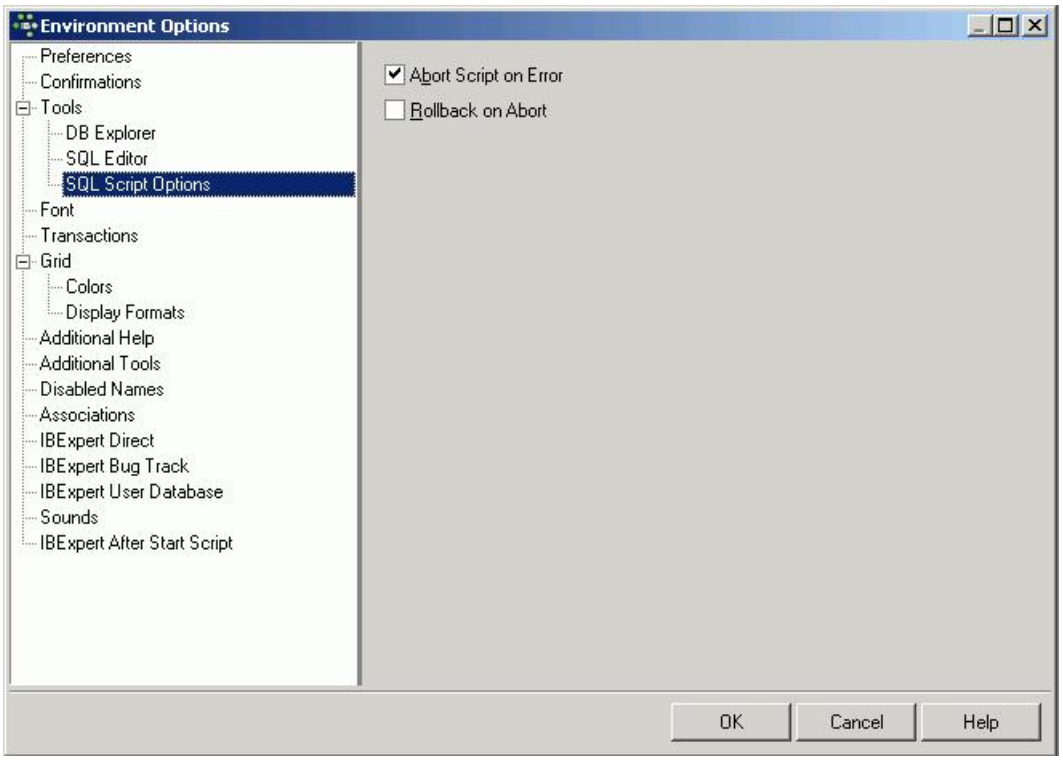


The following options may be user-defined for the SQL Editor:

- **Fetch All:** when this option is checked, all records corresponding to the query will be extracted from the table and displayed on the *Results* page, as opposed to only those displayed that are visible in the *Results* area, when this option is left unchecked.
- **Go to Results page after executing:** this option is only worth checking of course if you have specified *Separate Results page* (see below).
- **Clear editor after successful execution of DDL statement:** this clears the results page after the query has been committed.
- **Separate Results page:** turn this option off to place SQL query results directly below the Code Editor, or activate it to display query results on a separate page (found directly to the right of the *Edit* page).

SQL Script Options

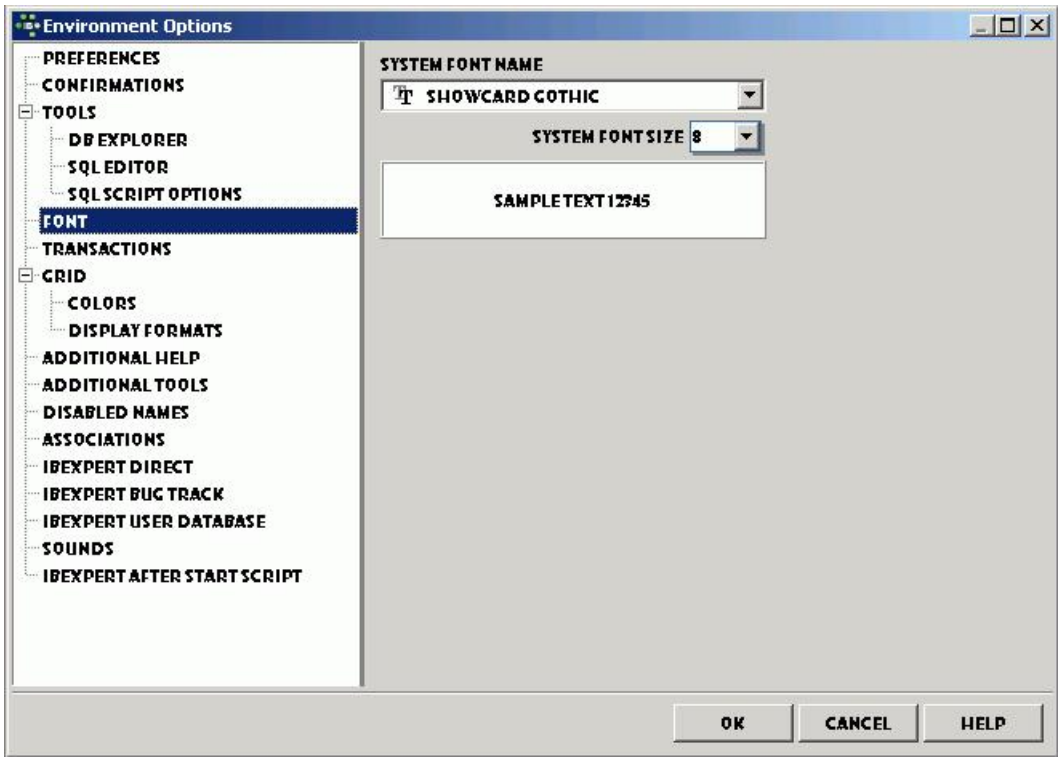
The SQL *Script Options* page offers the following user specifications:



- **Abort Script on Error:** the script execution is halted the moment an error is detected.
- **Rollback on Abort:** the script is automatically rolled back the moment an error is detected in the script. This option is only possible, if the first item, *Abort Script on Error*, is already selected.

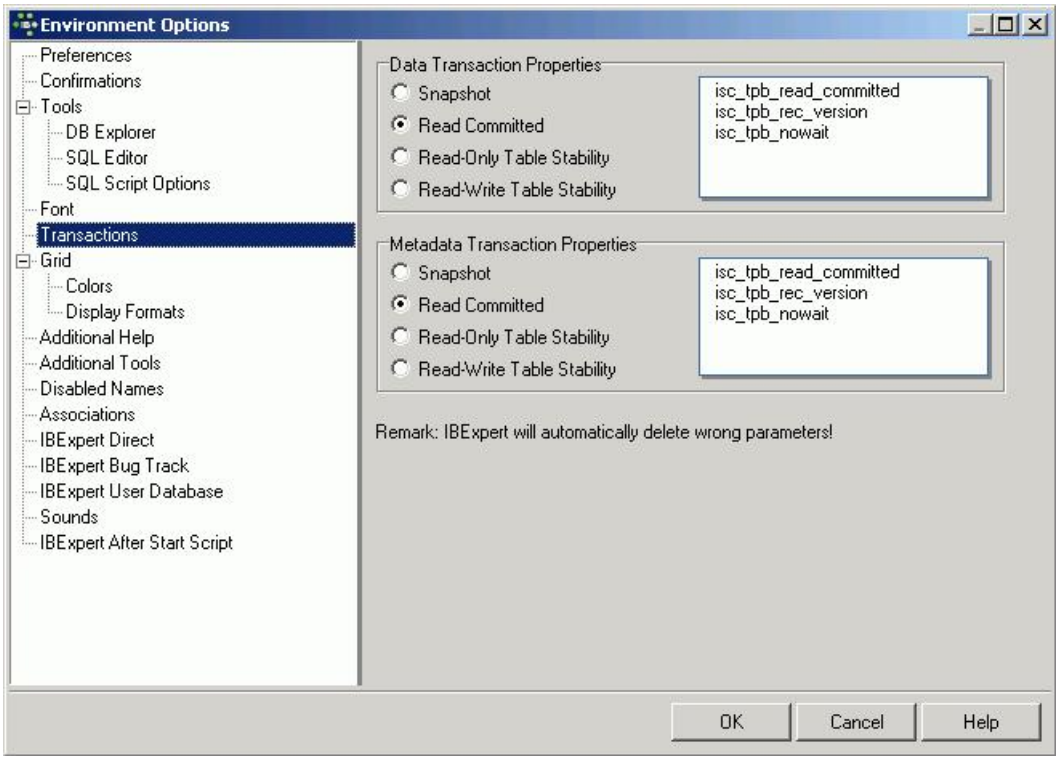
Font

Here it is possible for the user to specify the system (i.e. IBExpert) font name and size. The `Sample Text 12345` displays the specified font as it will appear in IBExpert.



Transactions

Here certain additional data and metadata transaction properties may be defined for the server connection.



These are all InterBase/Firebird API terms, and may be checked as wished.

Data Transaction Properties:

- Snapshot
- Read Committed

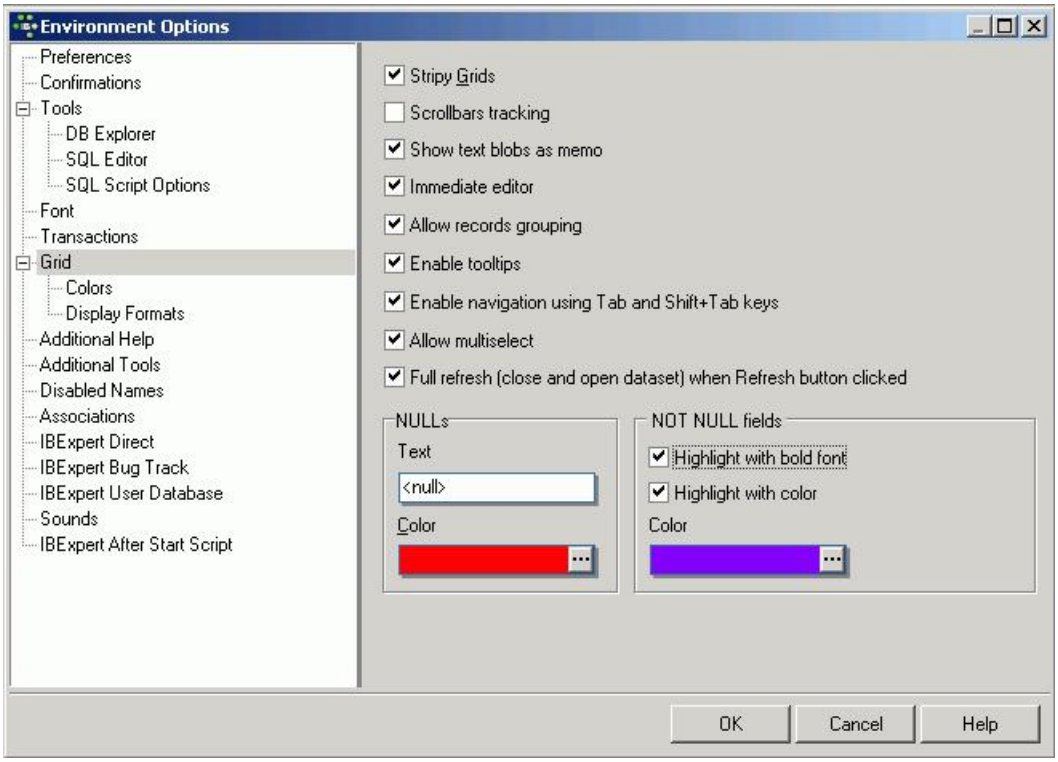
- Read-Only Table Stability
- Read-Write Table Stability

Metadata Transaction Properties:

- Snapshot
- Read Committed
- Read-Only Table Stability
- Read-Write Table Stability

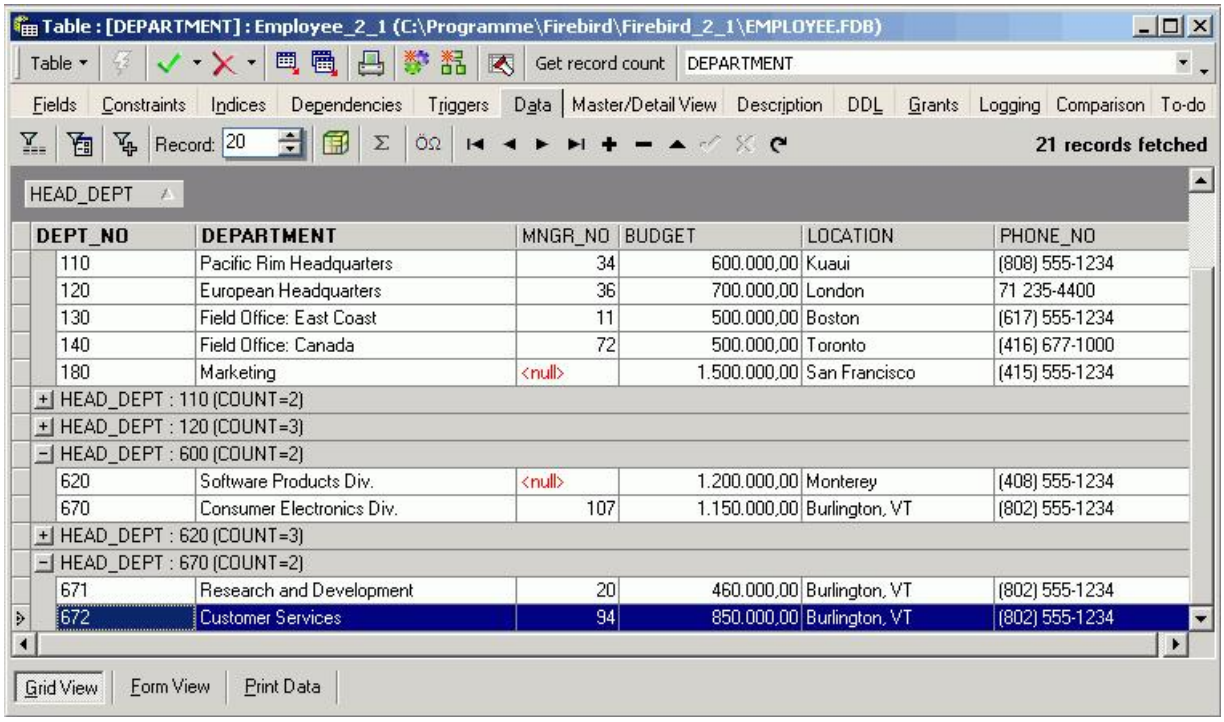
Grid

Here a range of options are available, applicable for all data grids:



Check boxes for the following options:

- **Stripy Grids:** makes reading wide lines of data rows easier.
- **Scrollbars tracking**
- **Show text blobs as memo:** The memo option enables the blob to be easily read by simply focusing the cursor over the blob field.
- **Immediate editor:** Enables immediate editing in the data grid by simply placing the cursor on a field, as opposed to having to first double-click on the field, in order to edit it.
- **Allow records grouping:** When this option is checked, an additional gray bar appears above the column headers over the grid. A column header simply needs to be dragged 'n' dropped into this area, to group by the selected column. A reorganized data view appears, where the group contents can be revealed or hidden, by clicking on the '+' or '-' buttons. Please note that this is not the same as the data grid right-click menu item Group Fields/ Ungroup Fields.



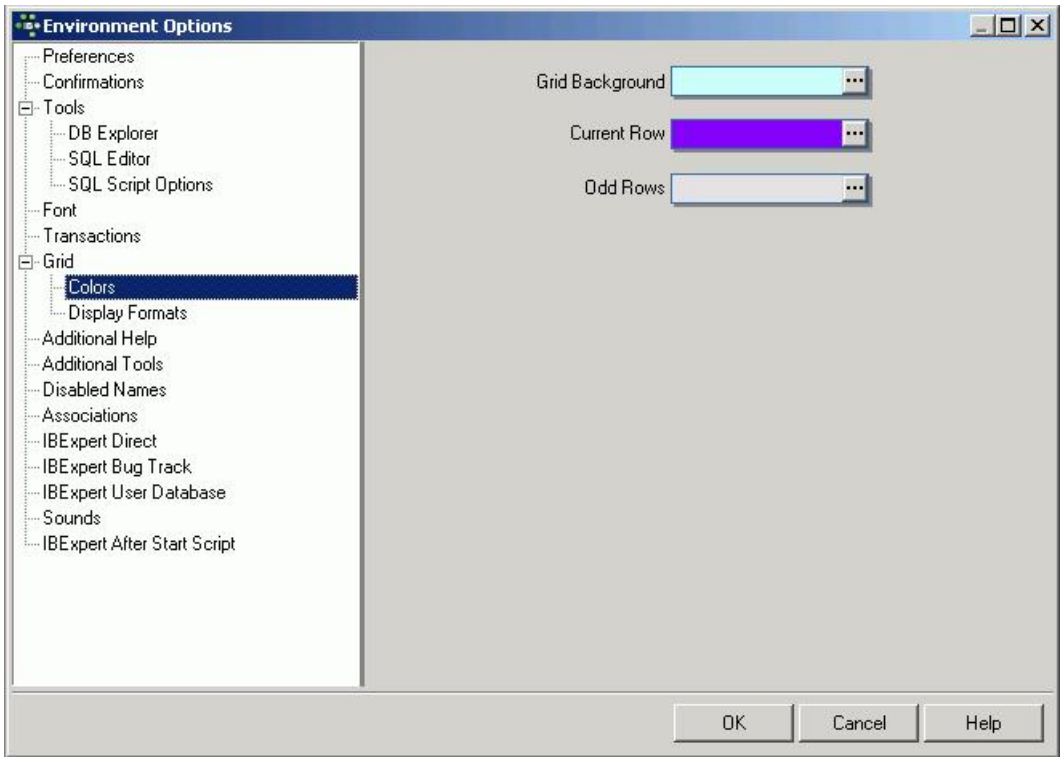
- **Enable tooltips:** when checked, this option displays the full field contents when the cursor is held over a particular field, if the column width is not sufficient to display all information. This is useful, if tables with many columns and long field contents need to be scanned.
- **Enable navigation using [Tab] and [Shift + Tab] keys**
- **Allow multiselect:** allows multiple data sets to be selected for editing (e.g. copying). If this is not checked, it is only possible to select one data set at a time. The change of mode can be recognized by the form/shade of the arrow on the left when pointing at a selected data set.
- **Full refresh (close and open dataset) when Refresh button clicked**

Furthermore it is possible to specify the exact representation of a `NULL` and `NOT NULL` fields. The default value is displayed as <null> (in red). `NOT NULL` fields can be displayed as bold text or be highlighted with color.

Colors

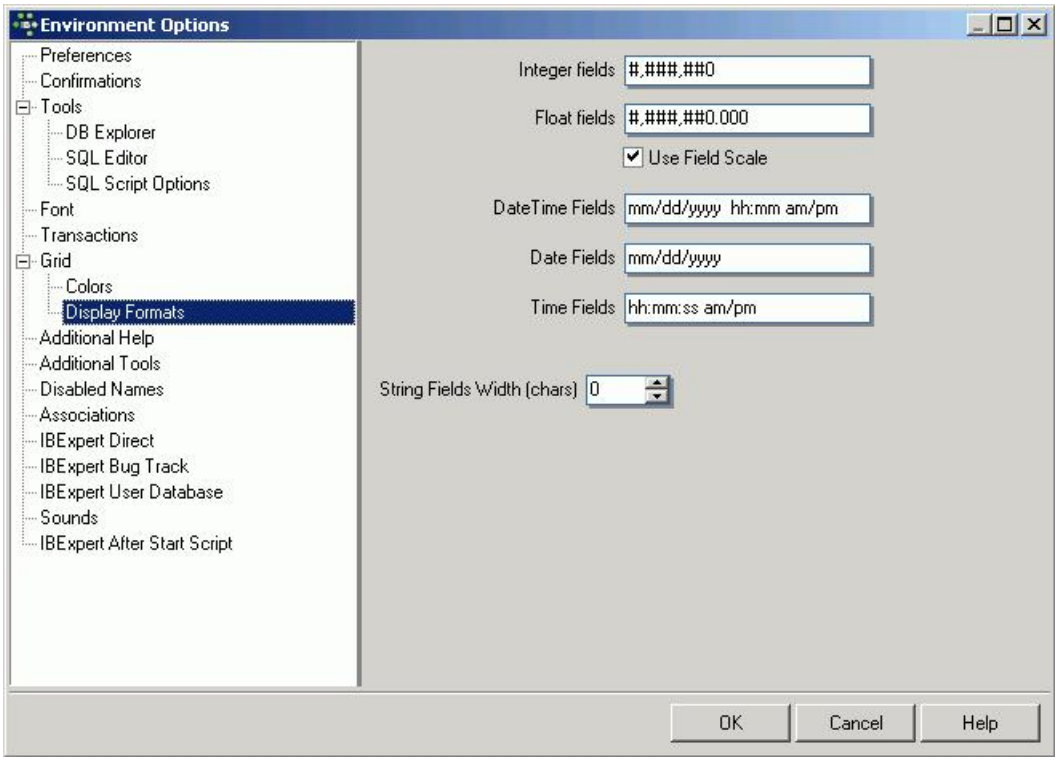
Here the user can specify the colors for different elements in the grids:

- Grid Background
- Current Row
- Odd Rows



Display Formats

These options allow the user to specify the display format in grids for INTEGER, FLOAT, DATE, TIME and DATE/TIME fields.



Further options include a check box option for *Use field scale*, which allows a field definition to override these standard specifications, and an option to specify the *String fields' width* for characters.

The following lists the various date and time formatting options available.

Date Time Formats

The following format allows you to alter the way the date and time is displayed. Please note that this does not alter the way this information is stored, only the way it is displayed.

Date time format strings specify the formatting of date-time values (such as `TDateTime`) when they are converted to strings. Date time format strings are passed to formatting methods and procedures (such as `FormatDateTime`), and are also used to set certain global variables (such as `ShortDateFormat`).

They are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as `d`), simply format numbers or strings. Other specifiers (such as `/`) refer to local-specific strings from global variables.

In the following table specifiers are given in lower case. Case is ignored in formats, except for the `am/pm` and `a/p` specifiers.

Specifier Displays

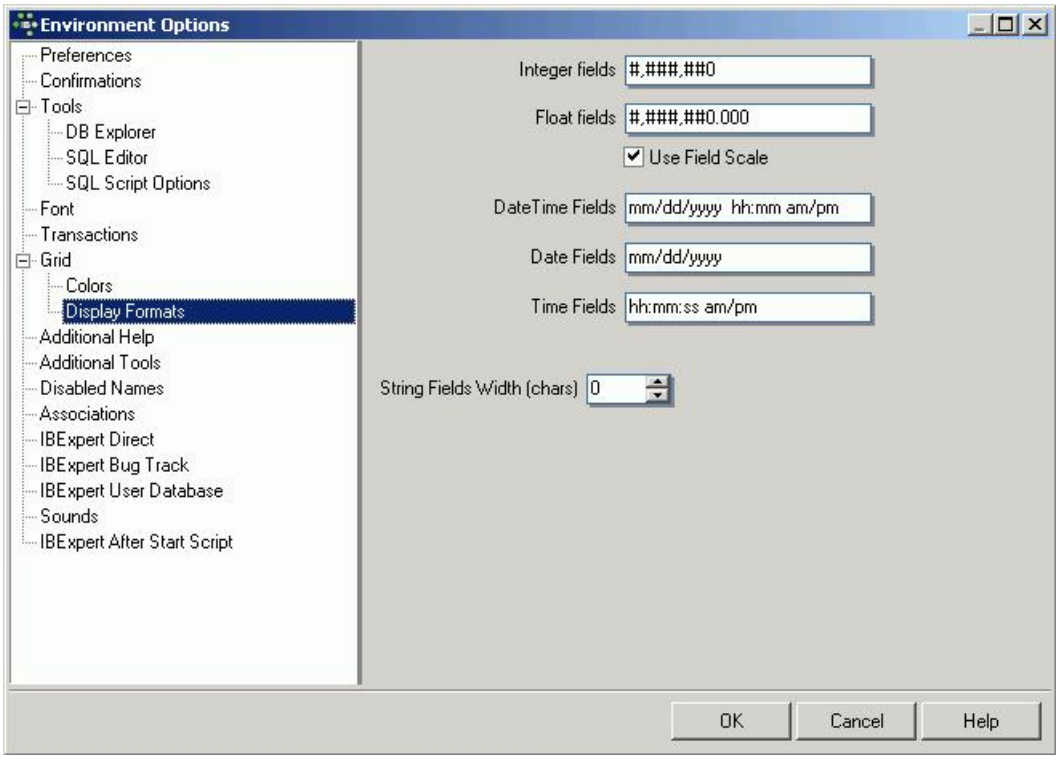
c Displays the date using the format given by the `ShortDateFormat` global variable, followed by the time using the format given by the `LongTimeFormat` global variable. The time is not displayed if the date-time value indicates midnight precisely.
d Displays the day as a number without a leading zero (`1-31`).
dd Displays the day as a number with a leading zero (`01-31`).
ddd Displays the day as an abbreviation (`Sun-Sat`) using the strings given by the `ShortDayNames` global variable.
dddd Displays the day as a full name (`Sunday-Saturday`) using the strings given by the `LongDayNames` global variable.
ddddd Displays the date using the format given by the `ShortDateFormat` global variable.
dddddd Displays the date using the format given by the `LongDateFormat` global variable.
e Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m Displays the month as a number without a leading zero (`1-12`). If the `m` specifier immediately follows an `h` or `hh` specifier, the minute rather than the month is displayed.
mm Displays the month as a number with a leading zero (`01-12`). If the `mm` specifier immediately follows an `h` or `hh` specifier, the minute rather than the month is displayed.
mmm Displays the month as an abbreviation (`Jan-Dec`) using the strings given by the `ShortMonthNames` global variable.
mmmm Displays the month as a full name (`January-December`) using the strings given by the `LongMonthNames` global variable.
yy Displays the year as a two-digit number (`00-99`).
yyyy Displays the year as a four-digit number (`0000-9999`).
h Displays the hour without a leading zero (`0-23`).
hh Displays the hour with a leading zero (`00-23`).
n Displays the minute without a leading zero (`0-59`).
nn Displays the minute with a leading zero (`00-59`).
s Displays the second without a leading zero (`0-59`).
ss Displays the second with a leading zero (`00-59`).

z Displays the millisecond without a leading zero (`0-999`).
zzz Displays the millisecond with a leading zero (`000-999`).
t Displays the time using the format given by the `ShortTimeFormat` global variable.
tt Displays the time using the format given by the `LongTimeFormat` global variable.
am/pm Uses the 12-hour clock for the preceding `h` or `hh` specifier, and displays `am` for any hour before noon, and `pm` for any hour after noon. The `am/pm` specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p Uses the 12-hour clock for the preceding `h` or `hh` specifier, and displays `a` for any hour before noon, and `p` for any hour after noon. The `a/p` specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm Uses the 12-hour clock for the preceding `h` or `hh` specifier, and displays the contents of the `TimeAMString` global variable for any hour before noon, and the contents of the `TimePMString` global variable for any hour after noon.
/ Displays the date separator character given by the `DateSeparator` global variable.
: Displays the time separator character given by the `TimeSeparator` global variable.
'xx'/"xx" Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

Example

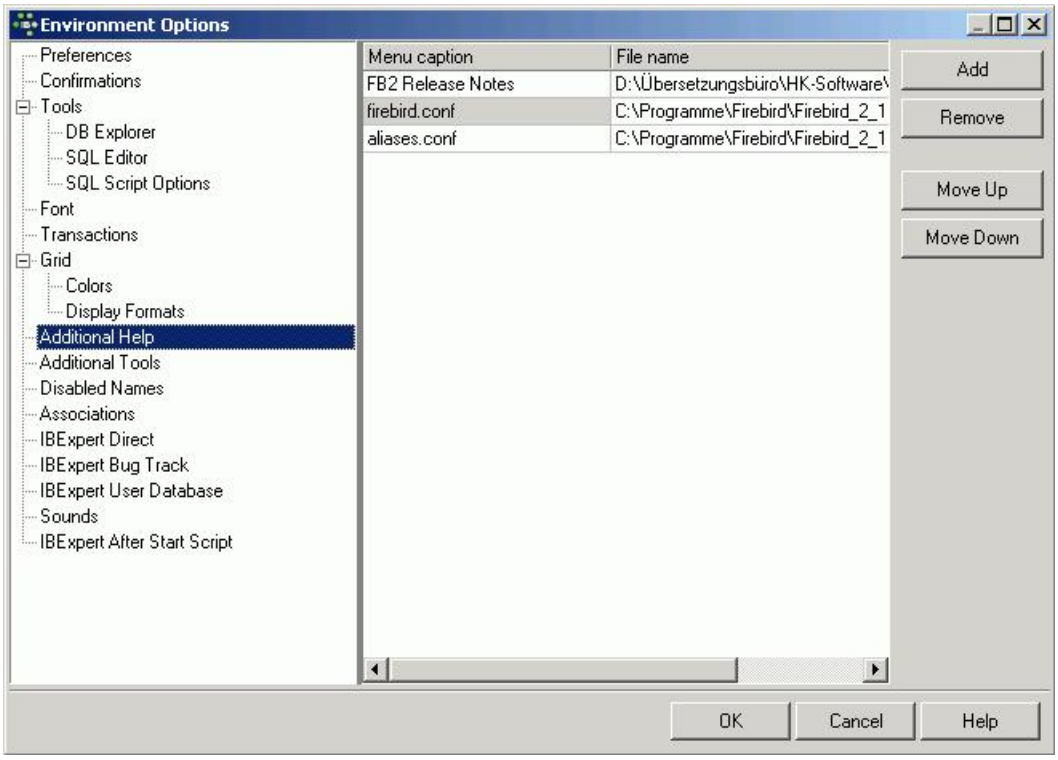
To format the date as month, day, year and the time as am or pm, simply enter the following in Display Formats:

Simply alter *DateTime Fields* to: **mm/dd/yyyy hh:mm am/pm**: and *Time Fields* to **hh:mm:ss am/pm**



Additional Help

The Additional Help dialog allows the user to add certain additional help files. This is particularly useful for incorporating the help files of third party components, installed in the IBExpert PlugIns menu.



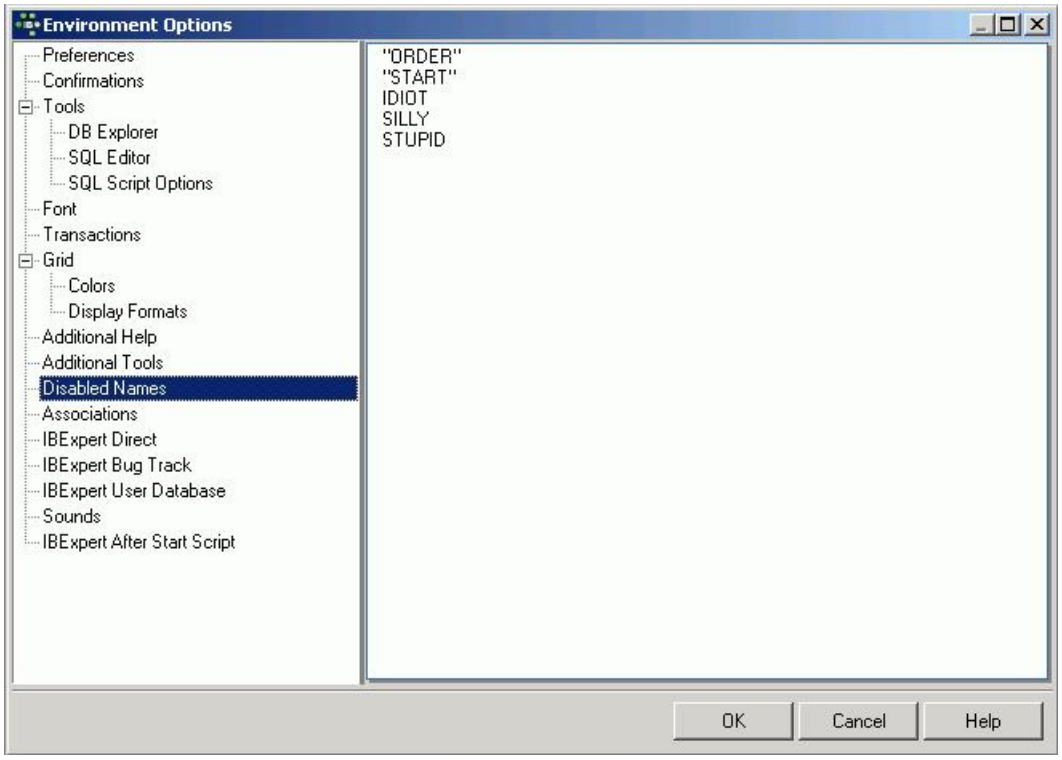
An additional menu item is automatically inserted in the IBExpert Help menu, for each of these help files.

Additional Tools

The Additional Tools dialog allows the user to add certain additional third party tools. For more details, please refer to the IBExpert PlugIns Menu.

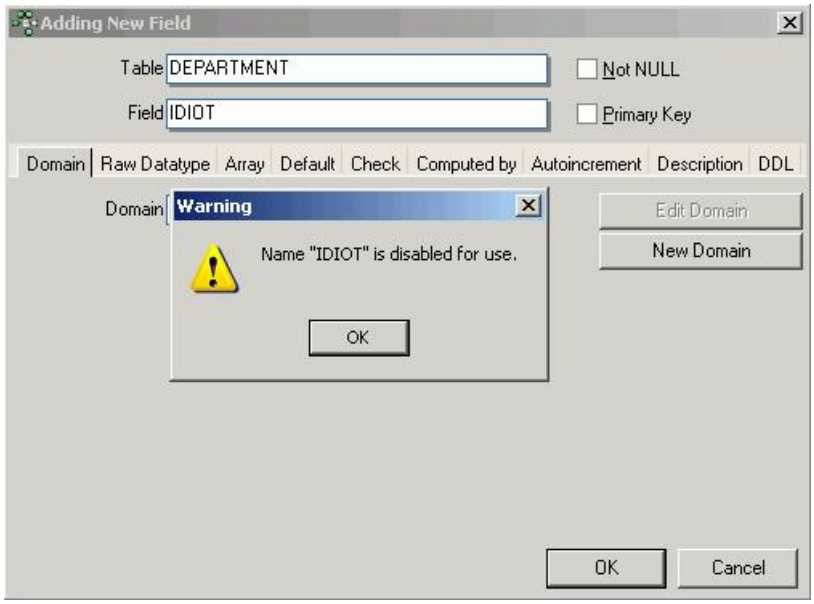
Disabled Names

This page can be used to define a list of disabled object names.



IBExpert refers to this list, when new database objects (and fields) are created, and publishes a warning if the new name corresponds to any name in this list.

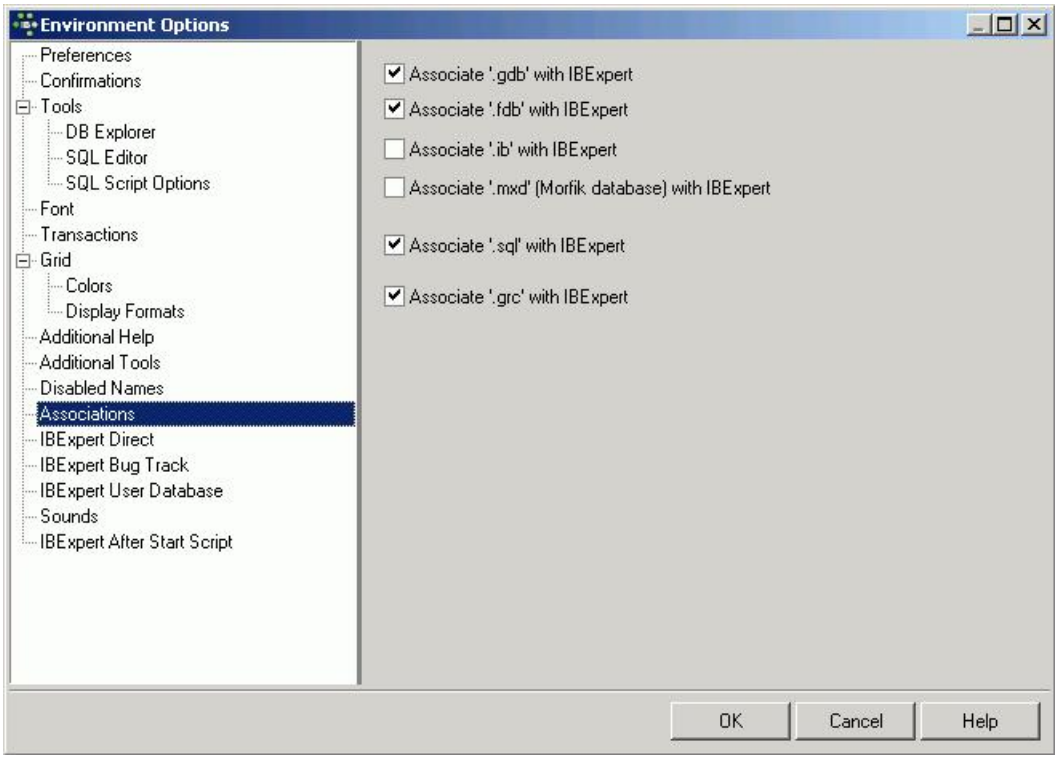
Names that should be avoided because they are Firebird keywords, such as ORDER and START (Firebird 2.1) do not need to be added to this list, as they are automatically generated by IBExpert with the necessary quotation marks ("). If you wish to avoid metadata names in quotation marks, these words need to be typed with the quotation marks in the list of disabled names (see illustration above).



Associations

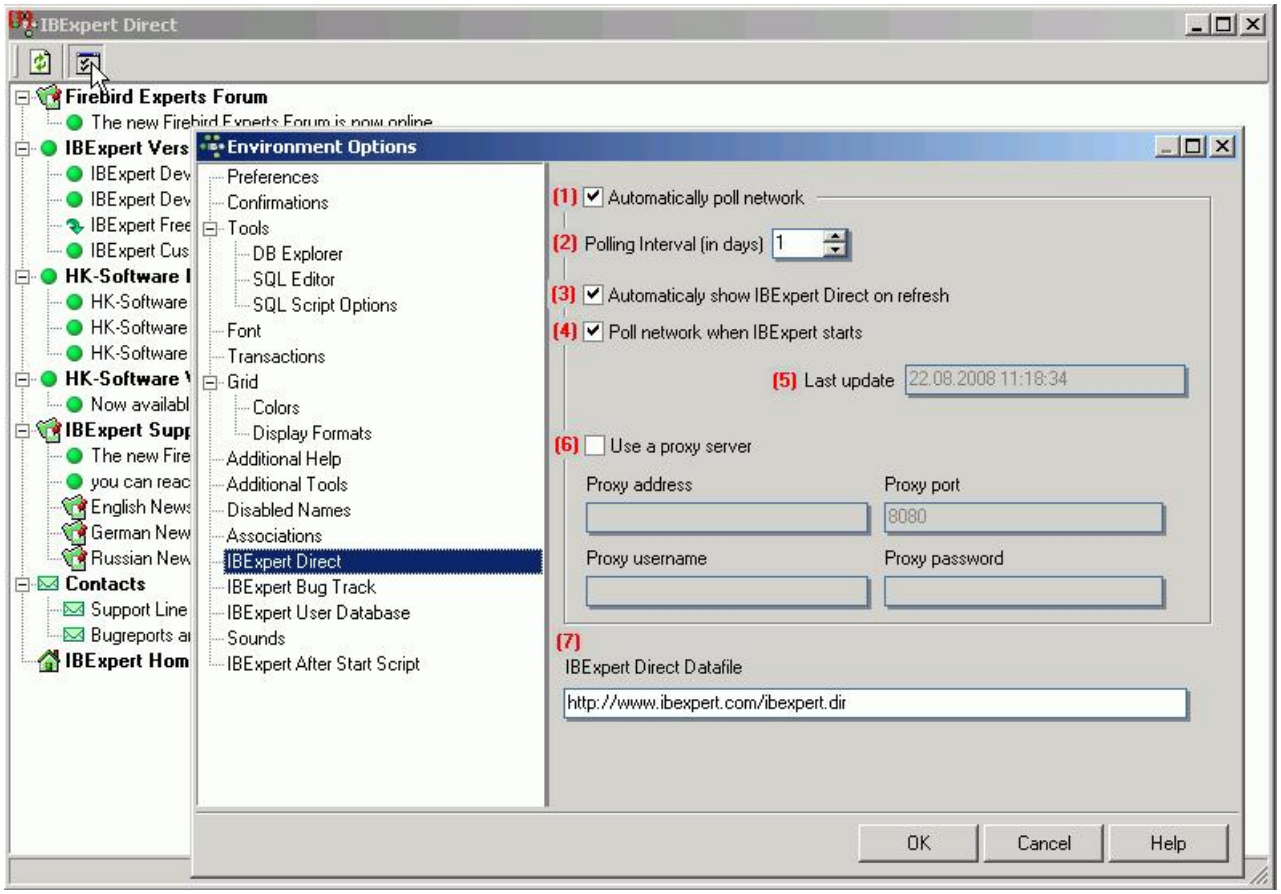
This dialog is important, to specify which file types IBExpert should recognize and associate with the InterBase/Firebird database. The check list includes the following suffixes:

- .GDB
- .FDB
- .IB
- .SQL
- .GRC



IBExpert Direct

The IBExpert Direct dialog allows the user to specify a number of options concerning this IBExpert menu item found in the Help menu. The IBExpert configuration window can be started either from the IBExpert Options menu item, Environment Options or alternatively directly from the IBExpert Help menu item IBExpert Direct, using the respective icon:



The options available include the following:

(1) Automatically poll network: this is recommended, as IBExpert Direct is an important information source, informing all users of news concerning IBExpert, such as new versions, documentation, downloads, plugins, newsgroups, as well as contact addresses and a direct link to the IBExpert home page, http://ibexpert.net/ibe/.

(2) The **polling interval in days** can be user-specified. Check boxes allow the user to specify whether IBExpert Direct should **(3)** automatically shown on refresh, or whether **(4)** the network should be polled for new items, each time IBExpert is started.

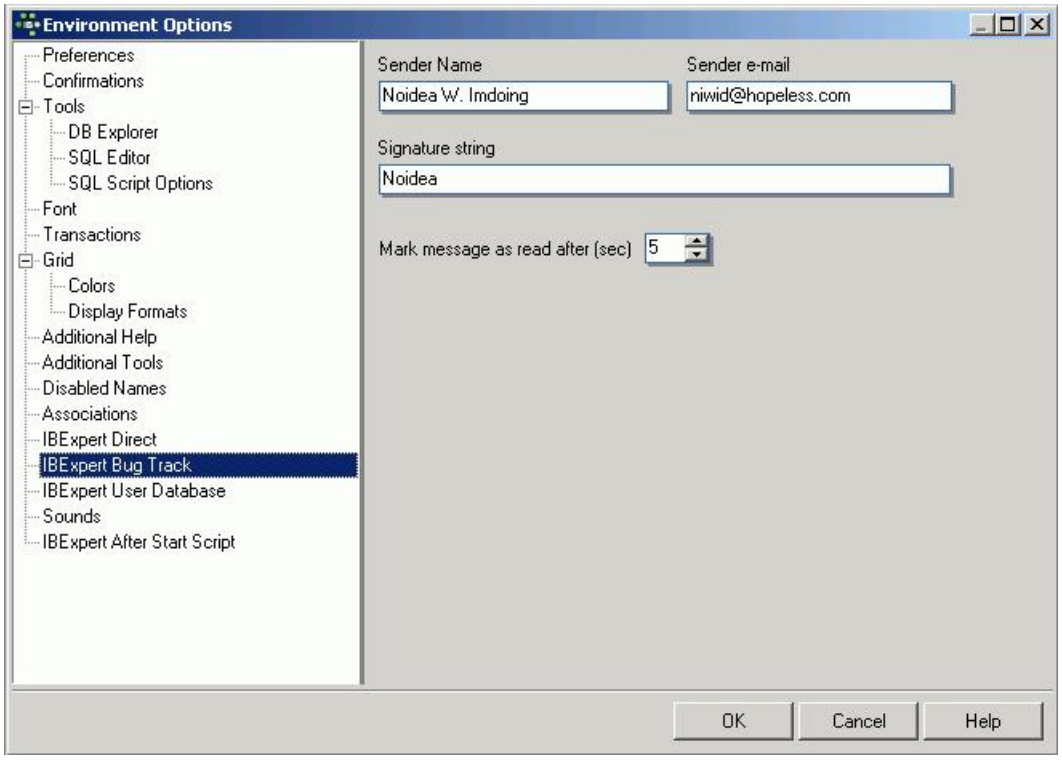
(5) The **Last update** field is purely a display field, showing the last time the network was polled for new IBExpert Direct news items.

(6) It is also possible to specify a proxy server if necessary, with fields for specification of the proxy address, port, user name and password.

(7) The last field displays the IBExpert Direct link address for IBExpert to internally download the file.

IBExpert Bug Track

This option allows the user to specify his signature before posting bugs in the IBExpert Help menu item, Bug Track System.



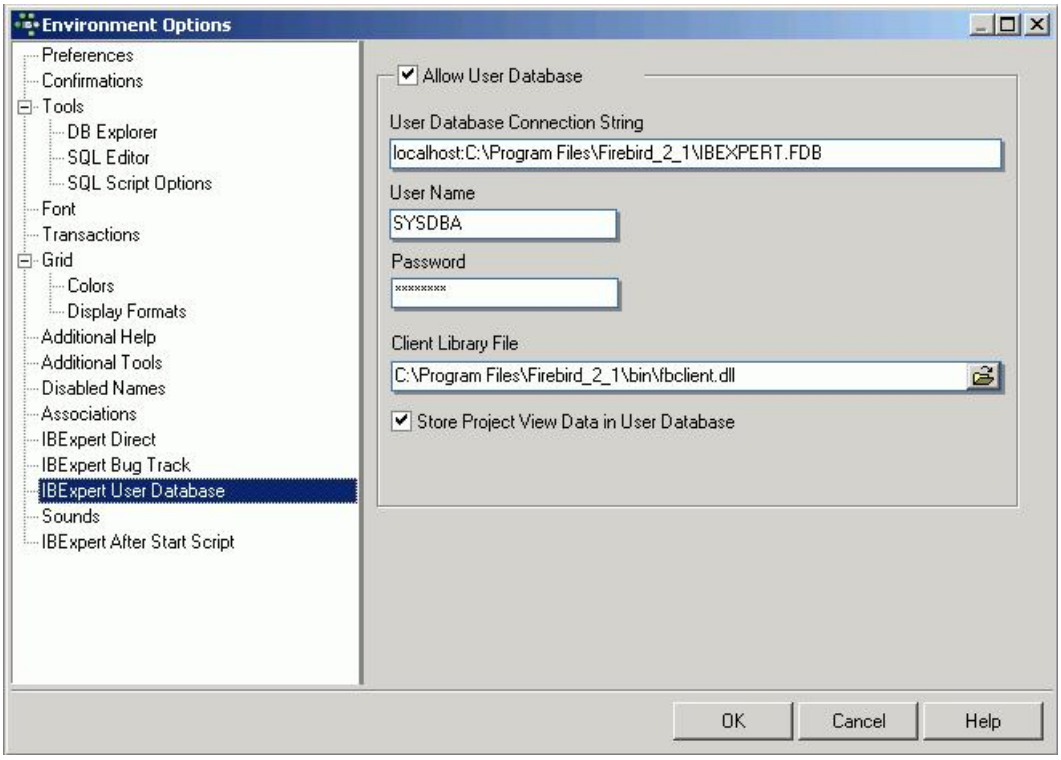
The Bug Track signature requires the following information:

- Sender Name
- Sender email
- Signature string

The option *Mark message as read after n(sec)* applies to all bug messages listed in the Bug Track System.

IBExpert User Database

The complete IBExpert configuration and work is stored here in the IBExpert User Database. The user database should always be used for your main storage for security reasons.



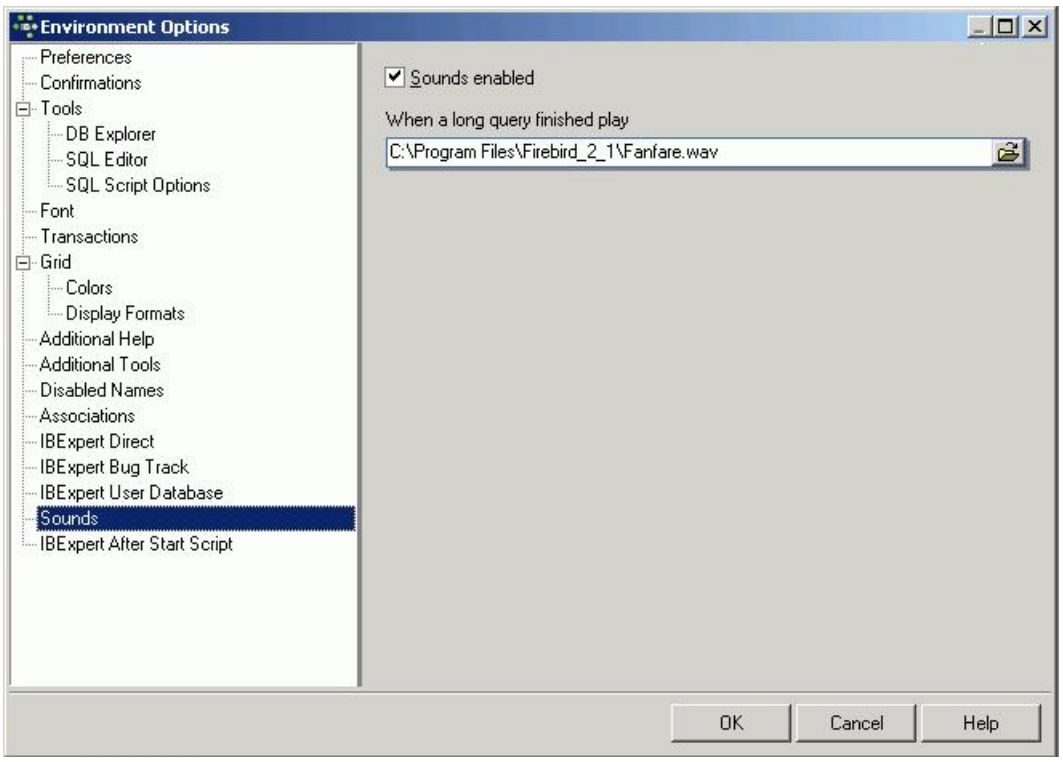
The following information is required in order to create a new user database. After checking the *Allow User Database* checkbox the following fields need to be completed:

- **User Database Connection String:** e.g. If you're using local server the connection string should be as follows: `localhost:c:\mydata\ibexpert.fdb` (for a TCP/IP protocol) or just `c:\mydata\ibexpert.fdb` (for a local protocol).
- **User Name** (default: `SYSDBA`).
- **Password** (`masterkey`).
- **Client Library File** including path and file name.
- **Check box:** *Store Project View Data in User Database*

The user database can then be created and initialized using the *Create and Init User Database* button, and then registered using the IBExpert Database menu item, Register Database.

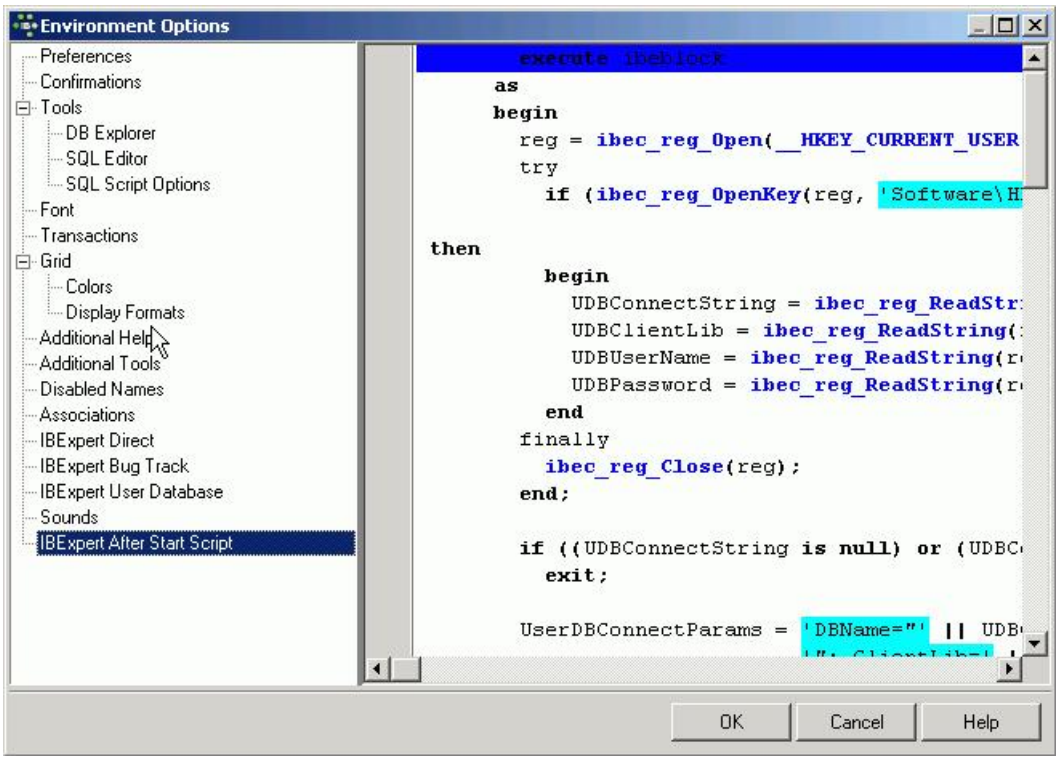
Sounds

Using the *Sounds* preference, it is possible to specify a `.WAV` file to announce the end of a time-consuming query.



IBExpert After Start Script

The *IBExpert After Start* Script feature was implemented in IBExpert version 2006.08.12. A script specified here will be executed after IBExpert is started.



The following example illustrates how to use the *After Start Script* to find all database registration records with missing database files (if local access is used), and place them into an individual folder. This only works with the User Database.

```
execute ibeblock
as
begin
  reg = ibec_reg_Open(__HKEY_CURRENT_USER, 0);
  try
    if (ibec_reg_OpenKey(reg, 'Software\HK Software\IBExpert\CurrentData', FALSE))

then
    begin
      UDBConnectString = ibec_reg_ReadString(reg, 'UDBConnectString');
      UDBClientLib = ibec_reg_ReadString(reg, 'UDBClientLib');
      UDBUserName = ibec_reg_ReadString(reg, 'UDBUserName');
      UDBPassword = ibec_reg_ReadString(reg, 'UDBPassword');
    end
  finally
    ibec_reg_Close(reg);
  end;

  if ((UDBConnectString is null) or (UDBConnectString = '')) then
    exit;

  UserDBConnectParams = 'DBName="' || UDBConnectString ||
                        '"; ClientLib=' || UDBClientLib ||
                        '; User=' || UDBUserName ||
                        '; Password=' || UDBPassword ||
                        '; Names=UNICODE_FSS; SqlDialect=1';

  UserDB = ibec_CreateConnection(__ctInterBase, UserDBConnectParams);
  try
    ibec_UseConnection(UserDB);

    -- Looking for missing database files (for local databases only)
    MissingFiles = null;
    i = 0;

    PropIni = ibec_ini_Open('');
    try
      for select id, props from databases
          where (rec_type = 0) and (props containing 'Protocol=3')
          into :id, :props
      do
      begin
        Props = '[DB]' || ibec_CRLF() || Props;
        ibec_ini_SetStrings(PropIni, Props);
        Props = ibec_ini_GetStrings(PropIni);
        DBFile = ibec_ini_ReadString(PropIni, 'DB', 'DBNames', '');
```

```
        if ((DBFile <> '') and (not ibec_FileExists(DBFile))) then
        begin
          MissingFiles[i] = ID;
          i = i + 1;
        end;
      end;
    finally
      ibec_ini_Close(PropIni);
    end;

    if (i > 0) then
    begin
      ParentID = null;
      select id from databases
      where (rec_type = 1) and (props containing 'FolderCaption=***MISSING DATABASE FILES***')
      into ParentID;
      if (ParentID is null) then
      begin
        ParentID = gen_id(GEN_DATABASE_ID, 1);
        insert into databases (ID, PARENT_ID, REC_TYPE, DB_ORDER, PROPS)
           values (:ParentID, 0, 1, 0, 'FolderCaption=***MISSING DATABASE FILES***');
        commit;
      end

      for i = 0 to ibec_High(MissingFiles) do
      begin
        id = MissingFiles[i];
        update databases set parent_id = :ParentID where id = :id;
        commit;
      end
    end;

  finally
    ibec_CloseConnection(UserDB);
  end;
end
```

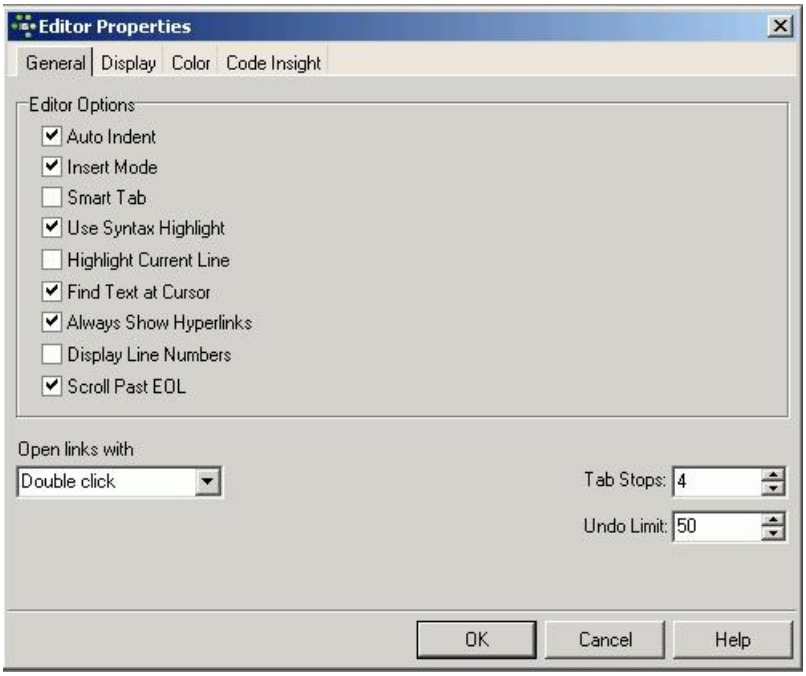
See also:
IBEBlock

Editor Options

Editor Options can be found in the IBExpert Options menu. It opens the *Editor Properties* dialog, which enables the user to organize and customize IBExpert editors as he wishes. It is possible, for example, to set certain defaults, or alter the font or colors, customize code completion etc.

General

The first page in the *Editor Properties* dialog is the *General* page, which offers the following options:



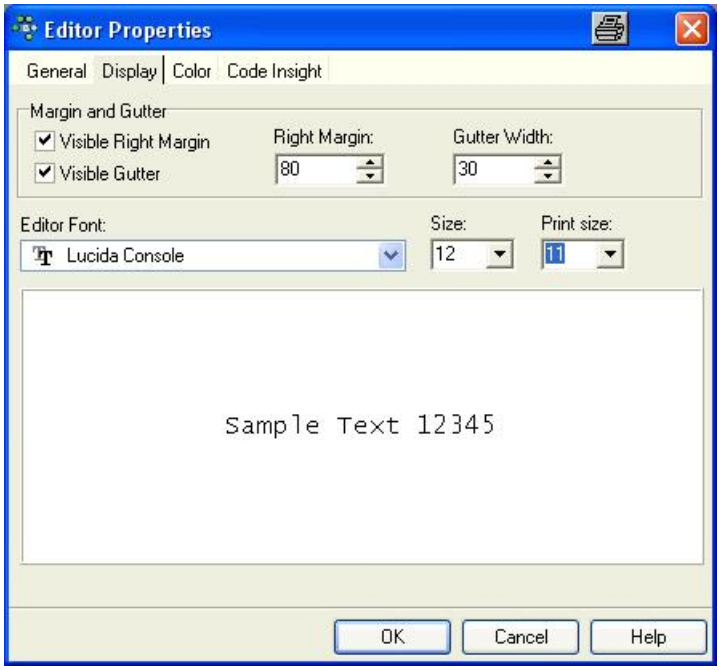
- **Auto Indent:** (default) this automatically indents code when editing SQL script; each new indention identical to the previous. The tab (= tabulator) length can be specified using the lower right Tab Stops counter (default = 4 characters).
- **Insert Mode:** (default) inserts text at the cursor without overwriting existing text. When disabled (i.e. when unchecked), the so-called typeover mode is activated, i.e. the text at the cursor is overwritten. It is possible to use the [Ins] key to switch the insert mode on and off in the code editor, without having to alter the default.
- **Smart Tab:** this automatically limits the tab stop lengths to the length of the previous line.
- **Use Syntax Highlight:** (default) enables highlighted syntax in the object editor window. To set highlighting options, please refer to Editor Options / Color.
- **Highlight Current Line:** useful for orientation in long scripts.
- **Find Text at Cursor:** (default)searches automatically for the word, where the cursor happens to be standing when starting the IBExpert Edit / Find menu item (see also Search - [CTRL + F]). This saves having to mark the word first, or type in the text to be searched for each time.
- **Always Show Hyperlinks:** (default) displays hyperlinks in SQL script as green underlined text (unless altered by the user under Editor Options / Color). It can be opened by double-clicking or single-clicking (user-defined; see *Open links with* below).
- **Show Lines Number:** useful when working with long scripts. This option displays line numbers in the gutter in the editor window. A gutter is automatically inserted, even if it has been unchecked on the *Display* page (please refer to Editor Options / Display).
- **Scroll past end of line:** (default) when this is not checked, the cursor jumps to the beginning of the next line automatically when it has reached the end of the text. If this option is checked, the cursor continues to travel to the right, even after the end of the text has been reached.

Furthermore it is possible to specify the following:

- **Open links with:** double click (default) or single click.
- **Tab Stops:** defines the tab length (see above).
- **Undo limit:** specifies the maximum number of keystrokes, that can be undone (default = 50).

Display

The *Display* page allows the user to specify certain visual editor properties.



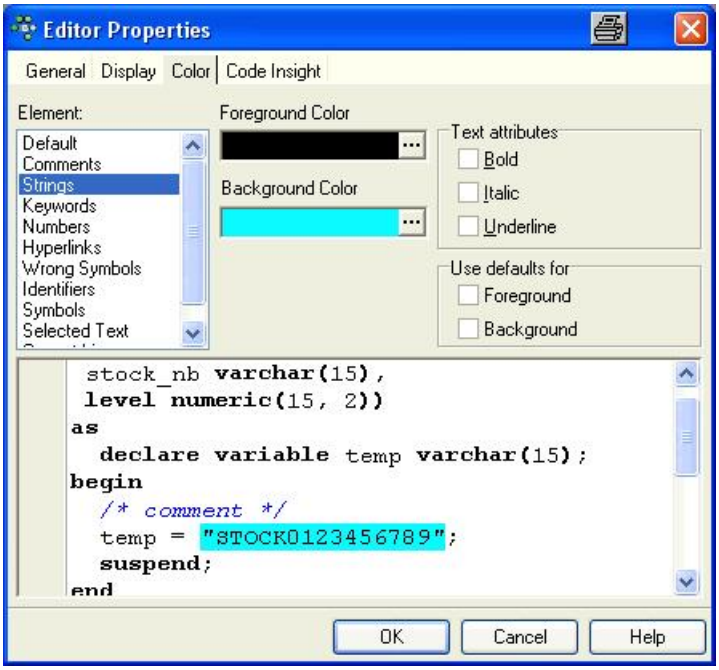
The options available here include:

1. **Margin (= right margin) and Gutter (= inner or left margin):**
   - Visible Right Margin and Gutter (check box option to blend margins in or out)
   - User specification of right margin position and gutter width (in characters).

   Note: checking the "Show Lines number" box on the General page automatically inserts a gutter, even if it is not checked here.
2. **Editor Font:**
   User specifications include font, size and print size (with sample text preview). The advantage here is that it is possible to specify a larger or smaller display font size than the print font size.

Color

The *Color* page allows the user to specify colors and text attributes for a range of elements:

%center 

The range of elements includes the following:

- default
- comments
- strings,
- keywords
- numbers
- hyperlinks
- wrong symbols
- identifiers

- symbols
- selected text
- current line
- double-quoted string (new to IBExpert version 2003.11.6.1)
- conditional directive (new to IBExpert version 2003.11.6.1)
- variable
- IBEBlock procedure/function (new to IBExpert version 2005.02.12.1)
- Pairing brackets (new to IBExpert version 2005.09.25)

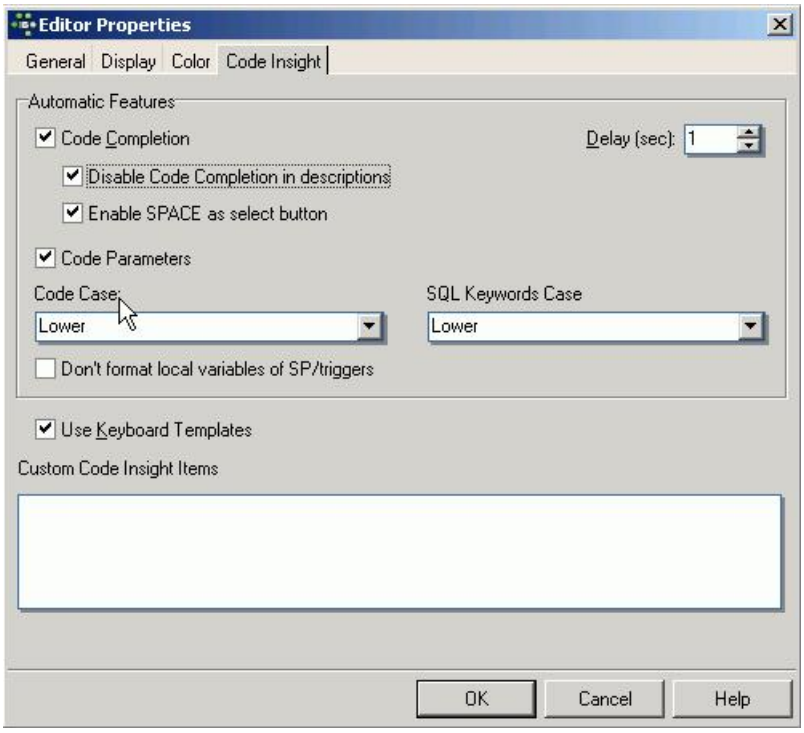
The following properties can be specified for the above elements:

- **Foreground Color:** determines the color of the selected element in the foreground (usually text).
- **Background Color:** determines the color of the selected element in the background (generally used to highlight text).
- **Text attributes:** includes specification of bold, italic and/or underline.
- **Use defaults for:** allows the default to be rapidly specified for both the foreground and background colors for a selected element.

The text preview panel displays the elements as they have been specified, allowing the user to approve or alter his choice, or return to the default settings using the Use defaults for *Foreground/Background* check boxes.

Code Insight

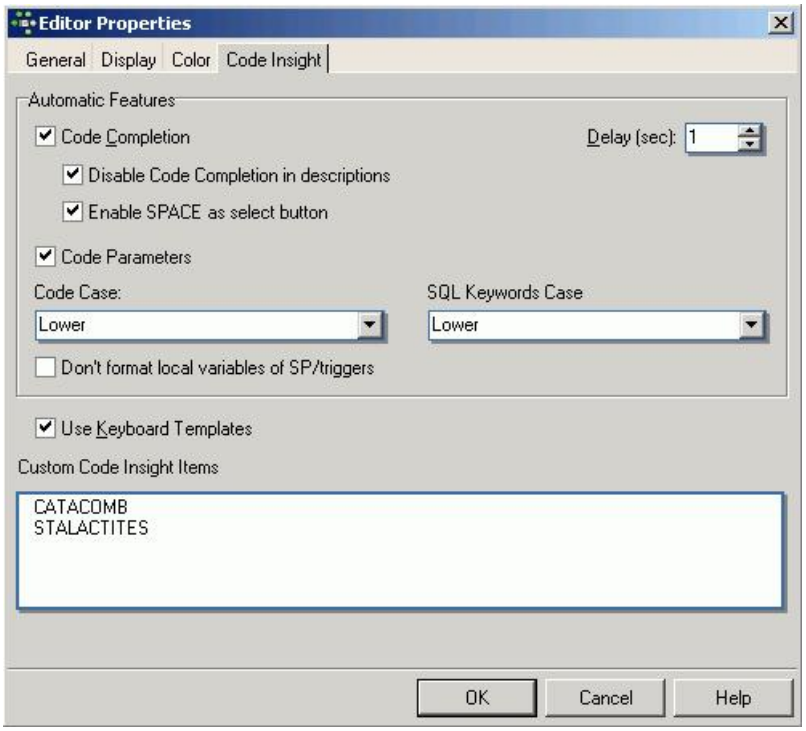
The Code Insight page offers a number of options related to the IBExpert automatic code completion:



These include:

- **Code Completion:** here the user can specify, whether code completion should be active or not.
- **Disable Code Completion in Descriptions:** a new feature in IBExpert version 2005.01.12.1, allowing the user to disable code completion while editing an object description.
- **Enable SPACE as select:** a new feature in IBExpert version 2006.08.12, this option can be activated to allow the SPACE bar to work as the [ENTER] key, i.e. it inserts selected items from the Code Insight list into the code.
- **Code Parameters:** this is a very useful option when active. For example when working with procedures, a list of all necessary input parameters automatically appears, and when one or more parameters have already been specified, the next parameter required appears in bold type. Since IBExpert version 2003.11.6.1 the list of fields to be inserted is now displayed when the `VALUES` part of an `INSERT` statement is typed.
- **Delay (sec):** Delay in seconds before the code completion pop-up list appears with a list of one or more possible suggestions (default value is 1 second).
- **Code Case:** user specification of the words (e.g. object names, field names) inserted automatically by code insight: either lower (default), upper, first upper or name case.
- **Code Case and SQL Keywords Case:** user specification of the SQL keywords inserted automatically by code insight: either lower (default), upper, first upper or name case. There is also a check option to disable formatting of local variables when working with stored procedures and triggers.

It is also possible to specify whether keyboard templates (for faster typing of regularly used words or expressions) should be used, and the *Custom Code Insight Items* display panel displays those items, specified by the user.

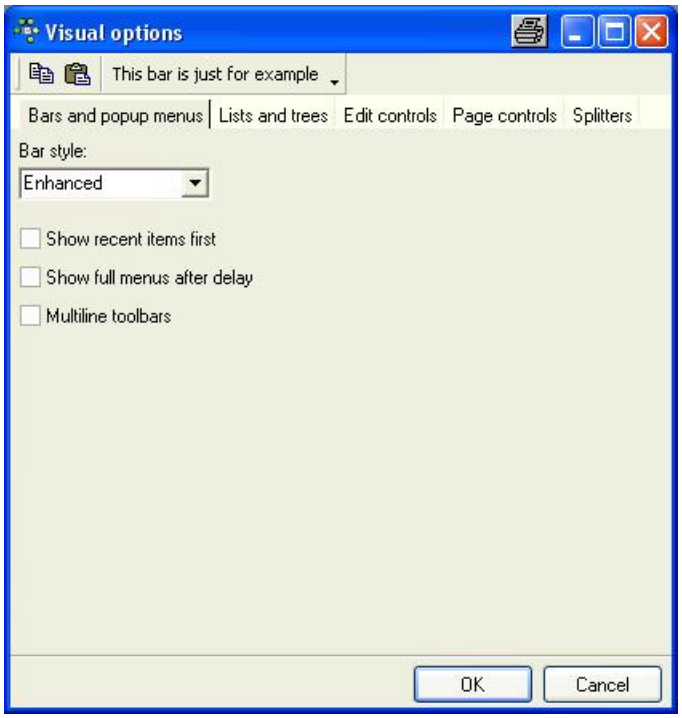

Editor Properties
General
Display
Color
Code Insight
Automatic Features
Code Completion
Delay (sec):
1
Disable Code Completion in descriptions
Enable SPACE as select button
Code Parameters
Code Case:
Lower
SQL Keywords Case
Lower
Don't format local variables of SP/triggers
Use Keyboard Templates
Custom Code Insight Items
CATACOMB
STALACTITES
OK
Cancel
Help

Visual Options

Visual Options can be found in the IBExpert Options menu. It opens the Visual Options Editor, which enables users to customize the IBExpert interface. It is possible, for example, to specify the behavior of pop-up menus, the appearance of border and button styles, and even of splitters.

Bars and Pop-up Menus

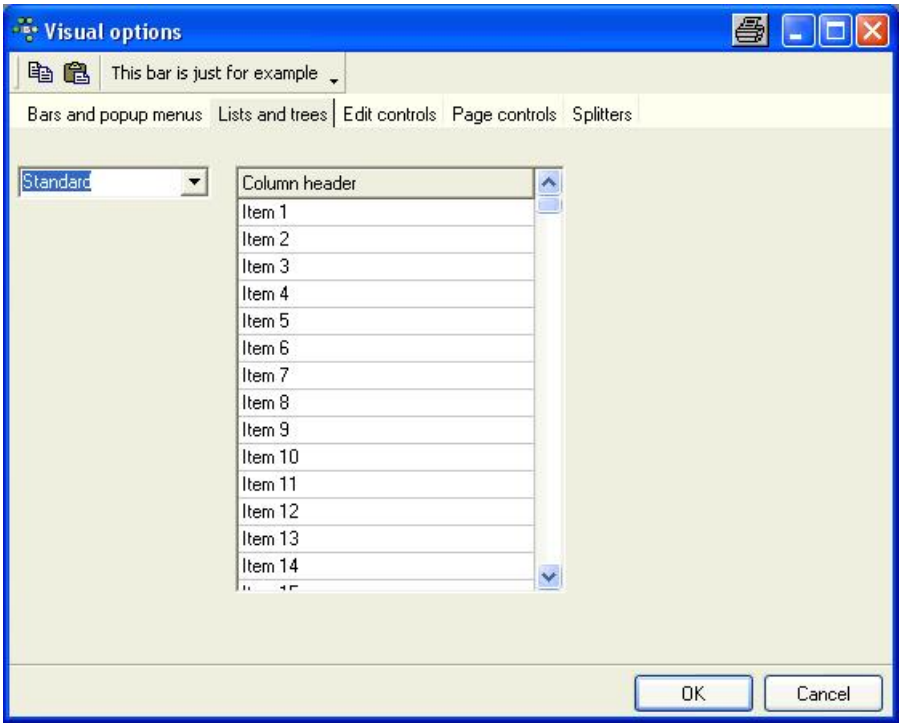
The first tab in the Visual Options Editor is the *Bars and Pop-up Menus* page, which offers the following options:



- **Bar Style:** the options *Standard*, *Enhanced* or *Flat* may be selected. The visual effects of the selection is immediately visible in the sample toolbar, displayed at the top of the Visual Options dialog.
- **Show recent items first:** reduces those menu items offered in the pull-down list, to those most recently selected by the user.
- **Show full menus after delay:** if one of the most recent menu items is not immediately selected, the full range of menu items is displayed.
- **Multiline toolbars:** allows toolbars to cover more than a single row (which may eventually lead to icons running off the right-hand side of the screen, if too many toolbars are active).

Lists and Trees

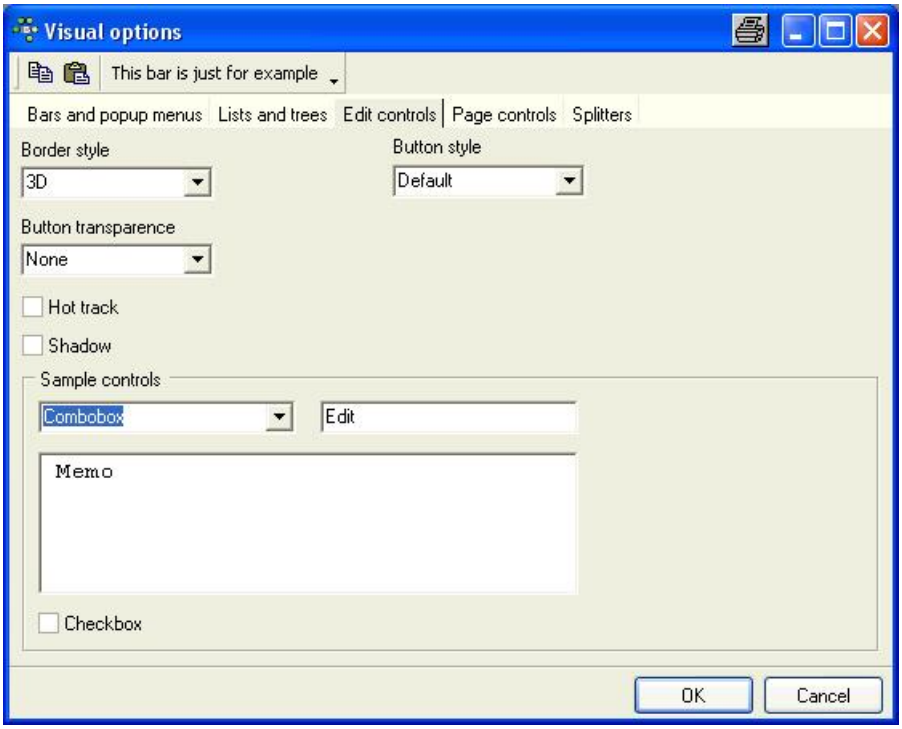
The *Lists and Trees* page offers the following options:



Lists and trees may be displayed in a *Standard*, *Flat* or *Ultraflat* format. The visual effects of the selection can immediately be seen in the example field grid, displayed to the right of the pull-down list.

Edit Controls

The third tab in the Visual Options Editor is the *Edit Controls* page, which offers the following options:

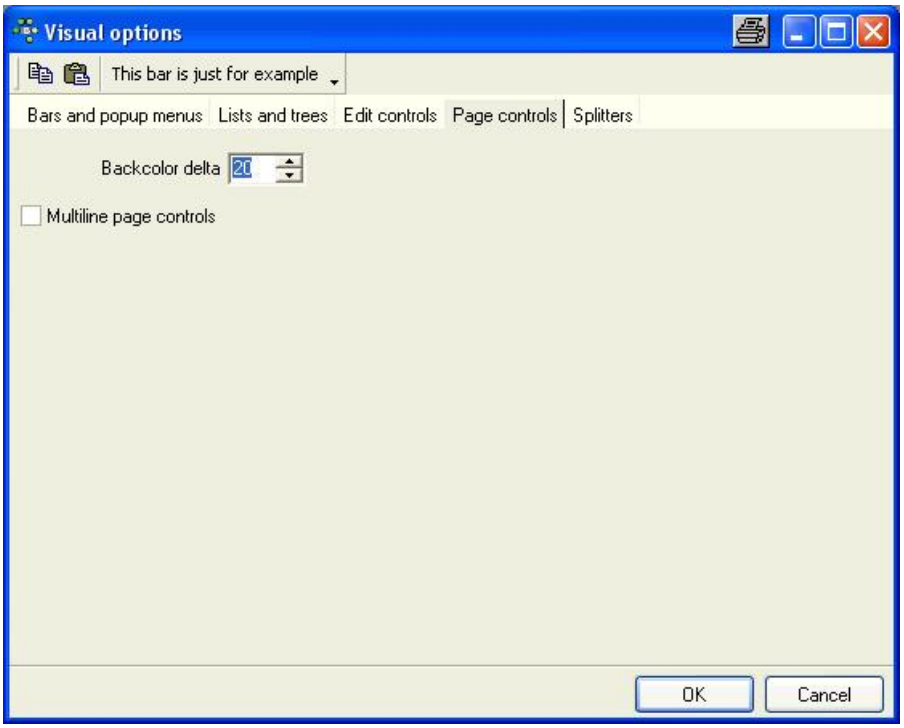


- **Border Style:** options offered include *None*, *Single*, *Thick*, *Flat* and *3D*. The visual effects of the selection is immediately visible in the sample controls panel in the lower area of the window.
- **Button Style:** the options offered here include *Default*, *3D*, *Flat Simple*, *HotFlat*. This changes the style of displaying application buttons. The effect can be previewed in the sample controls (observe the combo box and check box).
- **Button Transparence:** here the options include *None*, *Inactive*, *Always* and *Hide and Inactive*. This alters the appearance of transparent buttons. The effects can be viewed in the sample controls (observe the combo box).
- **Hot Track:** activating this option causes boxes and buttons to be highlighted with a 3D effect, when the mouse is focused over it. The effect only be previewed on all sample controls, if the *Border Style None* has been selected. Otherwise this effect can only be observed on the combo box.
- **Shadow:** this option places a shadow effect around boxes. The sample controls preview shows the effect of this.

The sample controls panel displays a preview of how a pull-down list (combo box), edit field, memo panel/window and check box appear, as specified by the user.

Page Controls

The *Page Controls* page offers the following options:

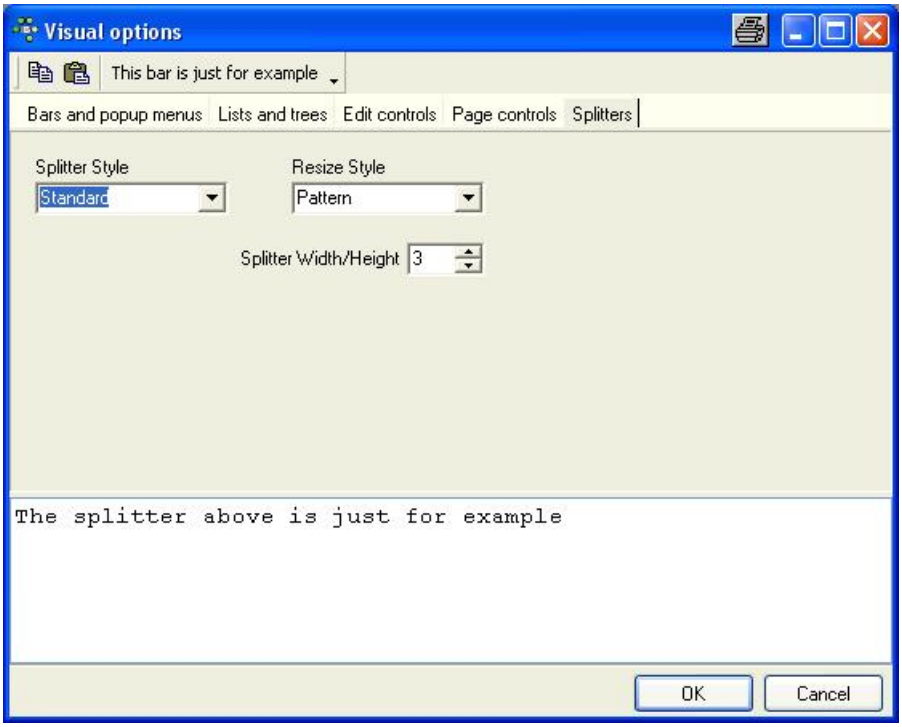


- **Backcolor delta:** this option alters the contrast shade of those page tabs currently in the background. The default value is 20; any changes to this value can be previewed immediately by observing the Visual Options Editor's own page tabs.
- **Multiline page controls:** when checked, this options allows page tabs (or page controls) to be placed over more than one line. This saves the user the necessity of sliding from left to right, in order to find the page he needs. The effect of this option can most easily be viewed in the DB Explorer. Usually the DB Explorer width is limited, in order to allow sufficient space in the main working window. It is therefore often the case that only a small number of the DB Explorer page tabs are visible, and it is necessary to move from left to right before opening, for example, the Windows page. Using this option, the page tabs are displayed over two rows, enabling the user to simply click on the page he needs.

Splitters

A splitter is a moveable line, dividing a child window or editor into two panels.

The *Splitters* page enables the user to specify the appearance of all IBExpert splitters:

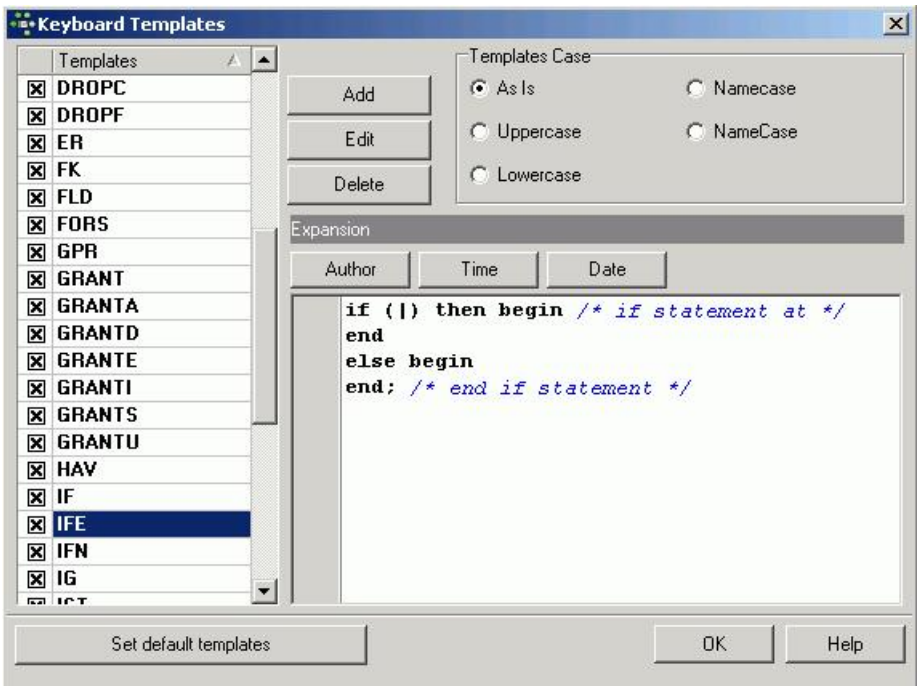


Available options include the following:

- **Splitter Style:** the options offered here include *Standard* and *Netscape*. The Netscape style includes a centered strip (if the splitter width is sufficient, directional arrows are visible). The user simply needs to click on this strip to move the splitter up or down (or left or right if the splitter is vertical), thereby reducing the size of one panel or window and simultaneously increasing the size of the second panel or window. It is also possible to manually adjust the splitter position using drag 'n' drop. When using the standard style the only way to move the splitter is by using drag 'n' drop.
- **Resize Style:** the options offered here include *None*, *Line*, *Update* and *Pattern*. The effects of these options can be viewed by dragging and dropping the sample splitter.
- **Splitter Width/Height:** the effects of any alterations here can be viewed immediately on the sample splitter, displayed in the lower half of this page.

Keyboard Templates

This can be found under the IBExpert Options menu. It can be used to customize and standardize typing abbreviations for frequently used typical statements, thus increasing efficiency.



For example scroll down to **IFE**. The full phrase can be viewed and, if needed, altered as wished in the *Expansion* panel. The pipe, `|`, (vertical bar) indicates the cursor position, when the text is inserted in the SQL Editor.

After confirming any alterations go back to the SQL Editor Edit page, type `ife` and press the space bar.

It is automatically expanded to the `if ... then ... else ...` statement as defined in the keyboard template; the cursor is automatically positioned as specified.

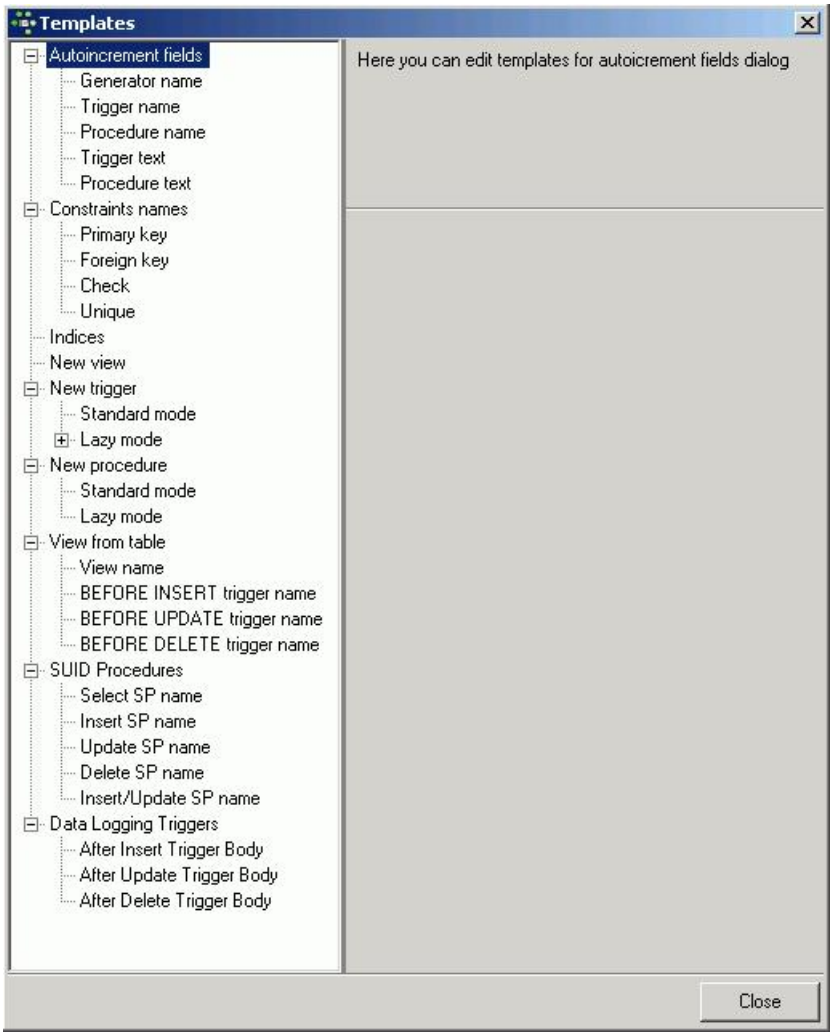
Templates can be added or selected templates edited and deleted as wished. Templates can also be simply deactivated (instead of deleted), by clicking on the flagged checkbox to the left of the template name. To reactivate a deactivated template, simply check the box again.

Further attributes such as *Templates Case* can also be specified in this editor. Available options include *As Is*, *Uppercase*, *Lowercase*, *Namecase*, and *NameCase*.

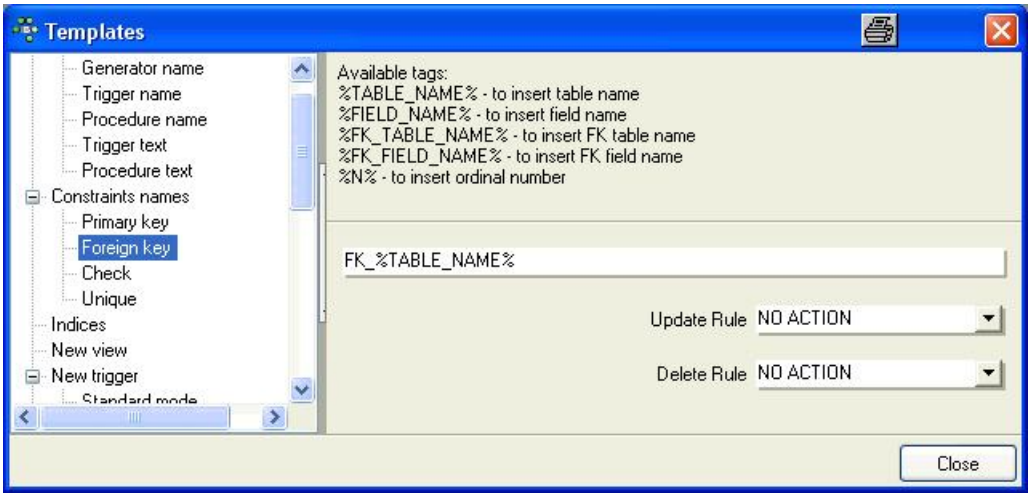
A further feature allows the user to insert author, date and time fields automatically and rapidly, with a simple button click. For example, the abbreviation `ME`, with the expansion `/* #author #date */` (click the *Author* and *Date* buttons to insert the fields, add the comment symbols, done!) results in an simple documentation comment at the beginning of all SQLs listing author and date (i.e. `/* SYSDBA 08/07/2003 */`) simply by typing `ME`!

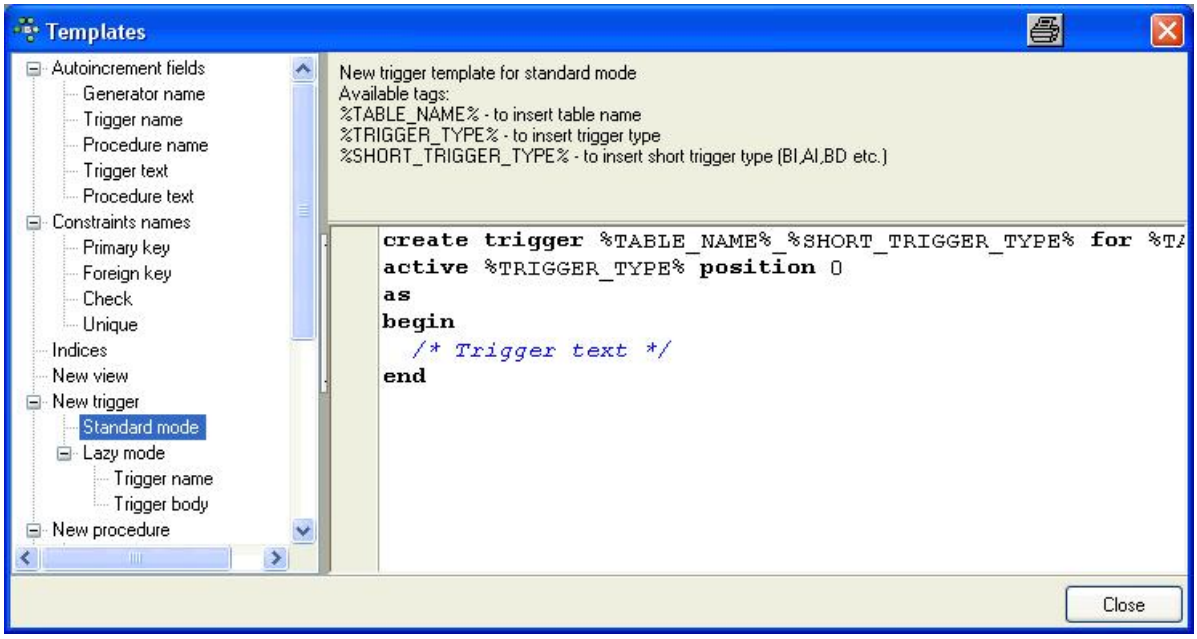
General Templates

General Templates can be found in the IBExpert Options menu. This can be used to standardize and automate the naming conventions of new database objects, and in some cases, to edit SQL code templates for creating some of these objects.



Below are a couple of illustrations of such templates.





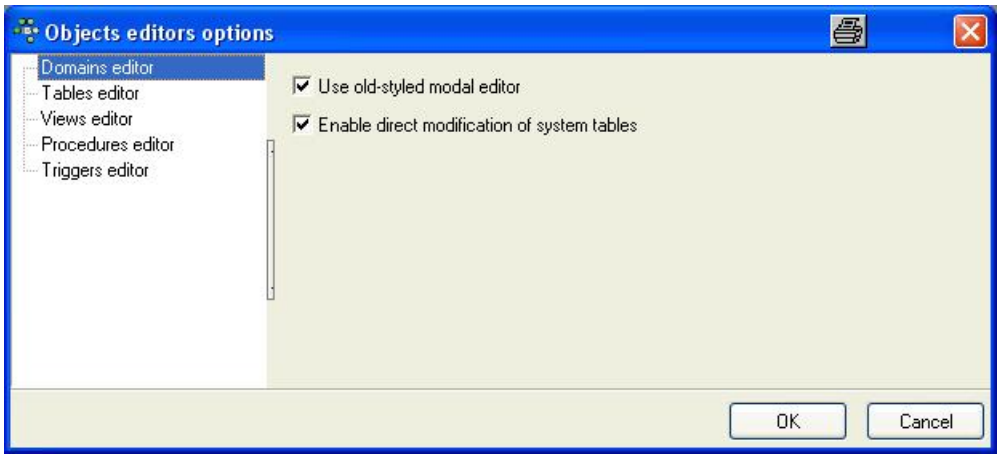
Templates for data logging triggers were added in version 2004.6.17. Please refer to Log Manager for further information.

Object Editor Options

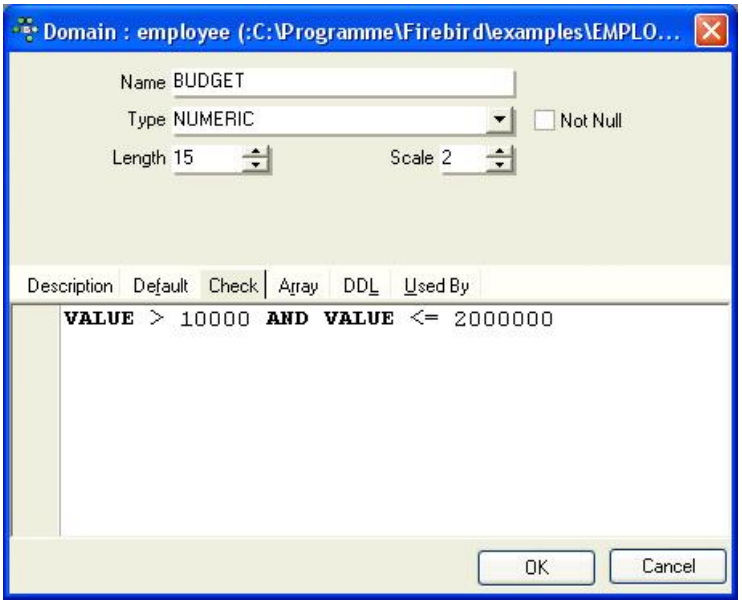
Object Editor Options can be found in the IBExpert Options menu. It opens an Objects Editors *Options* dialog, which enables users to customize certain database object editors. It is possible, for example, to specify which page should be active, when the Table Editor or View Editor is opened, or specify the standard editor mode in the Procedure Editor or Trigger Editor, and more.

Domains Editor Options

The Domains Editor Options page offers the following two options:



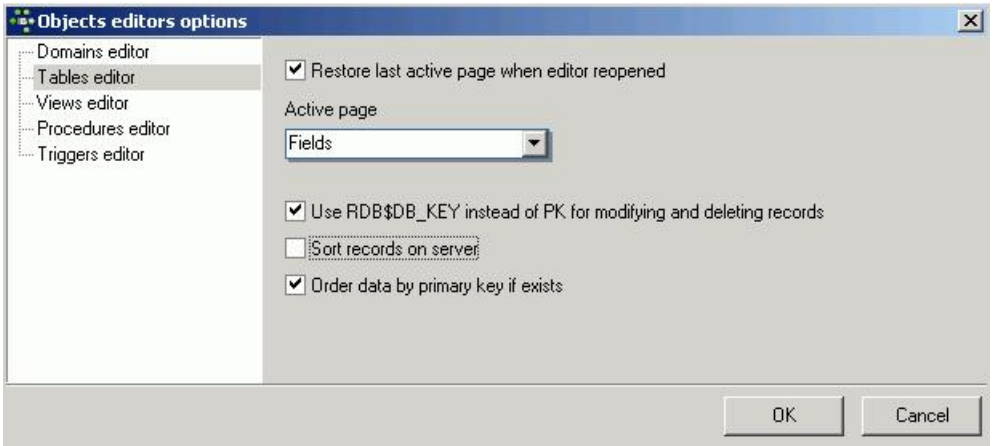
Use old-styled modal editor - when checked, this replaces the current Domain Editor with the old-style editor from earlier versions of IBExpert:



Enable direct modification of system tables - for reasons of security, it is wise not to check this item, unless the SYSDBA, administrator or database owner really need to make changes to any of the system tables.

Tables Editor Options

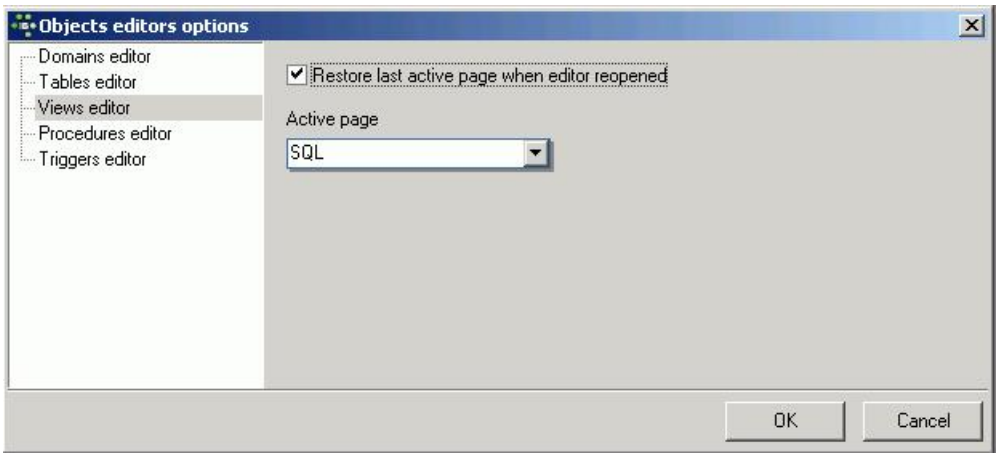
The Tables Editor Options page offers the following options:



- **Restore last active page when editor reopened:** checking this options results in the last active page remaining the active page, when the editor is reopened.
- **Active page:** offers a choice of all available pages in the Table Editor, i.e. *Fields*, *Constraints*, *Indices*, *Dependencies*, *Triggers*, *Data*, *Description*, *DDL*, *Grants*. This option does not function if the *Restore last active page when editor reopened* option is checked.
- **Use RDB$DB_KEY instead of PK for modifying and deleting records:** `RDB$DB_KEY` is an internal system field. Every single data set in the database has one of these system keys (a binary column is inserted by InterBase/Firebird for this purpose into each table). It is always unique, and can - in certain cases be very useful. For example, if a developer has created tables in his database, with no primary key, and a particular table column contains the name `Miller` twice, it is only possible, using SQL, to delete either both data sets or none. `RDB$DB_KEY` is a possibility to clearly identify individual data sets, and prevent multiple data records accidentally being deleted.
- **Sort records on server:** records may be sorted in the client memory, by simply clicking on a table column header, without running a new `SELECT`. If the data is to be sorted on the server, a new `SELECT` statement is required. This is often necessary with large data quantities as the client memory is insufficient.
- **Order data by primary key if exists:** a further sorting option for data.

Views Editor Options

The Views Editor Options page offers the following options:

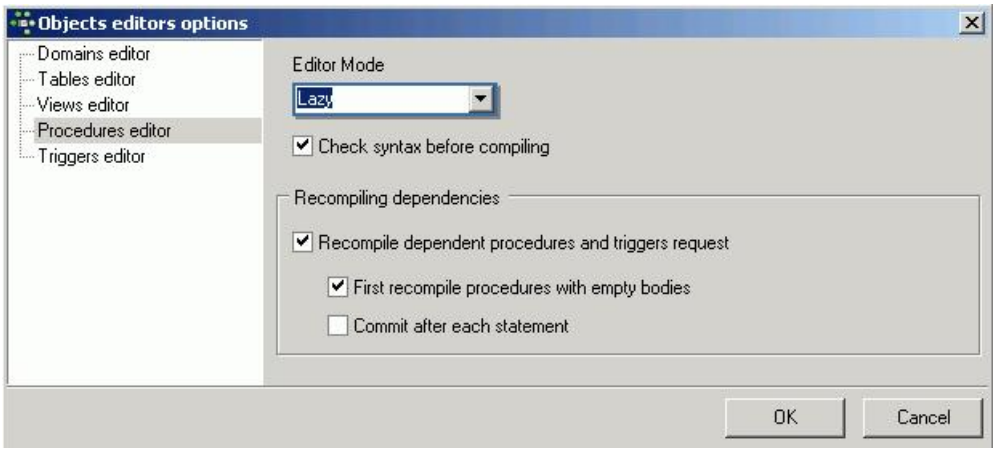


- **Restore last active page when editor reopened:** checking this options results in the last active page remaining the active page, when the editor is reopened.

- **Active page:** offers a choice of all available pages in the View Editor, i.e. *SQL*, *Fields*, *Dependencies*, *riggers*, *Data*, *Description*, *Grants*, *DDL*, *Version History*, *Recreate Script*, *Plan Analyzer*. This option does not function if the *Restore active page when editor reopened* option is checked.

Procedures Editor Options

The Procedures Editor Options page offers the following options:



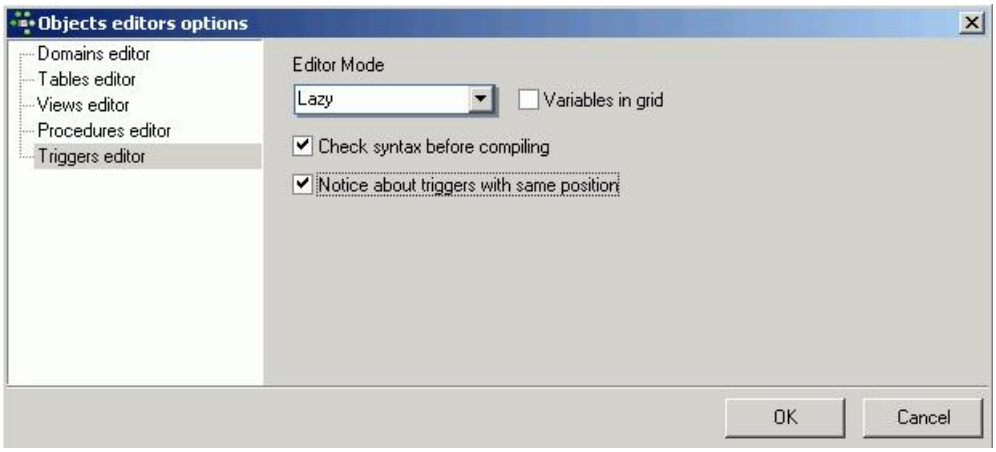
- **Editor Mode:** a default editor mode can be specified here; either Lazy Mode or *Standard*.
- **Check Syntax before compiling:** here the syntax is first checked locally for any errors, before sending the SQL to the server. This is quicker than sending everything to the server, which will then need to stop and return any eventual errors.

A number of *Recompiling Dependencies* are also offered:

- **Recompile dependent procedures and triggers request:** this option provides a reminder, asking whether procedures depending upon the amended procedure, should also be recompiled.
- **First recompile procedures with empty bodies:** this option compiles the procedure body source code after the procedure has been compiled, in order to avoid invalid references within the procedures. As soon as one stored procedure has been made dependent on another, procedures are automatically compiled in this way.
- **Commit after each statement:** allows procedures to be compiled step by step, in order to determine where exactly an error lies.

Triggers Editor Options

The Triggers Editor Options page offers the following options:



- **Editor Mode:** a default editor mode can be specified here; either Lazy Mode or *Standard*.
- **Variables in grid:** when working in lazy mode, all variables are displayed in a table.
- **Check Syntax before compiling:** here the syntax is first checked locally for any errors, before sending the SQL to the server. This is quicker than sending everything to the server, which will then need to stop and return any eventual errors.
- **Notice about triggers with same position:** if two triggers are both specified the same position, InterBase/Firebird allows this. However InterBase/Firebird chooses which trigger comes first purely by chance. This is therefore a useful warning, just in case two triggers have accidentally been given the same position number.

IBExpert Tools menu

The IBExpert Tools menu offers an extensive range of tools to aid database administration, maintenance and manipulation.

SQL Editor

The SQL Editor is an IBExpert tool which simplifies the input of SQL commands. It is used to create and execute SQL queries and view and analyze the results.

It is an essential part of IBExpert. As a rule, all work on a database is performed using SQL. The SQL Editor allows you to execute DML and DDL statements, analyze query plans and query performance, move data between databases, export query results into many formats, create views and stored procedures from SELECT etc.

The SQL Editor is intended for the execution of single commands. The Script Executive should be used for more complex scripts.

If you are new to Firebird/InterBase SQL, please refer to *Firebird Development using IBExpert* for a comprehensive introduction to SQL. The *SQL Language Reference* and the *Firebird 2 SQL Reference Guide* provide references to all Firebird/InterBase SQL keywords, syntax and parameters.

The SQL Editor can be started by selecting the IBExpert Tools menu item, SQL Editor, clicking the respective icon in the Tools toolbar, or using [F12]. This cleans the active SQL window for new input. An additional SQL Editor can be opened using Tools / New SQL Editor or [Shift + F12].

When creating stored procedures or triggers using the DB Explorer menu item New Procedure or New Trigger, an SQL Editor window is also opened. As these editors offer certain additional features (such as lazy mode, debugger), please refer to stored procedure or trigger for specific details.

The SQL Editor can be used together with the DB Explorer to quickly insert datbase object names (e.g. table fields can be marked and moved from the DB Explorer or the SQL Assistant into the SQL Editor using drag 'n' drop).

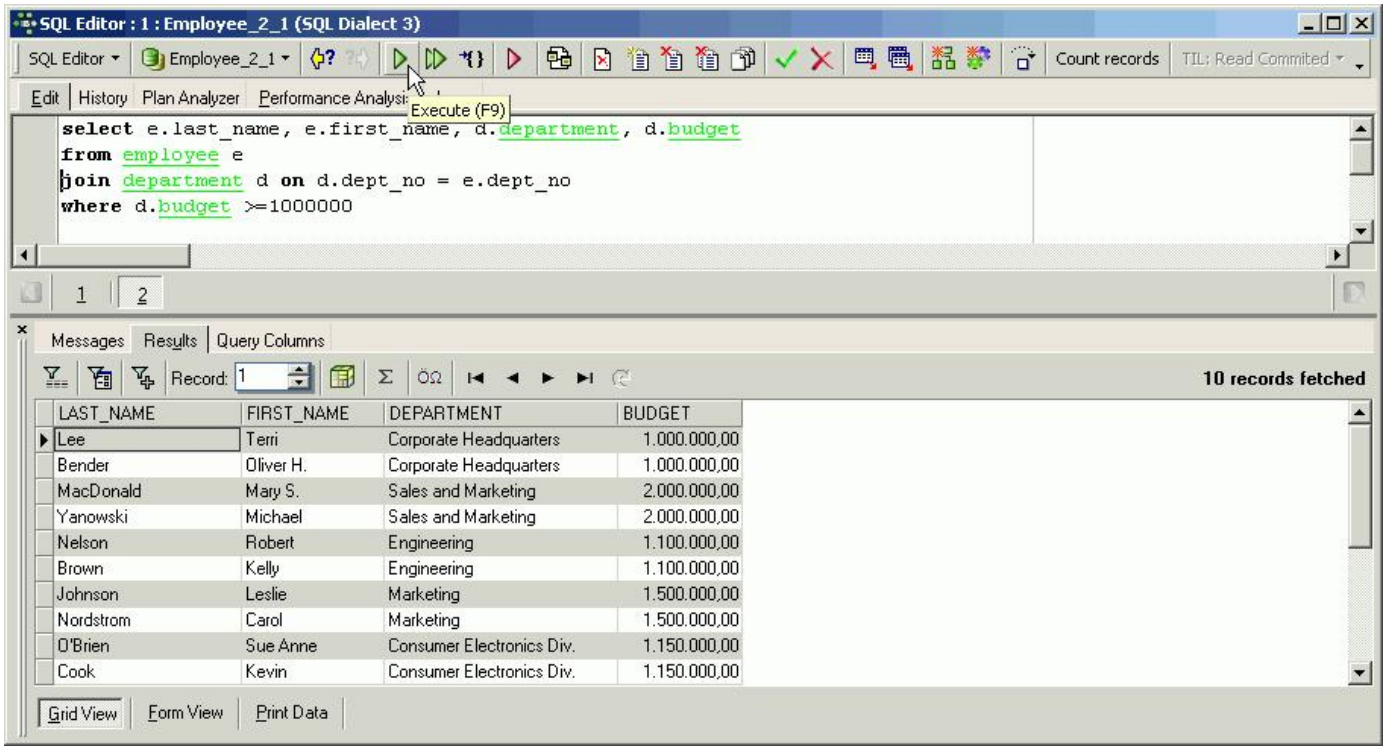
More than seven tables should not be incorporated into an SQL, as this is too time-consuming for InterBase/Firebird to analyze the indices in order to determine the most efficient solution. The database server therefore simply starts randomly, which leads to slow and lengthy queries. Since Firebird 1.5 the Optimizer has been considerably improved when working with multiple tables.

A stored procedure or view can be created from the current query directly in the SQL Editor, using the respective icons in the SQL Editor toolbar (see *Create viewor procedure from SELECT* below). And since IBExpert version 2005.12.04 there is the added possibility to turn query parameters into the input parameters of a stored procedure. Ten SQLs can be incorporated into a stored procedure.

The Tools / SQL Editor menu item includes the following:

1. Edit window (and Results)
2. Statements History
3. Plan Analyzer
4. Performance Analysis
5. Logs

The Edit window is the main input window for all SQL transactions. The History page lists previous queries. The Plan Analyzer provides information in a tree structure with statistics. A statistical summary can also be viewed in the lower panel on the *Messages* page. The Performance Analysis shows how much effort was required by InterBase/Firebird to perform this query.



For those not yet competent in SQL, the Visual Query Builder is there to make life easier! It is ideal for the beginner, although somewhat limited for more advanced work; more complex queries would need to be performed in the SQL Editor or perhaps even the Script Executive.

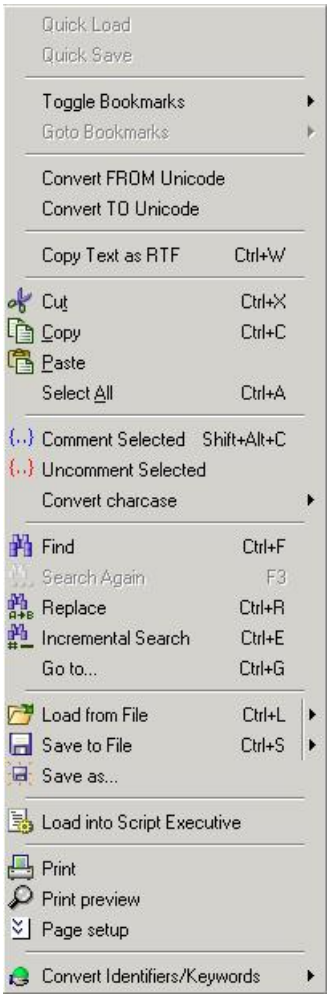
To access the Visual Query Builder simply click the

icon in the SQL Editor toolbar, or use the key combination [Ctrl + Alt + B].

To customize the SQL Editor, please refer to the IBExpert Options menu item, Editor Options and Environment Options / SQL Editor.

SQL Editor Menu

In addition to the icons in the SQL Editor toolbar, the SQL Editor has its own menu, opened using the right mouse button:

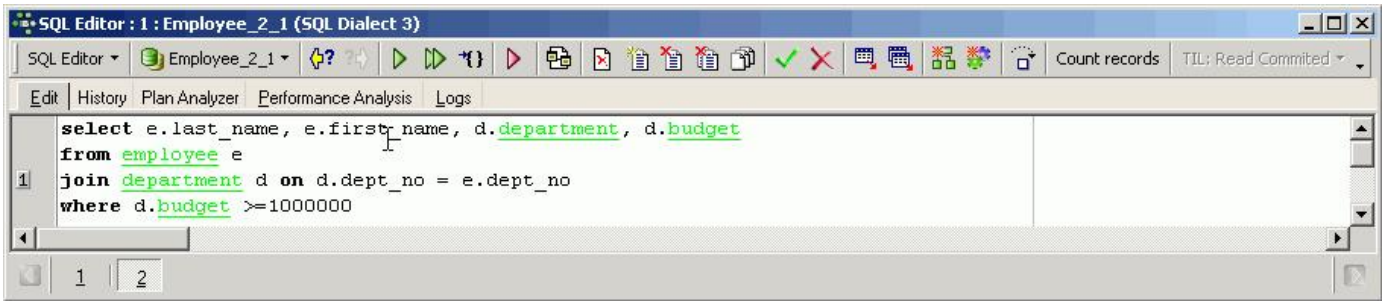


The most important menu items are detailed in this section or can be found in the IBExpert Edit menu.

Bookmark

Bookmarks are useful for flagging sections of long SQL scripts. They are purely an aid for the user and have no influence upon the SQL script or database whatsoever.

Bookmarks can be set in the SQL Editor and in the Code Editor in the Stored Procedure and Trigger Editors, using the mouse right-click menu item *Toggle Bookmarks*. They can alternatively be specified using the key combination [Ctrl + Shift + 0-9].

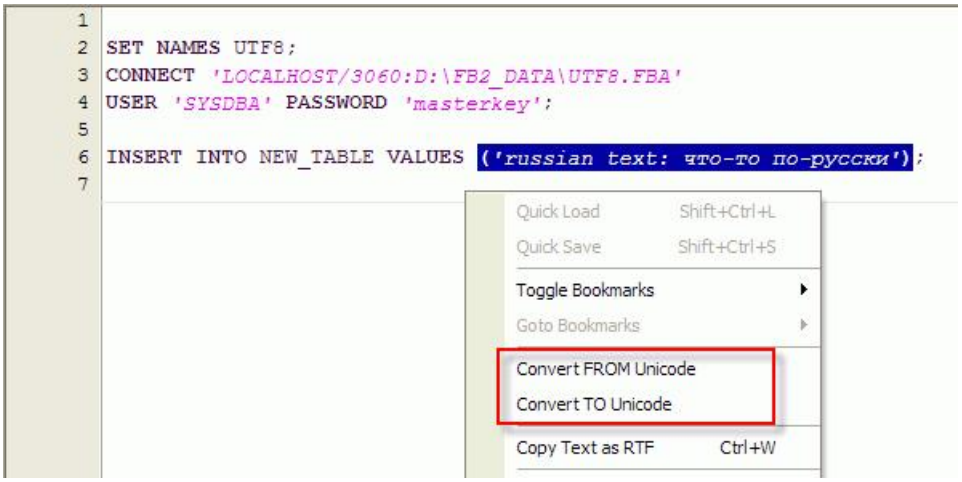


The bookmarks themselves can be seen in the left margin of the SQL Edit window. They can be numbered as wished. The mouse right-click menu item *Go To Bookmarks* can be used to spring from bookmark to bookmark. Alternatively the key combination [Ctrl + 0-9] can be used.

Bookmarks can be removed by simply unchecking those bookmarks listed in the *Toggle Bookmarks* menu.

Convert FROM Unicode / Convert TO Unicode

To convert strings from/to unicode use the corresponding items of popup menu of code editor:



Copy Text as RTF

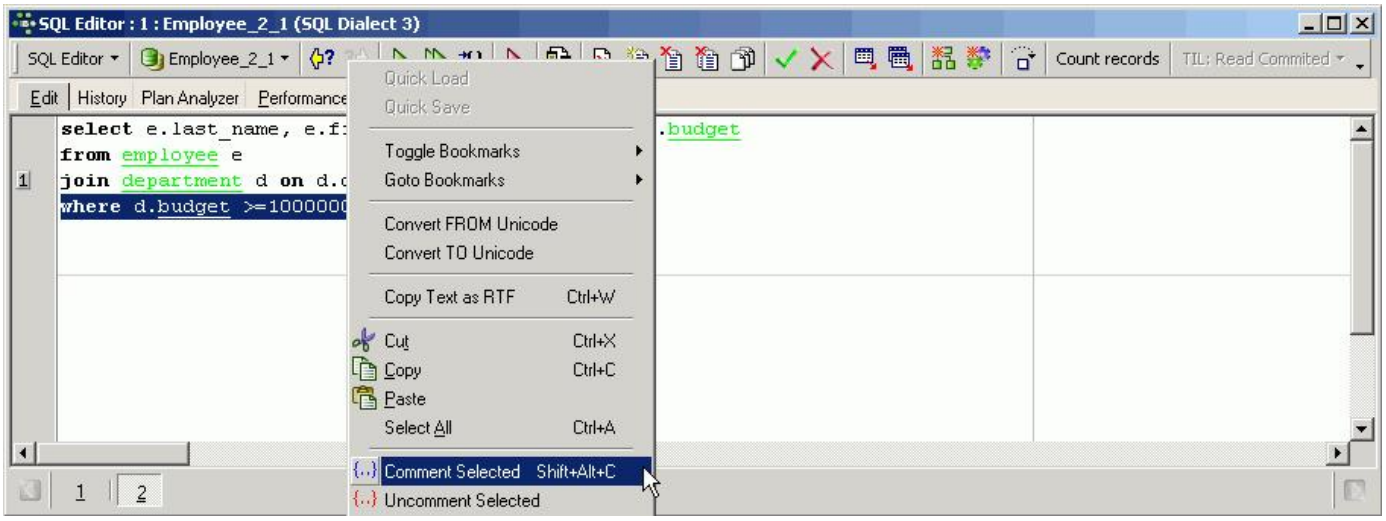
In order to copy a script, including the text formats (color, bold, indent etc.), select the script or script parts to be copied, right-click and select the menu item *Copy Text as RTF* (or [Ctrl + W]).

This feature is ideal, for example, for documentation purposes.

Comment Selected/Uncomment Selected

In certain situations it may be necessary to disable certain commands or parts thereof. This can be easily done without it being necessary to delete these commands. Simply select the rows concerned, right-click and select the menu item *Comment Selected* (or using [Ctrl + Alt + .]). This alters command rows to comments.

The commented text can be reinstated as SQL text by using the right mouse key menu item *Uncomment Select* (or using [Ctrl+ Alt + ,]).

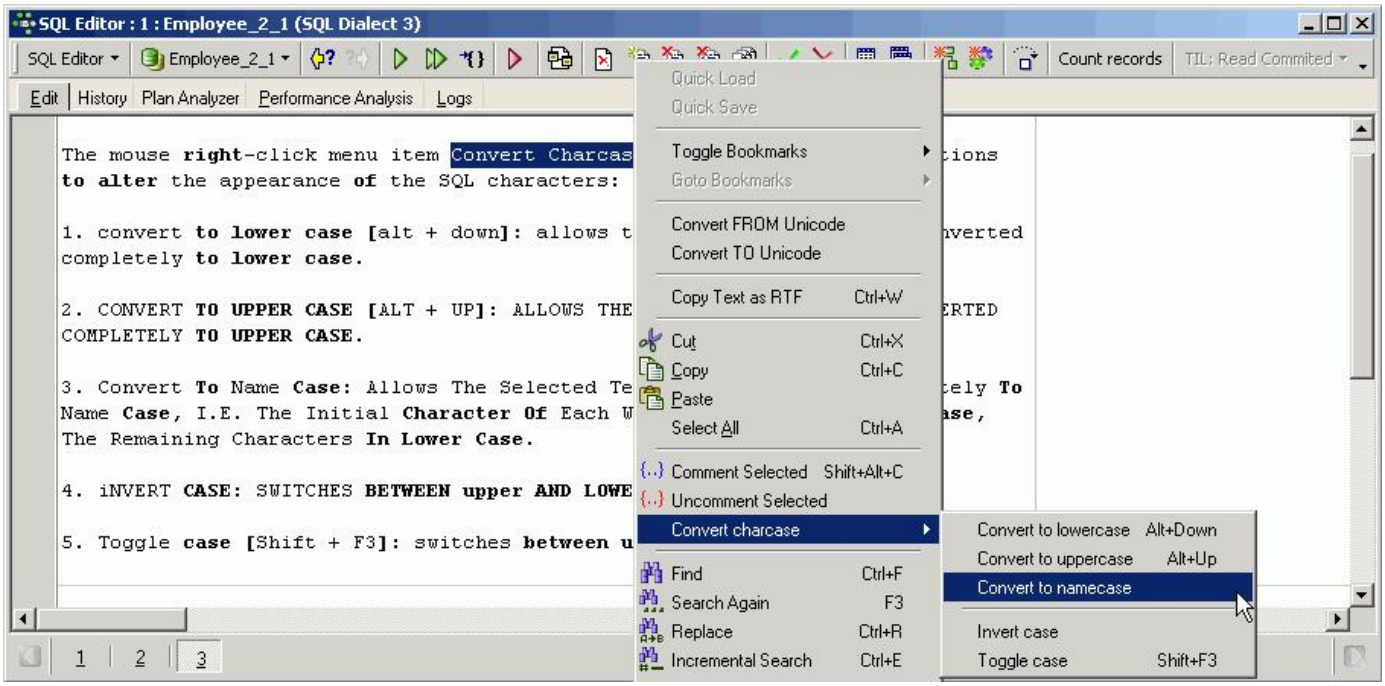


This is particularly useful when attempting to discover error sources or performing parts of standard SELECTS.

Convert Charcase

The mouse right-click menu item *Convert Charcase* offers the following options to alter the appearance of the SQL characters:

1. **Convert to lower case [Alt + Down]:** allows the selected text to be converted completely to lower case.
2. **Convert to upper case [Alt + Up]:** allows the selected text to be converted completely to upper case.
3. **Convert to name case:** allows the selected text to be converted completely to name case, i.e. the initial character of each word is written in upper case, the remaining characters in lower case.

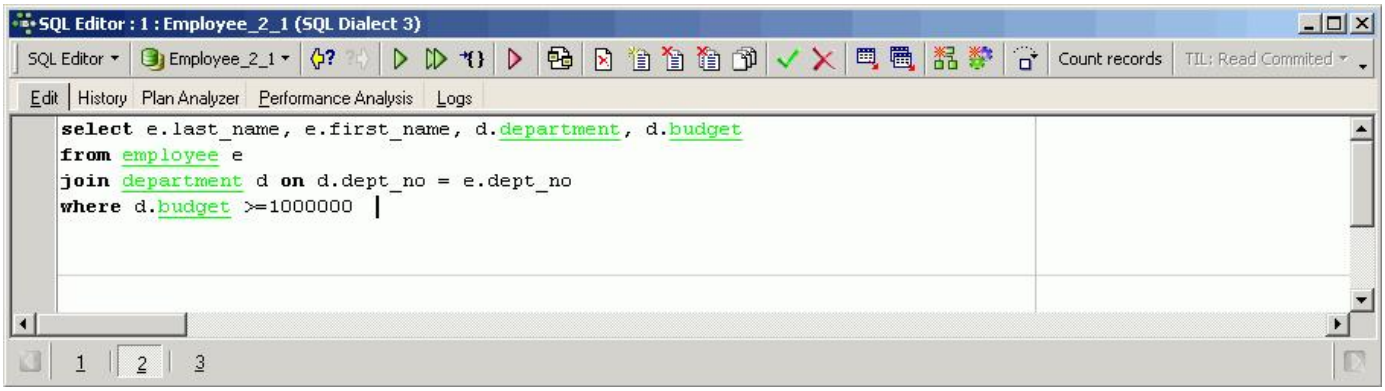


4. **Invert case:** switches between upper and lower case.

5. **Toggle case [Shift + F3]:** switches between upper, lower and name case.

Edit page

The *Edit* page appears as the active window when the SQL Editor is opened. It is the main input window for SQL commands. The SQL Editor toolbar and right mouse button menu (SQL Editor menu) offer a wide range of operations.

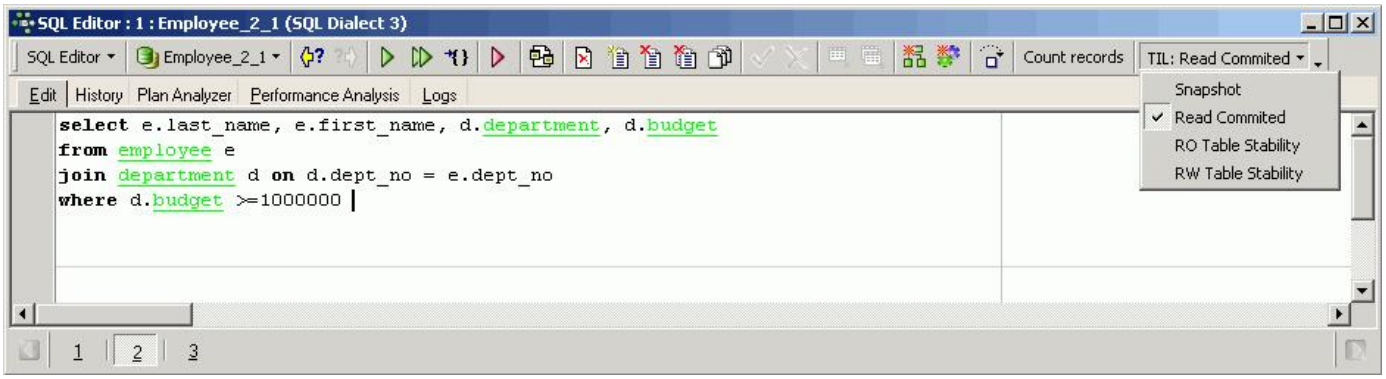


IBExpert has a number of features that really make your life easier when writing SQLs. Please refer to *Inserting text* and *Code Insight* below.

The lower status bar displays the number of open queries, allowing these to be quickly loaded in the active editing window by clicking on the respective buttons. Alternatively [Ctrl + N] can be used to load the next statement or a new window can be loaded using [Shift + F12] (IBExpert Tools menu item New SQL Editor).

The SQL Editor allows you to prepare statements and get a statement plan without having to execute your SQL by using [Ctrl+F9]. To prepare only a part of a statement just select the corresponding part of the statement and press [Ctrl+F9] or click the *Prepare* button on the SQL Editor toolbar. Since IBExpert version 2006.10.14 it is possible to view a list of query columns following preparation, on the *Query Columns* page in the lower panel.

It is so easy to execute and analyze statements (or parts of them) before finally committing. Since IBExpert version 2.5.0.61 there is the added possibility to quickly change the Transaction Isolation Level (TIL) for a separate SQL Editor. There is a corresponding button on the SQL Editor toolbar which allows you to select one of the following isolation levels: *Snapshot*, *Read Committed*, *Read-Only Table Stability* and *Read-Write Table Stability*.



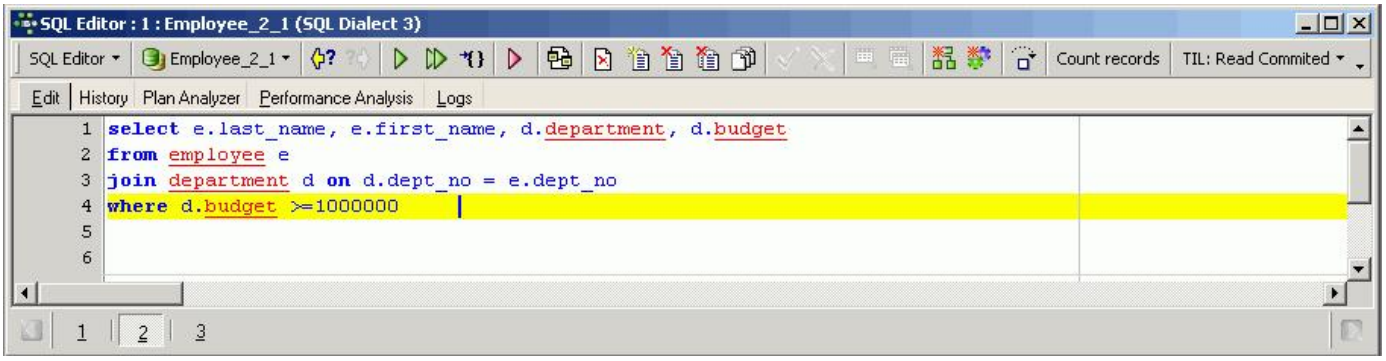
It is also possible to customize the highlighting of variables. Use the IBExpert menu item Options / Editor Options / Color to select color and font style for variables.

A Code Insight system is included to simplify command input and database objects are underlined for easy recognition.

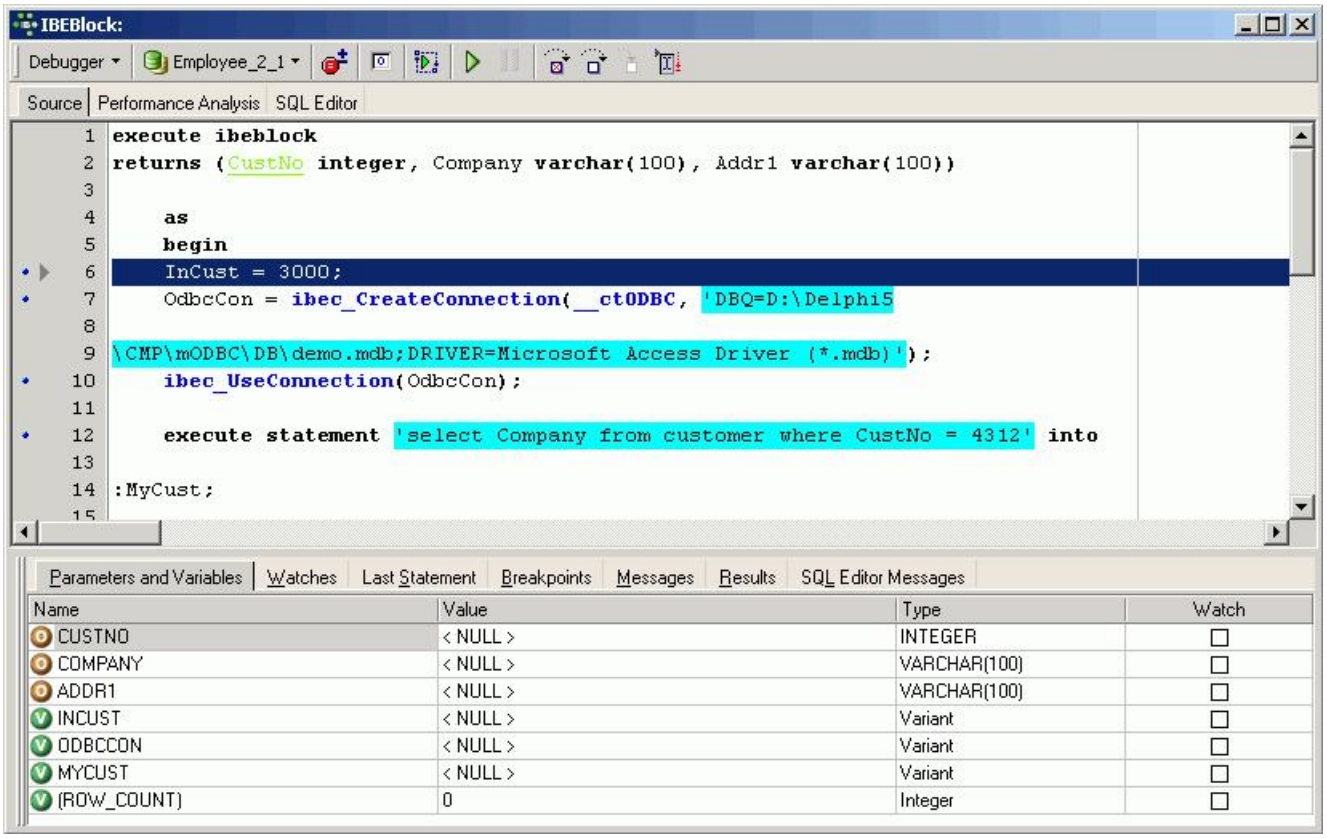
There is also a wide range of keyboard shortcuts available in the SQL Editor, e.g. [Ctrl + Alt + R] produces a list of all triggers which can be selected using the mouse or directional keys (insert using the [Tab] key); a marked block of code can be indented with [Ctrl + Shift + I] or moved back using [Ctrl + Shift + U]. Please refer to Localizing Form further keyboard shortcuts. To view the full list call the *Localizing Form* using [Ctrl + Shift + Alt + L].

Hyperlinks allow you to quickly reference database objects if necessary.

There are a number of options to customize the appearance of the code in the Text Editor. Please refer to SQL Code Editor for details, and to the IBExpert Options menu item, Editor Options, to view and specify all options available.



- Since IBExpert version 2005.02.12.1 there is added support for the `INSERTEX` command (for importing data from a comma-separated values file).
- New to IBExpert version 2005.09.25 is the support for the new Firebird 2 `INSERT ... RETURNING` statement. IBExpert shows returnable values in the messages area.
- IBExpert version 2004.04.01.1 includes added support for the `EXECUTE BLOCK` statement (Firebird 2). Since IBExpert version 2006.01.29 Firebird 2.0 blocks can also be debugged using the *Block Debugger* directly in the SQL Editor (or alternatively in the Block Editor). Simply click the *Debug* icon to open the Block Editor:



For further information regarding this Editor, please refer to *Debugger*.

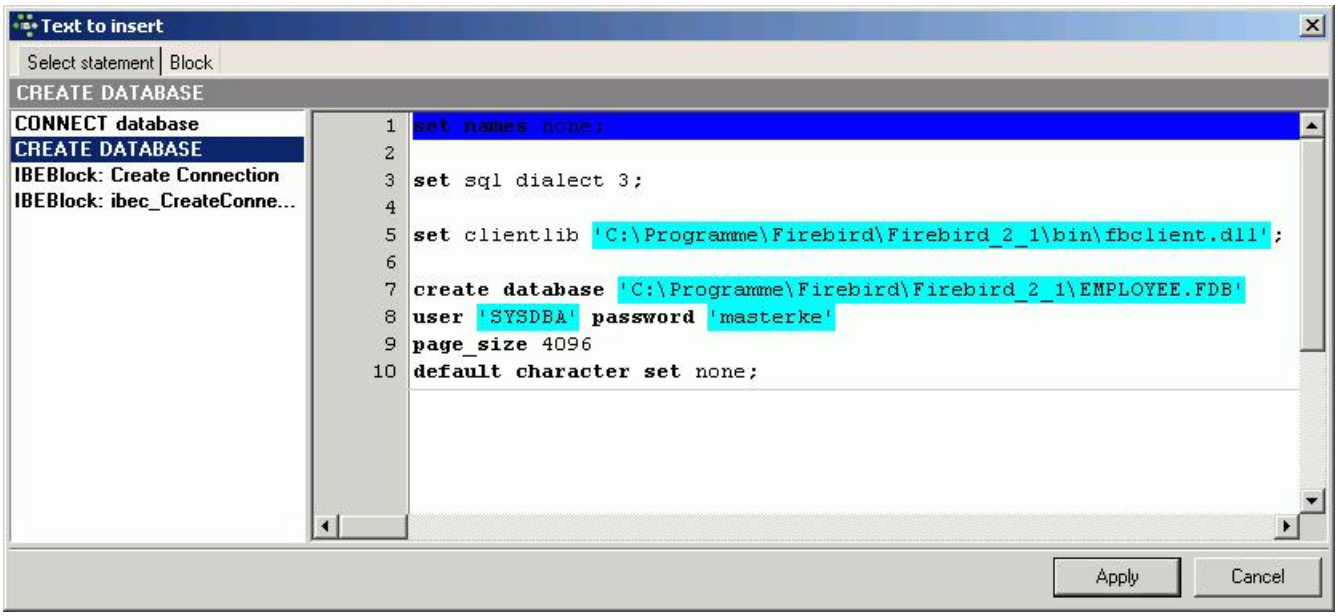
Following query execution [F9] or [Shift + F9], the returned data is displayed below the Code Editor (default setting), unless the SQL Editor has been reconfigured to display the query results on a separate Results page. (This new feature was introduced in IBExpert version 2006.01.29: using the IBExpert Options menu item Environment Options / Tools / SQL Editor, simply activate the *Separate Result page* checkbox to display query and results on separate pages).

Inserting text

Objects and fields can be simply and quickly dragged and dropped from the DB Explorer and SQL Assistant into the *Edit* page. When an object node(s) is dragged from the DB Explorer or SQL Assistant, IBExpert opens a *Text to insert* dialog.

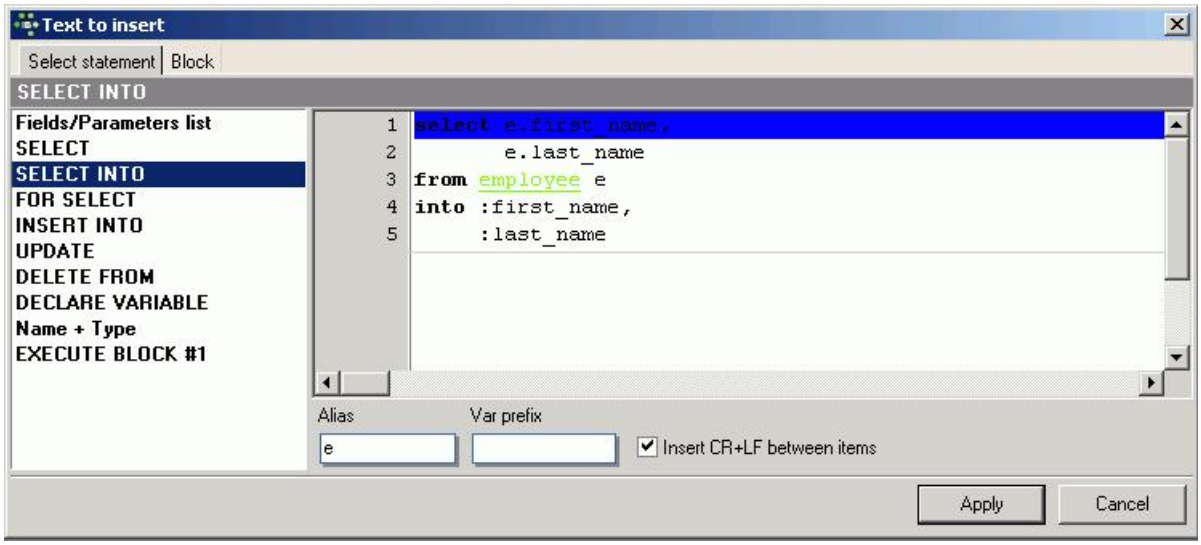
Since version 2006.08.12 IBExpert offers you the following choices:

- When dragging a database node from the DB Explorer tree into any code editor:
  - the `CONNECT database` statement
  - the `CREATE DATABASE` statement
  - IBEBlock with the `ibec_CreateConnection` function.



- By dragging a table or view name, all fields are automatically inserted in to the editor. Single or multiple fields can be dragged from a single table, by selecting with the [Ctrl] or [Shift] key depressed. Here the following statements are offered:

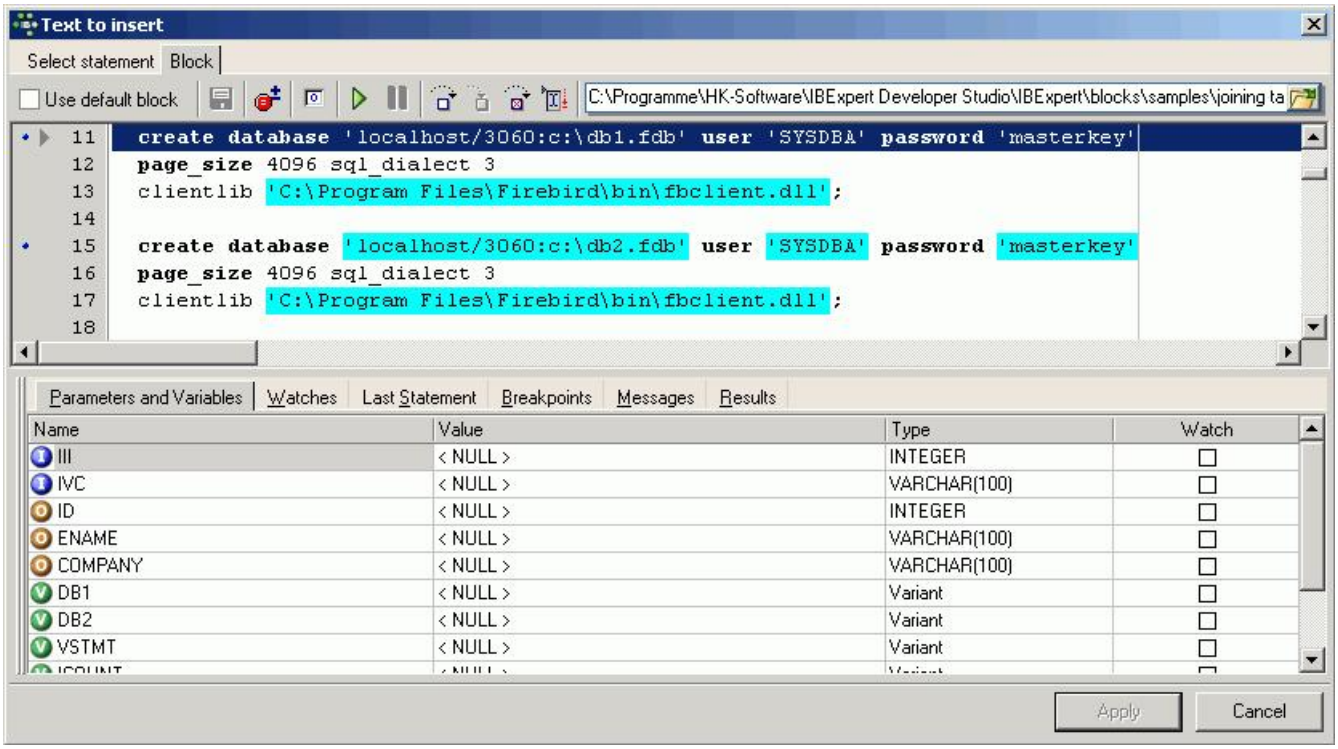
- Fields/Parameters list
- SELECT
- SELECT INTO
- FOR SELECT
- INSERT INTO
- UPDATE
- DELETE FROM
- DECLARE VARIABLE
- Name + Type
- EXECUTE BLOCK #1



Here you can also quickly create your table aliases by entering a table alias just once, it is then automatically inserted after the table name and as a prefix for all relevant fields.

And check the *CR+LF* (Carriage Return/Linefeed) box if you'd like your code to be aligned.

The *Block* page is of course for IBEBlocks:



Further information can be found in the *IBEBlock EXECUTE IBEBLOCK* chapter.

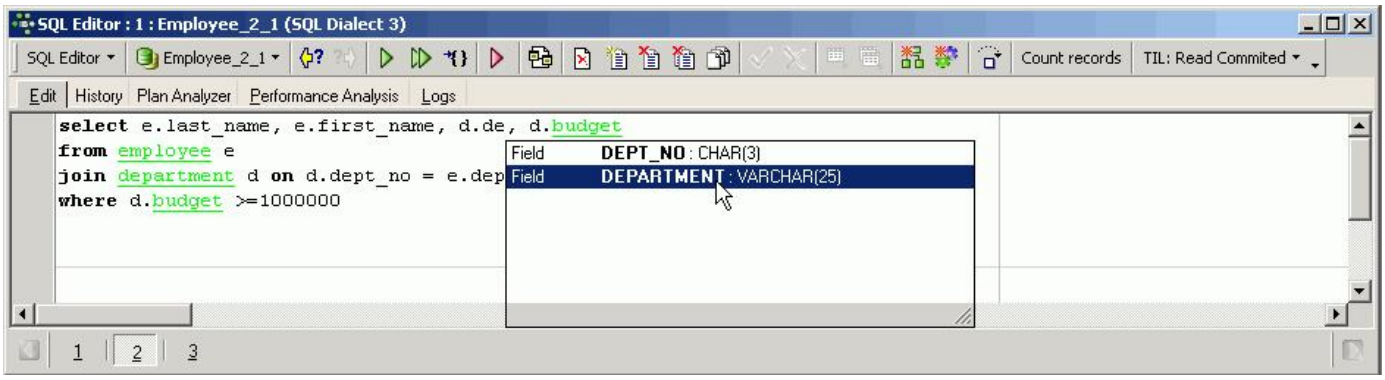
Code Insight

A Code Insight (aka *Code Completion*) system is included in the IBExpert SQL Code Editors to simplify command input. When the first word characters are typed in the SQL Text Editor, alternatives for word completion are offered in a pop-up list. Simply click the required word, or select the word using the directional keys and insert using the [Tab] key.

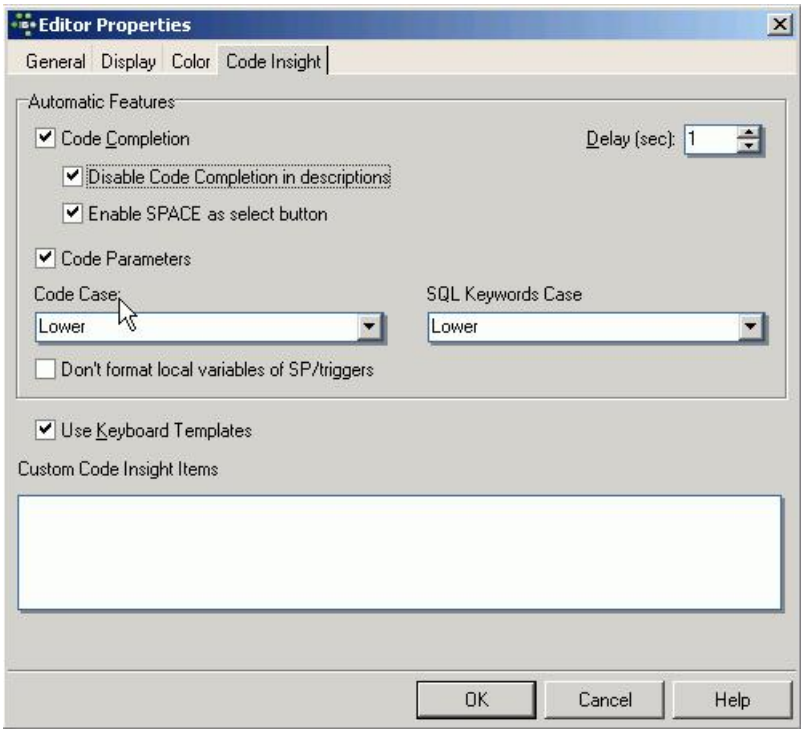
Alternatively the key combination [Ctrl + space bar] can be used to explicitly activate the *Code Insight* dialog. Database objects are underlined for easy recognition. If you wish to view a list of parameters/variables, use the key combination [Ctrl + Alt + L]. This solution has been offered as it would otherwise be necessary to parse the editor each time before the Code Insight list appears.

To call a list of certain database objects, use the logical key combinations, for example, when the key combination [Alt + Ctrl + T] is used, IBExpert offers a list of all tables beginning with the initial letter(s) already entered.

IBExpert also recognizes table aliases and automatically offers a list of all fields in the alias table, e.g. by defining the `JOB` table with an alias `J`. By holding down the [Ctrl] key you can select multiple fields, e.g. `job_code`, `job_grade` and `job_country`. By pressing the [Enter] key all fields would be automatically inserted into the SQL with the alias prefix `J`.



Using the IBExpert menu item, Options / Editor Options / Code Insight, this can be individually adapted as wished.



Further abbreviations and definitions can be specified using the IBExpert menu option, Options / Keyboard Templates.

Please note that system object information will only be offered by the Code Insight lists if these objects are visible in the Database Explorer. To list these objects in the DB Explorer, you will need to check the *Show system tables* and *Show objects' details* options found in Database Registration Info / Additional / DB Explorer.

Hyperlinks

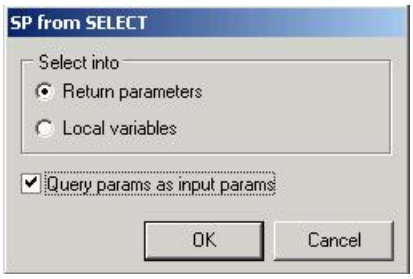
As with all IBExpert editors, the SQL Editor even offers hyperlinks. When an object name is written on the Edit page, the respective object editor can be opened by double-clicking on the hyperlink name.

To switch off the automatic hyperlink option, or to change its appearance, please refer to Options / Editor Options.

Create view or procedure from SELECT

If you wish to create a view or procedure from a valid `SELECT` statement in the SQL Editor, simply use the relevant icon to the right of the toolbar. It is possible to create a view or a procedure from an SQL statement without typing all variables and parameters.

Since IBExpert version 2005.12.04 there is the added possibility to turn query parameters into the input parameters of a stored procedure:



When creating a procedure from a select it is necessary to specify whether to select into return parameters or local variables.

IBExpert offers some other interesting features (please refer to *Special features* below).

Results

The *Results* page is automatically generated as soon as a query is executed. Since IBExpert version 2006.01.29 the *Results* page is only generated as a separate page if the default setting *Separate Result page* is activated in the IBExpert Options menu item Environment Options / Tools / SQL Editor. When deactivated, the results appear in a window below the query. When using earlier versions of IBExpert, the results are always generated on a separate page in the SQL Editor.

The Environment Options SQL Editor page can also be used to specify whether all records corresponding to the query should be extracted from the table, or just those result sets that fit onto the *Results* page view.

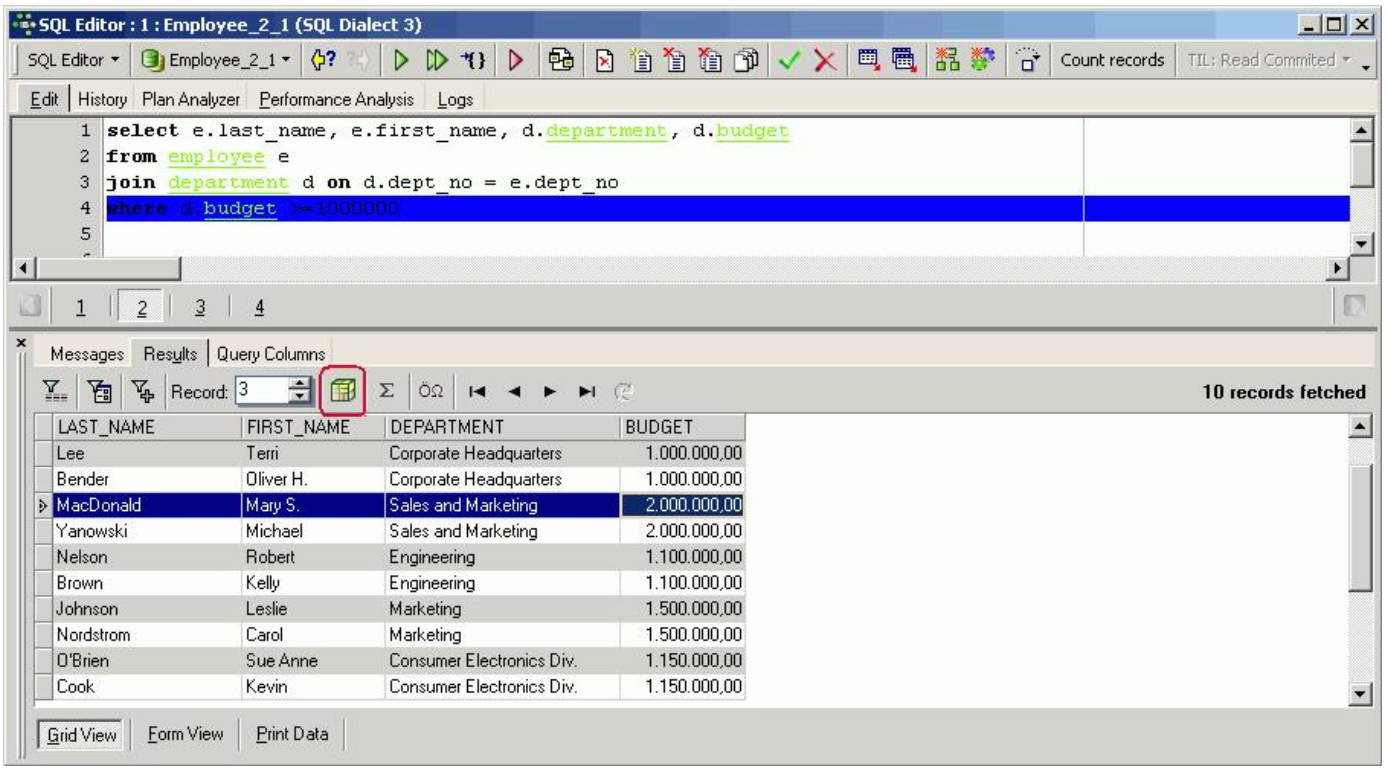
There are three modes of view:

1. Grid View

All data is displayed in a grid (or table form). By clicking on the column header the result set can be sorted (in ascending or descending order) according to that column. New data sets can also be added, altered and deleted here. And all operations, as with any operations performed anywhere in IBExpert, may be monitored by the SQL Monitor (started from the IBExpert Tools menu), particularly useful, should problems be encountered with SIUD operations.

Further information regarding the *Grid View* can be found under Table Editor / Data.

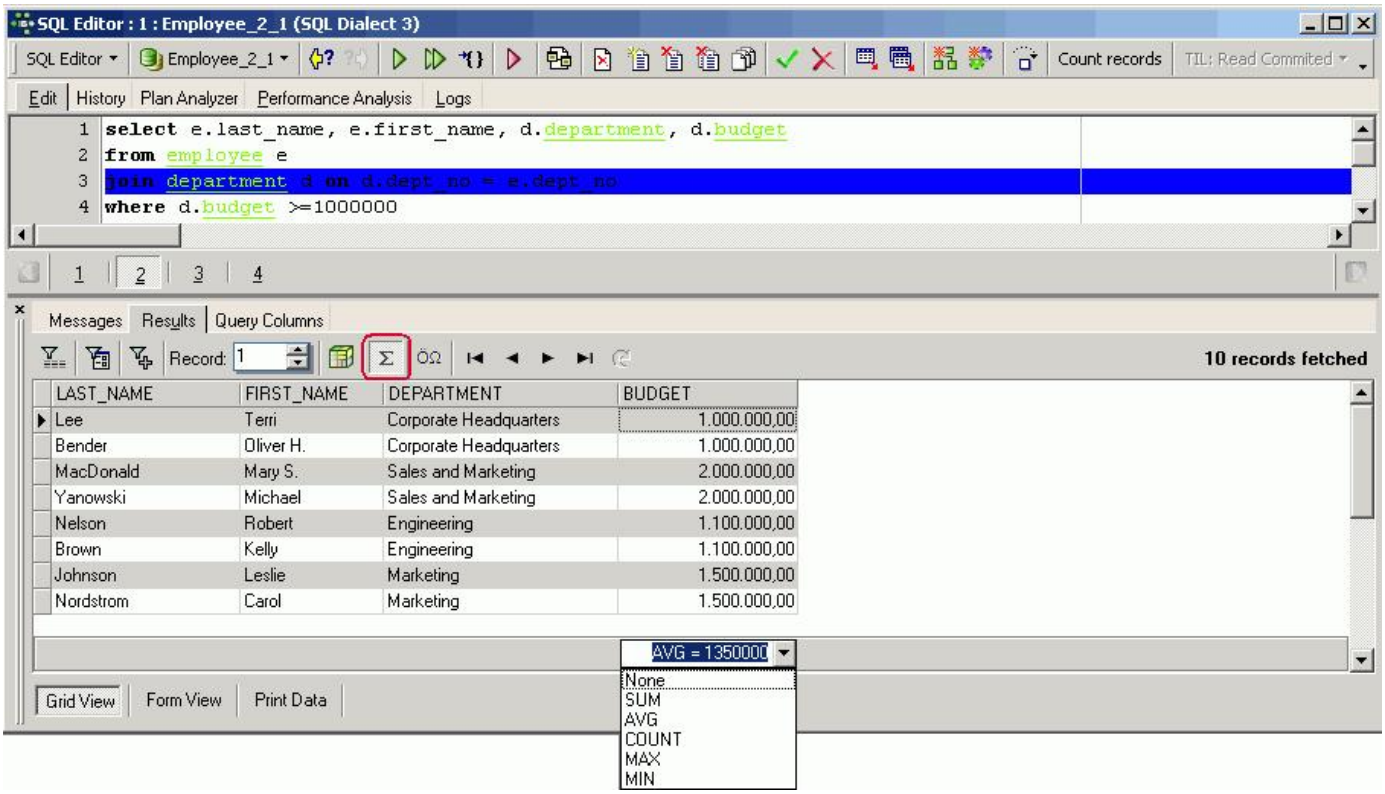
There are many options to be found in the IBExpert Options menu item, Environment Options / Grid, which allow the user to customize this grid view. Additional options are offered in the IBExpert menu items Register Database or Database Registration Info, for example, *Trim Char Fields in Grids*.



Results can only be edited in the *Grid View* if they are a live result set. Selected record(s) can be copied to clipboard as `UPDATE` statement(s). This will only work on a live query with a primary key. Since version 2004.1.22.1 mandatory (`NOT NULL`) fields are now highlighted while working with live queries. Captions of `NOT NULL` fields are displayed in bold.

A new feature introduced in IBExpert version 2004.10.30.1 is the OLAP and data warehouse tool, Data Analysis, opened using the *Data Analysis* icon (highlighted in red in the above illustration).

IBExpert version 2004.8.5.1 added the option to calculate aggregate functions (COUNT, SUM, MIN, MAX, AVG) on NUMERIC and DATETIME columns. Simply click *Showsummary footer* button on the navigation toolbar in the grid view to display the summary footer:



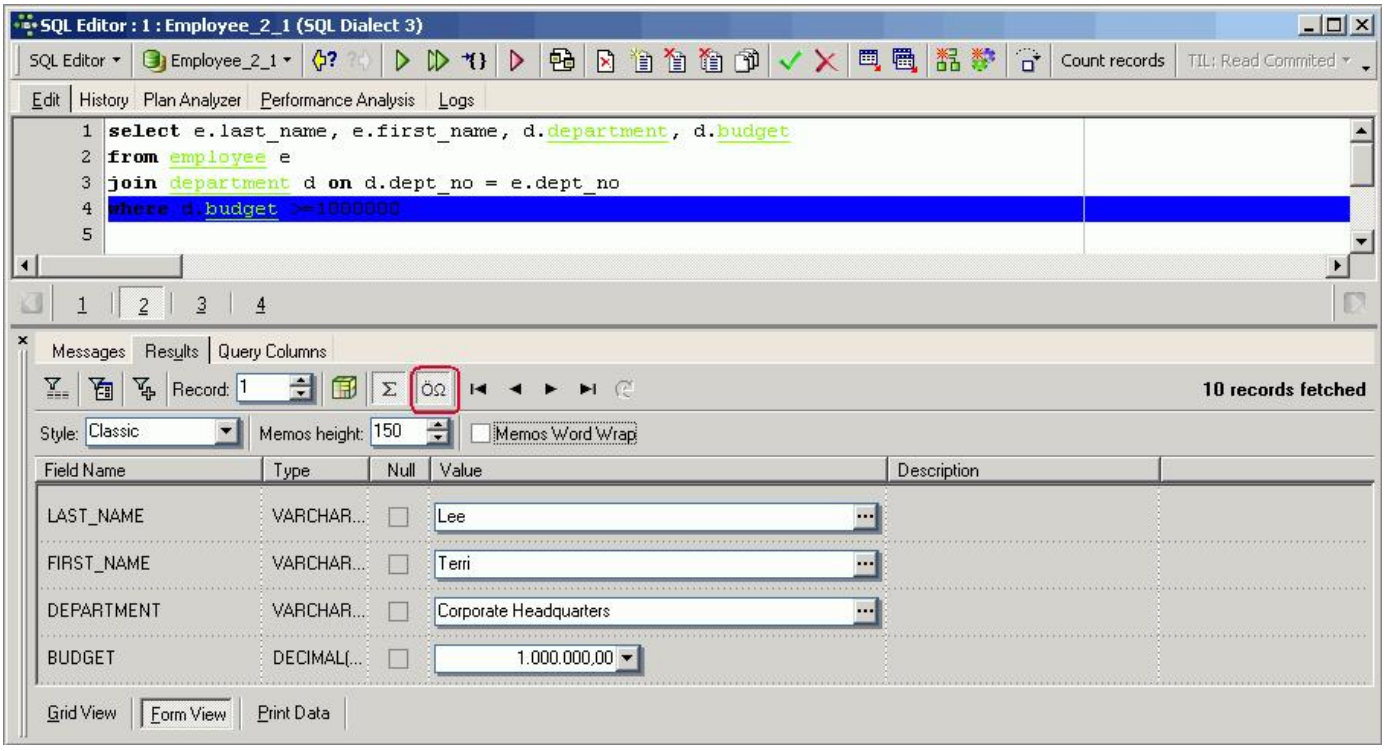
Then select the aggregate function from the pull-down list for each NUMERIC / DATETIME column as required.

IMPORTANT: these calculations are all done on the client side so do not use this feature on huge data sets with millions of records because IBExpert will fetch all records from the server to the client in order to calculate aggregates.

Since IBExpert version 2004.8.26.1 it is also possible to display data as Unicode. Simply click the relevant icon or use [F3] (see illustration below). It is not possible to edit the data directly in the grid. To edit data in unicode, use the *Form View* or modal editor connected with string cell. And IBExpert version 2007.07.18 introduced the possiblity to convert text from/to unicode. If no text is selected here, the entire content of the code editor will be converted.

2. Form View

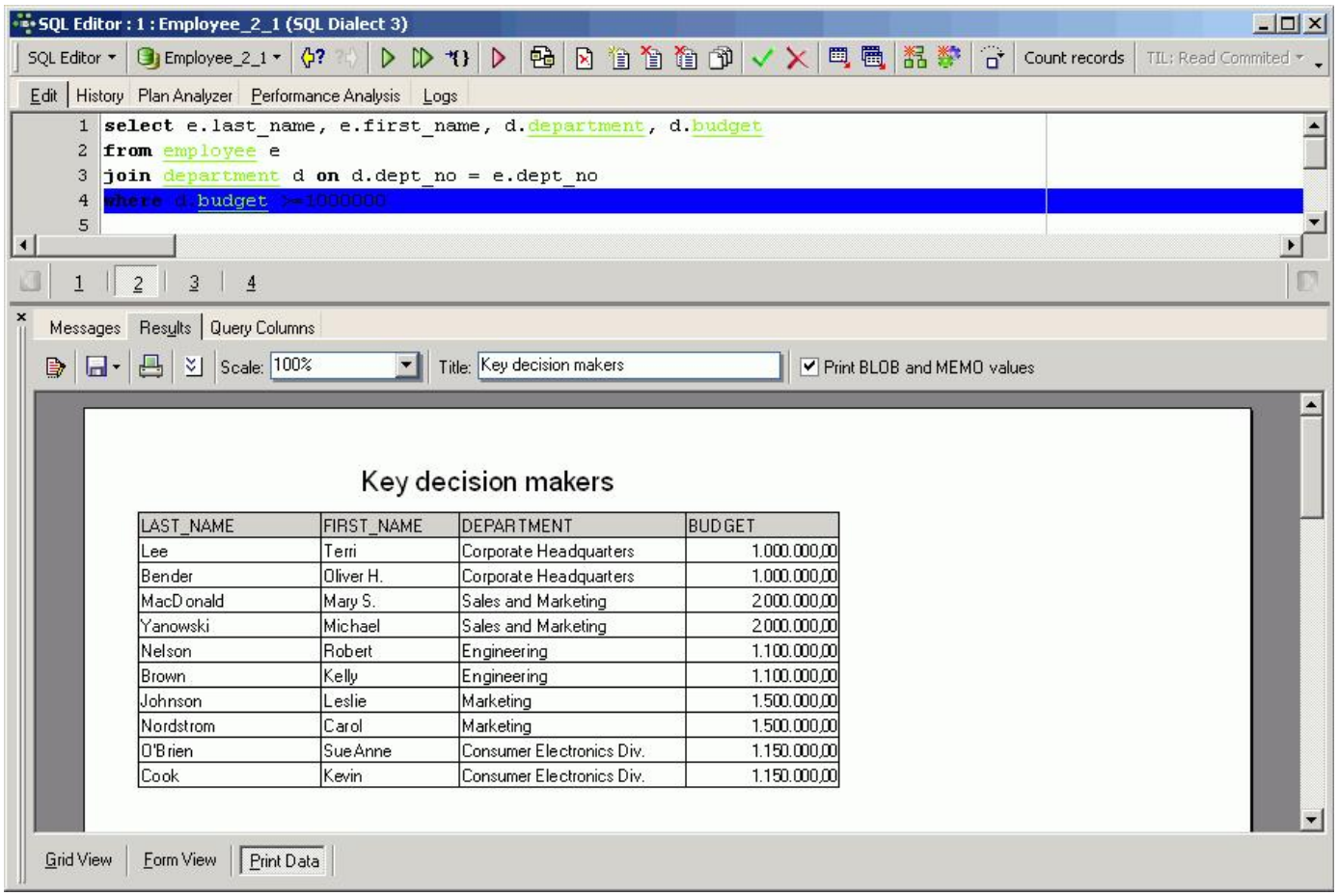
One data set is displayed at a time in a form-type display.



The *Form View* was completely redesigned in 2004. It now also displays field descriptions. It is also possible to select alternative layouts (select *Classic* or *Compact* from the drop-down list), the compact alternative for those who prefer a more compact and faster interface. Visual options now also include specification of *Memo Height* and *Memo Word Wrap*.

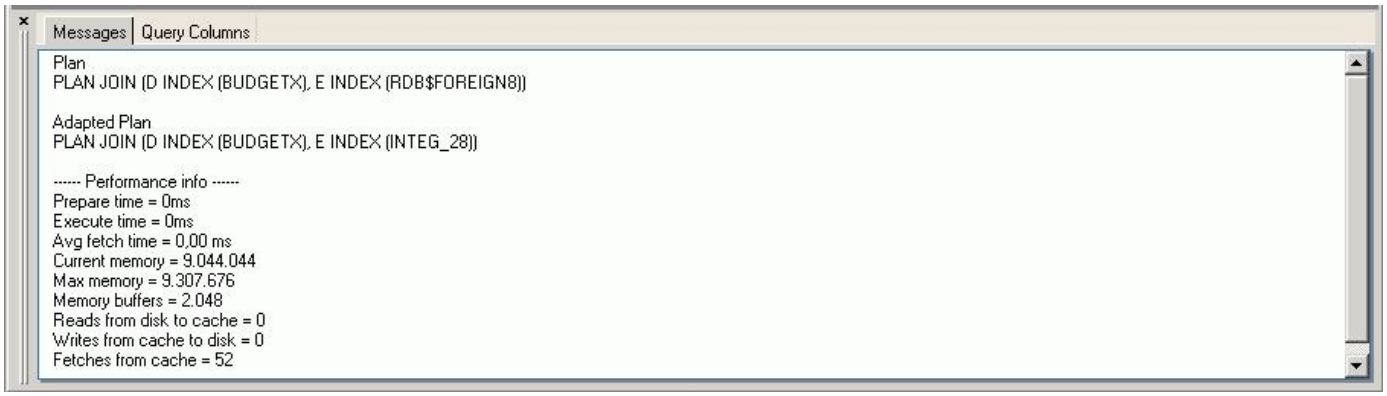
3. Print Data

Displays data in WYSIWYG mode, the data can be either edited and saved to file as a simple report or printed.

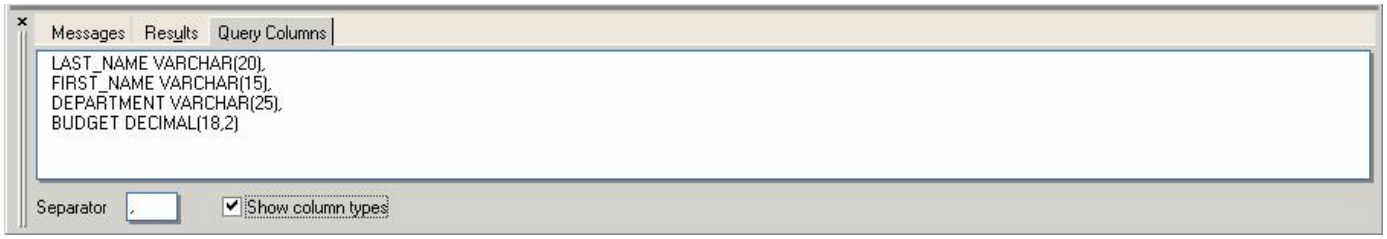


Messages and Query Columns

If you have checked the *Separate Results* page in the Environment Options / SQL Editor you will see two pages: *Messages* and *Query Columns* below the Results page:



If you have defined your results to appear below the *Edit* page, you will find *Messages* and *Query Columns* to the left and right of the results:



The Results page also has its own right-click menu, which can be used to perform numerous operations upon the resulting data (please refer to Table Editor / Data for more information).

Filter Panel

It is possible to work with filters on your results and also on data on the Table Editor's Data page (*Grid* and *Form* view), allowing the addition/deletion of criteria and filters directly in the data sets resulting from the executed SQL.

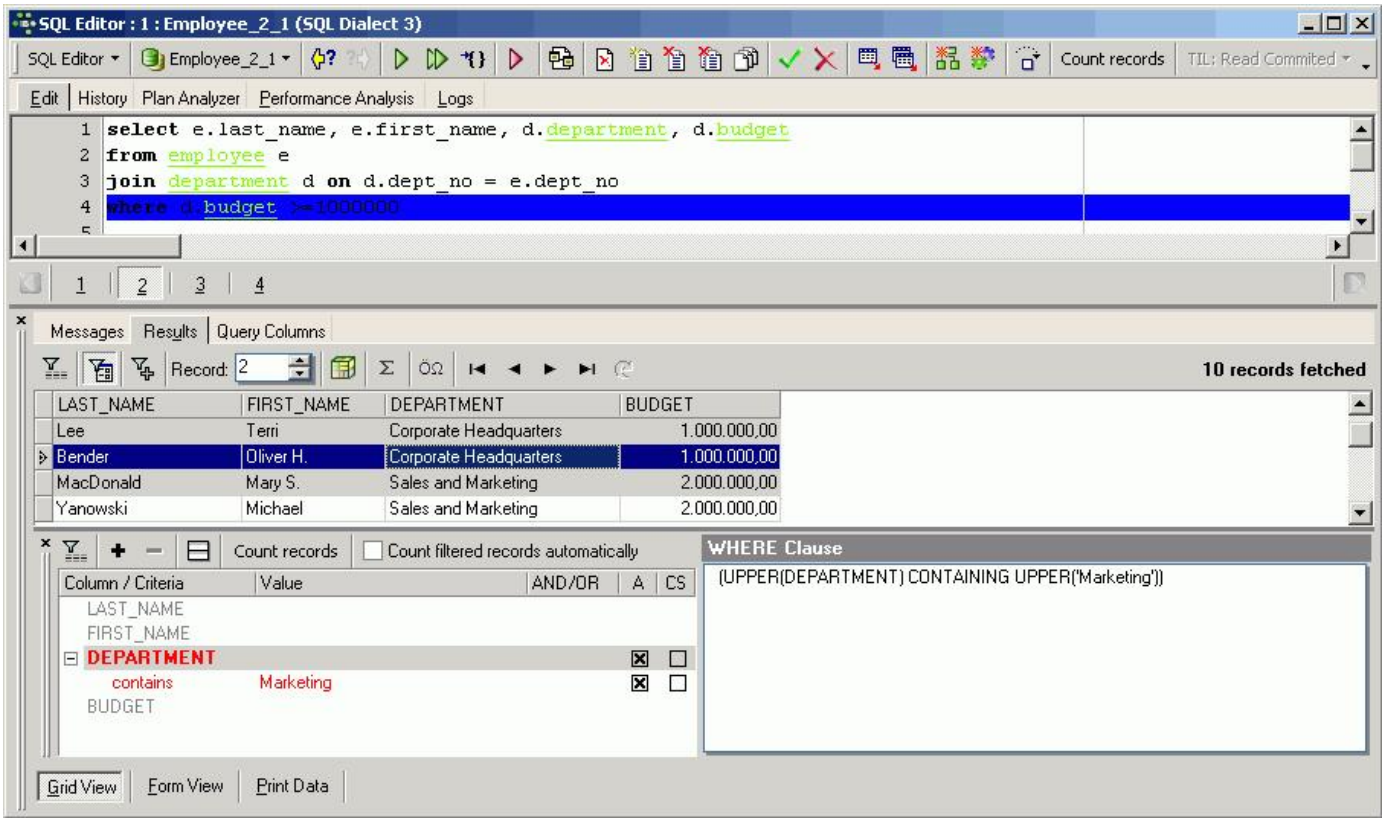
The *Filter Panel* is opened using the *ShowFilter Panel* icon:

or [Ctrl + Alt + F]. A new two-part window appears. This can be split horizontally or vertically by clicking on the *Vertical Layout* icon or using the key combination [Shift + Ctrl + L].

New filter criteria can be added by placing the cursor on the field, where a filter is to be inserted, and using the +-button or [Ins] key. To delete filters use the -button or [Ctrl + Del] key combination. Select the comparison operator from the pull-down list adjacent to the list of field names and specify the desired value(s) .

When a second field is marked and a new filter for this field is added, the AND column is automatically filled (default is AND, may be altered to OR if wished, using the space bar or mouse click). The two right-hand columns provide check box options, to specify whether a filter should be active or not (column *A*), and to specify whether case-sensitivity is of importance (*CS* column). The second panel displays the WHERE clause that has just been specified.

Since IBExpert version 2005.02.12.1 the number of filtered records is automatically recalculated when the filter condition is changed.



The filter area can be deactivated by re-clicking the *ShowFilter Panel* icon or [Ctrl + Alt + F].

Export Data

Please refer to *Export Data* for further information.

Export Data into Script

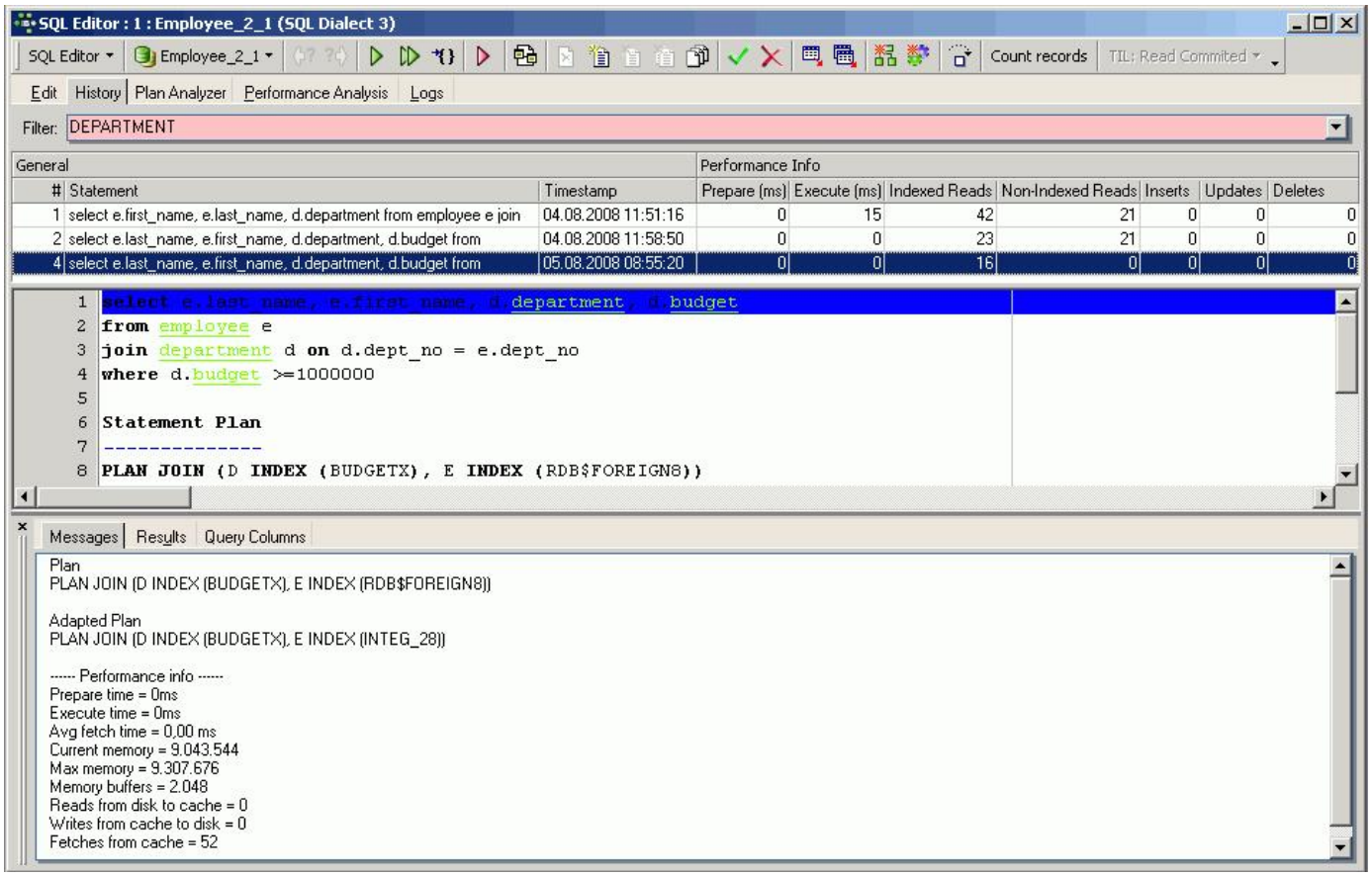
Please refer to *Export Data into Script* for further information.

Statements History

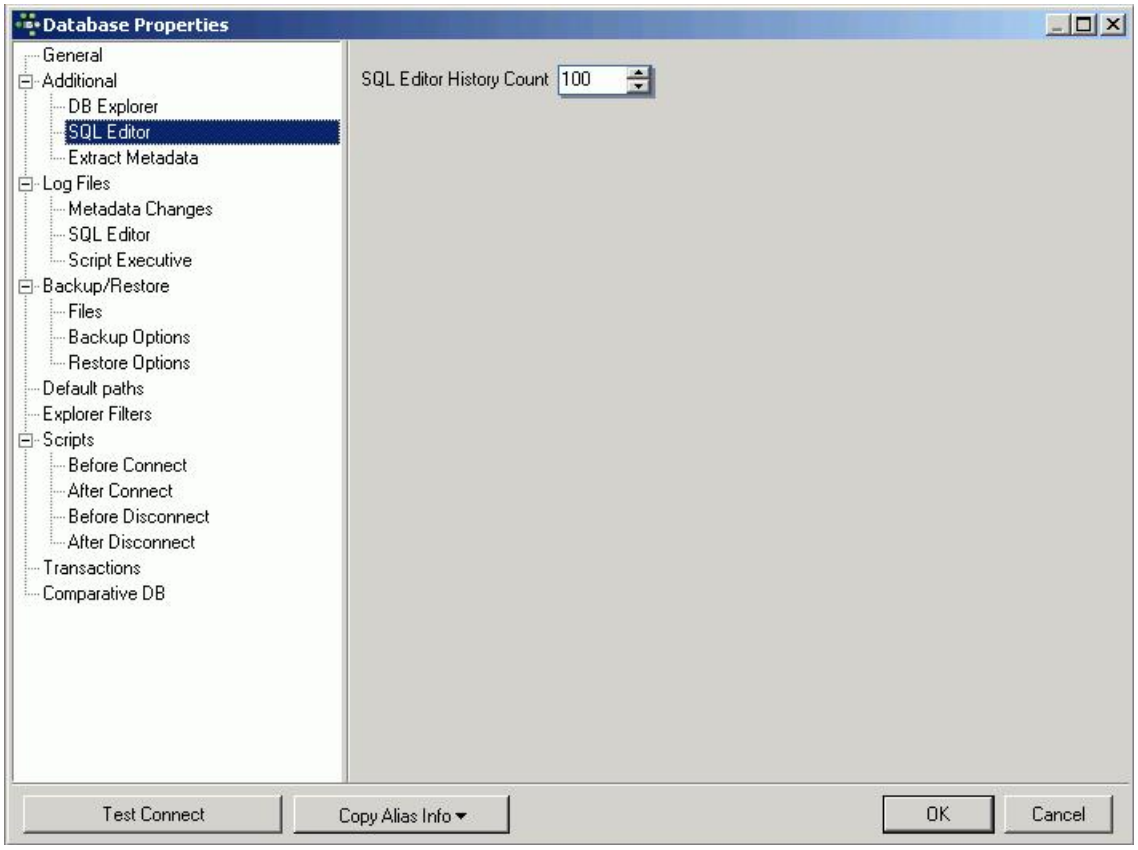
The *History* page can be found in the SQL Editor, and lists previous SQL queries that have been executed and produced a result (not necessarily committed), along with their performance statistics. This saves having to reenter recurring commands, and offers a concise overview of individual SQL performances for comparison. All statements are only visible when the same database alias is in use.

Below this list, the middle panel displays the script of a selected query.

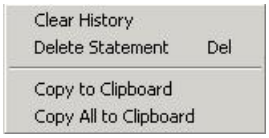
The filter (directly above the statement list) can be used to display only those objects containing the character string entered in the filter, e.g. Find all SQLs containing a SELECT or all SQLs containing DEPARTMENT.



The SQL *History* lists a record of the last 100 statements. This default quantity of 100 stored statements can be altered by using the IBExpert menu item Database or the DB Explorer right mouse button menu: Database Registration Info / Additional / SQL Editor, where the SQL Editor *History Count* can be specified as wished.



The SQL *History* list can be streamlined, as and when required, by deleting individual list entries, using the right mouse button.

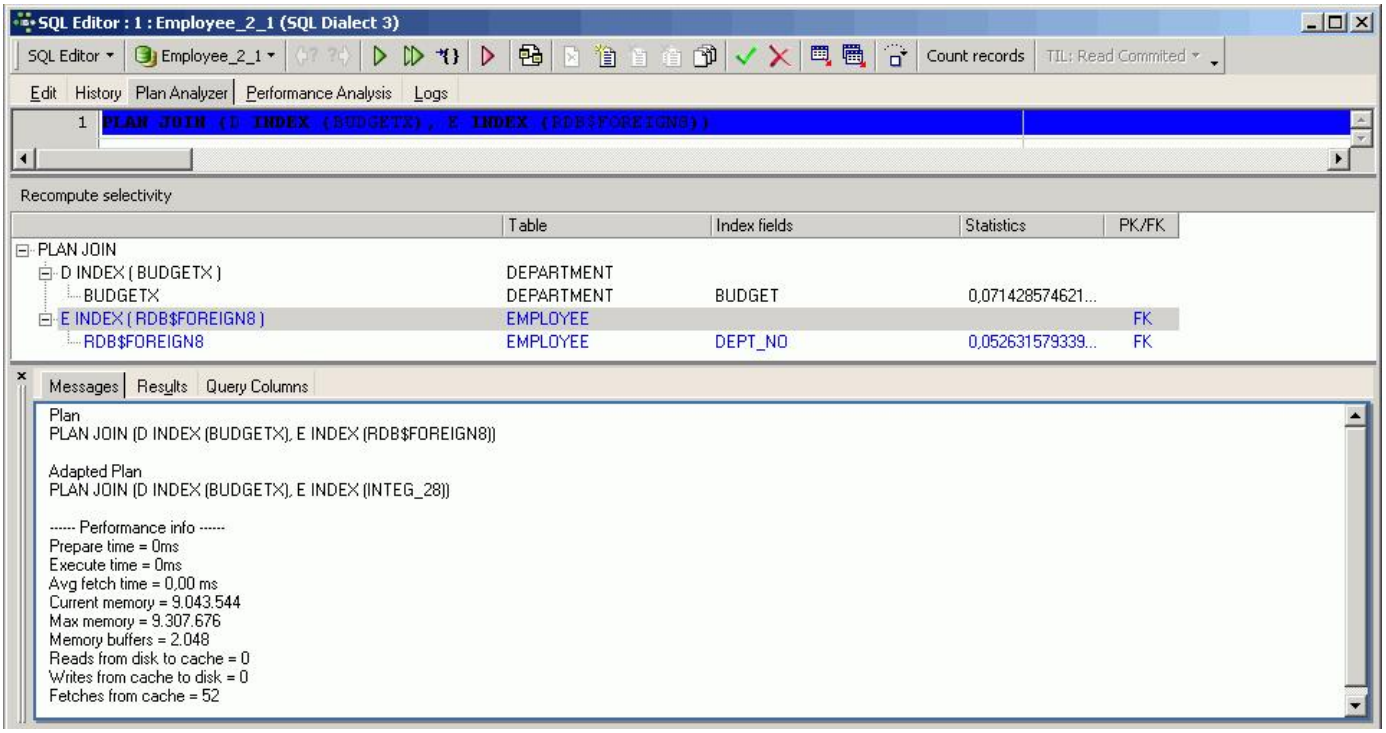


This menu also allows single statements (or all) to be copied to clipboard.

Plan Analyzer

The SQL Editor *Plan Analyzer* (also a part of the Procedure Editor and Trigger Editor) shows how Firebird/InterBase approaches a query, e.g. with SORTS, JOINS etc., which tables and indices are used. This information is displayed in a tree structure: firstly what and which data quantities, and secondly what is carried out with this data and how.

The plan is an InterBase/Firebird description, showing how the Optimizer uses tables and indices to obtain the result set. If the word SORT is displayed, you should check whether improvements upon the query or the indices are possible.



The *Plan Analyzer* provides information in the center panel in a tree structure with statistics, and a summary of the plan and performance is listed in the lower panel.

For further information regarding the use and effects of indices in queries, please refer to *Index statistics*.

Performance Analysis

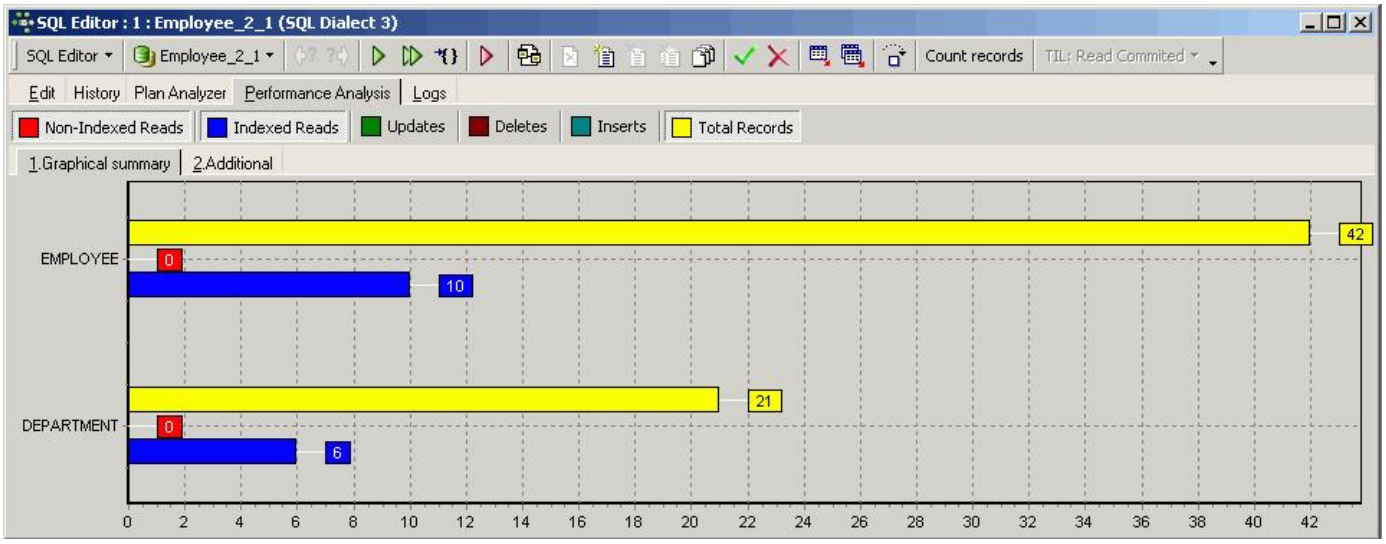
The *Performance Analysis* is part of the SQL Editor, Visual Query Builder and Stored Procedure Editor. It displays information showing how much effort was required by InterBase/Firebird to carry out an executed query or procedure. The analysis is performed after a SELECT statement is opened or a stored procedure started.

It is possible to deactivate the *Performance Analysis*, by checking the *Disable Performance Analysis* option, found under Database / Register Database or Database Registration Info / Additional. This may be desirable when working remotely with a slow modem connection.

It is however often interesting to know what exactly a procedure or query does and how; and all this can be viewed in the *Performance Analysis*. The main advantage here of course, is the possibility to compare the performance of different queries and procedures.

Graphical Summary

The *Graphical Summary* provides an overview, broken down by the tables involved, of the number of operations performed by the query/procedure, including reads (indexed and non-indexed), updates, deletes and inserts. It shows whether indices have been used indicating the efficiency of the database's indices. The figures displayed refer to the number of data sets.



The x-axis lists the names of the tables consulted by the query/procedure, with the number of operations displayed graphically. Click the performance type you wish to view, and it will be added to the graph. Click the button again, to remove it.

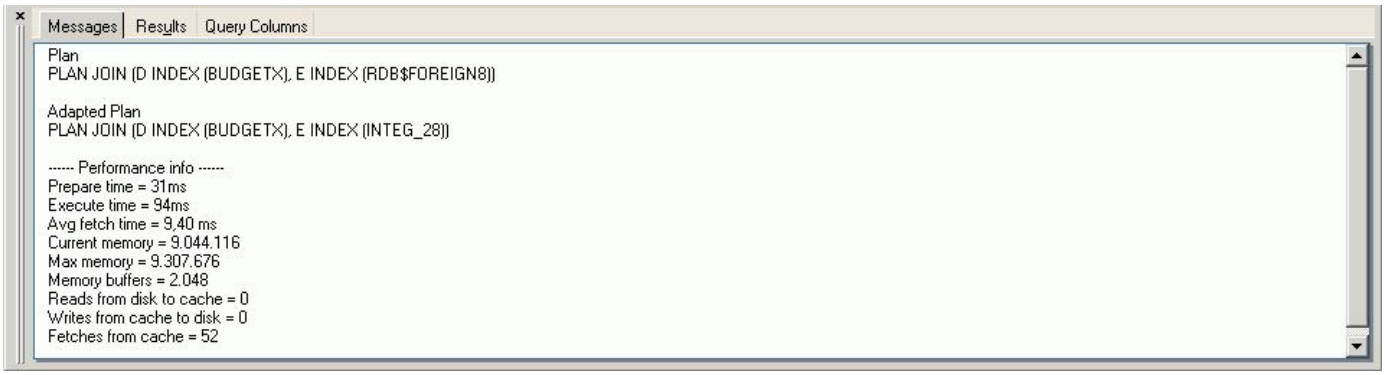
SELECT statements will only have a *Reads* result, but some stored procedures will also have results for *Updates*, *Deletes* and/or *Inserts*.

The operation types are as follows:

1. **Non-indexed reads:** A non-indexed reads indicates that the data was read without the aid of an index. In most situations this can be both time- and memory-consuming. Non-indexed reads always include a large number of data sets, as the server needs to search through the whole table(s) to find the relevant information. All data pages from these table(s) need to be loaded into the cache.

   The SQL Editor's query plan shows which tables were read without an index using the term NATURAL.

2. **Indexed reads:** An indexed read indicates that the data was selected by the InterBase/Firebird server using one or more indices (named in the query plan displayed on the *Messages* page in the lower panel).



   This results in many cases in a significantly lower number of data sets being consulted than with a non-indexed read, saving both time and memory.

   For further information regarding the use of indices, please refer to *index*. For details of improvements in Firebird 2.0, refer to the *Enhancements to indexing* chapter in the *Firebird 2.0.4 Release Notes*.

3. **Updates:** This displays the number and type of updating operations in an executed query/procedure. The figures displayed refer to the number of data sets, broken down by table.

4. **Deletes:** This displays the number and type of deleting operations in an executed query/procedure.

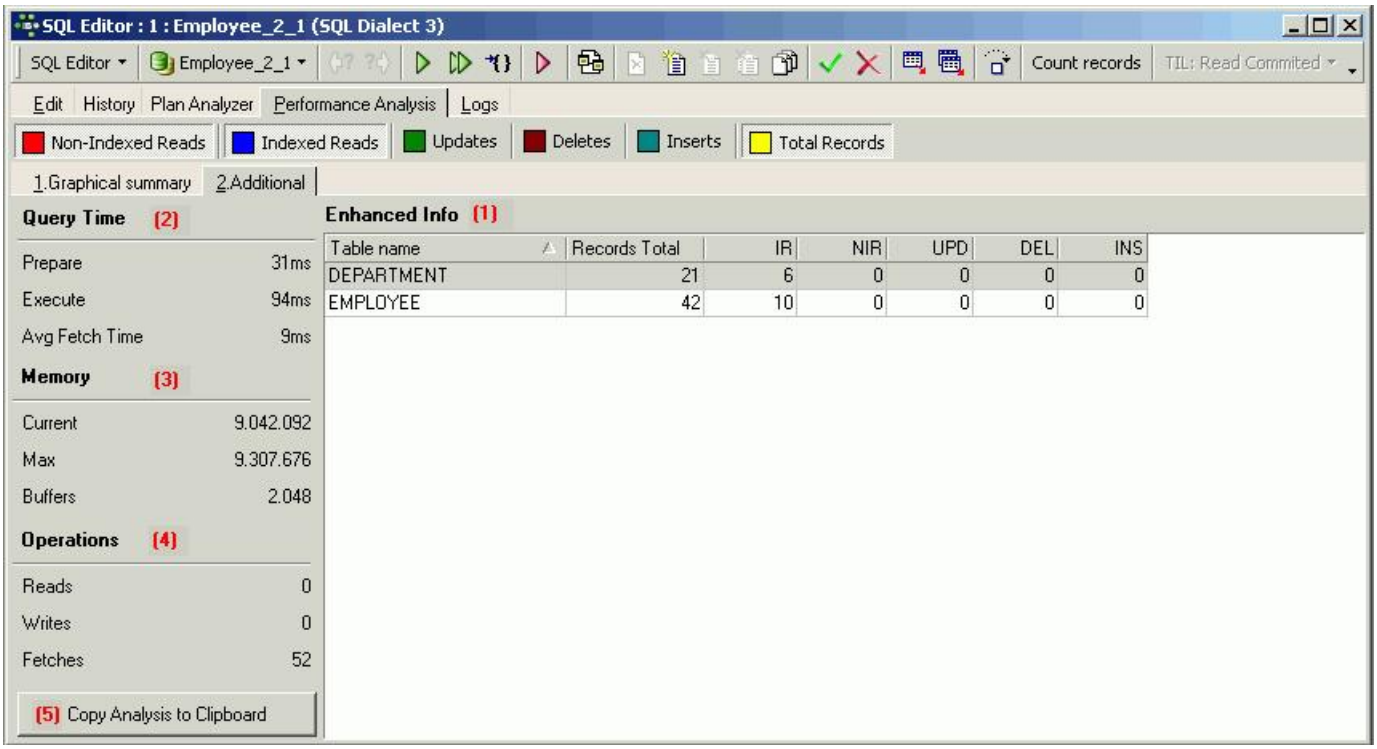
5. **Inserts:** This displays the number and type of inserting operations in an executed query/procedure.

6. **Total number of records:** This displays the total number of records consulted.

In the SQL Editor the lower panel displays the query plan, along with a summary of the performance information included on the Additional page. For further information regarding the query plan, please refer to the Plan Analyzer.

Additional

This displays a statistical report. The *Enhanced Info* displays a statistical summary of the information shown in the *Graphical Summary*. Certain additional information, such as *query time*, *memory* and *operations*, is also included in this section.



The analysis displayed on the *Additional* page can also be documented using the *Copy Analysis to Clipboard* button.

(1) Enhanced Info

The *Enhanced Info* displays a statistical summary of the information shown in the *Graphical summary*.

The names of tables consulted during execution of the query/procedure are listed in the first column, with the number of data sets listed according to the following criteria:

- IR = Indexed Read
- NIR = Non-Indexed Read
- UPD = Updates
- DEL = Deletes
- INS = Inserts

(2) Query Time

Query time shows the time needed to prepare for the execution of the query/procedure, along with the execution time and average fetch time.

Prepare: This measures the preparation time required by InterBase/Firebird to plan and prepare the query/procedure execution, i.e. from the moment when the source text is sent to the server and is compiled on the server in binary form (it decides which indices, tables etc. need to be used to perform the query/procedure).

When a query/procedure is executed a second time, the query time is usually 0 ms, as it has already been prepared.

Execute: This measures the direct execution time of the command.

Avg fetch time: This shows the average fetch time pro data set. This figure is calculated based only on those data sets that can be seen in the returns and does not include those that are not yet visible. An optimal analysis can be attained when the query/procedure is executed using [Shift + F9] = *Execute and Fetch all*.

(3) Memory

This shows the memory development during and following execution of the procedure/query.

Current: This displays the current memory used by the server.

Max.: This displays the maximum memory used by the server during execution of the query/procedure.

Buffers: This displays the number of data pages that are being held as cache on the server (from InterBase 6 onwards the standard is 2,048). This can be found in the corresponding configuration file: since Firebird 1.5 it is called `firebird.conf`; in older Firebird versions or InterBase, it is called `ibconfig`, found in the main InterBase directory.

This can be altered for the current database if wished, using the IBExpert Services menu item, Database Properties / Buffers. The total KB is calculated according to the current database page size. For an alteration to become effective, it is necessary for all users to disconnect from the database and then reconnect. Buffers are only reserved if they are really necessary for pages loaded from the database file.

(4) Operations

Operations displays the number of data pages that were read from the database file to the memory, written and fetched, while executing the query/procedure.

Reads: This displays the number of pages read for the executed query/procedure. This is necessary when data sets have to be loaded, that are not already in the memory.

Writes: This displays the number of pages written while executing the query/procedure. If the total cache buffers are too small to load subsequent pages, it may be necessary for the server to save altered pages to the hard drive, in order to make room for further pages to be loaded. If these values are very high, it may be wise to increase the buffers, providing of course that physical memory is sufficient.

Fetches: When a query/procedure is started, the command (or series of commands) is sent to the database server. To obtain results, numerous data sets/pages need to be referred to (= *fetch*), in order to perform the operation. Fetches are, in other words, internal operations performed by InterBase/Firebird in order to successfully execute a query/procedure. This indicates, for example, if deleted data sets in a `SELECT` are recognized as deleted, they will still appear here in the number of fetches, as the server also searches through those data sets that have been marked as deleted. This can however offer an advantage over the number of indexed and non-indexed reads, as these only display operations on undeleted data sets. If the query is executed again, the result is quicker if the garbage collection is running simultaneously.

Using the *Performance Analysis*, the number of fetches in data pages could possibly indicate why one query is quicker than another with an equal number of data sets and the same index plan.

(5) Copy Analysis to Clipboard

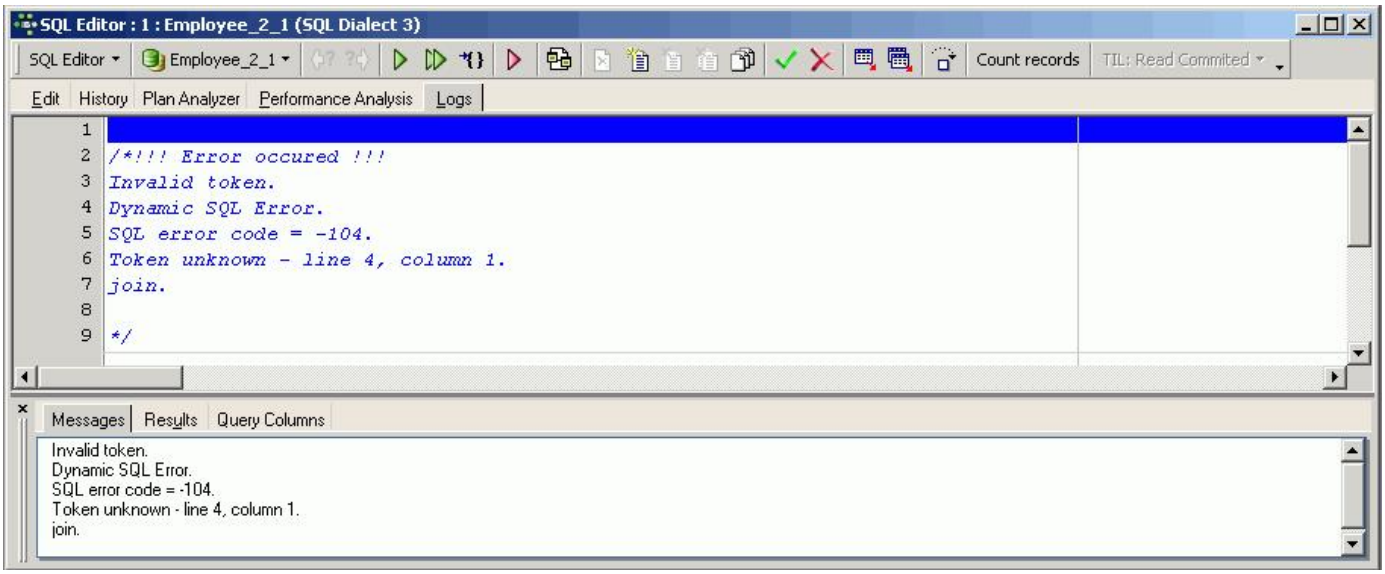
The *Copy Analysis to Clipboard* button copies all information included in the Additional page, including both the grid contents (= *Enhanced Info*) and the statistics listed in the left-hand panel (= *query time*, *memory* and *operations*).

The *Copy Analysis to Clipboard* button can be found in the bottom left corner of the dialog in the *Performance Analysis*. Should this not be visible, it is probably because the windows in IBExpert are set to *Cascading*. This can be easily solved by clicking the SQL/Procedure Editor dialog window to full-size (right-hand blue icon in the dialog title bar).

Logs

The *Log* page can be found in the SQL Editor and displays a list of qualified error messages etc. It shows what Firebird/InterBase did and when it did it in each respective SQL window.

Since IBExpert version 2006.14.10 it is also possible to log `EXECUTE BLOCK` statements.



Optimizing SQL statements

How does Firebird/InterBase process a query? SQLs are sent to the server, where the Optimizer first analyzes them: which tables are involved and which indices are the best to use etc., preparing them for execution. The server needs to select a strategy for creating a result set. The parser selects all tables involved and possible indices, usually selecting the index with the best selectivity, i.e. the one resulting in the smallest result set. Further information regarding index selectivity can be found in the *Index statistics* chapter.

The index statistics are compared in order to choose the most selective index for each `WHERE`, `JOIN` or `ORDER BY` condition.

In Firebird/InterBase it is possible to use more than one index, which isn't possible in some other databases. Compound indices should however only be used when really necessary. An `ORDER BY` is no reason for using an index, because an `ORDER BY` always has something to do with output formats. Usually `WHERE` conditions are used to limit the result set. `WHERE` and `JOIN` conditions should certainly be supported by an index. If you specify an `ORDER BY` over several fields, the index needs to be composed in exactly the same sequence as the `ORDER BY`. `ORDER BY` cannot accept compound indices composed of single indices.

The index plan is made during the preparation, and it is at this stage that the Optimizer selects in which sequence it will use the indices chosen. If the server cannot find a suitable index, it compiles a temporary sort quantity.

Take into consideration that when the `LIKE` command is used together with a wildcard (because you're searching a string that occurs somewhere in the field and not necessarily at the beginning), the Optimizer cannot use an index.

All table data needed is read from the cache. If the pages required are not already in the cache, they need to be transferred from the hard disk to the memory. This is the most time-consuming part of the operation for the Firbird server. If this process appears to be somewhat slow, check the parameters in `firebird.conf`. Please refer to *Temporary files* and *Memory configuration* in the *Firebird Administration using IBExpert* documentation.

After preparing your query, Firebird displays the query plan - which can be viewed in the SQL Editor's index plan, visible in the Plan Analyzer. If a lot of non-indexed reads (the red ones) appear in the Performance Analysis, it is often helpful to create some indices, reopen the query and check if it has been of help.

Following preparation, if no changes are to be made, the query can be executed.

When all data has been extracted and sorted accordingly the result set is sent back to the client sending the query. If only the first `n` records are to be fetched, the server only reads the required number of data pages. For certain commands such as `DISTINCT` and `GROUP BY`, the server must read all relevant data pages. So if `DISTINCT` or `GROUP BY` are not really necessary, don't use them!

Check the Performance Analysis and use it to compare different versions of the same SQL. Analyze the reads, writes and fetches! Reads and writes are typically 0 when InterBase/Firebird can operate in the cache. Fetches are the internal operations in InterBase/Firebird, so when one query is slower than the other, it may not be visible directly in the graphical view, for example when InterBase/Firebird creates external temporary sort files.

Use the Plan Analyzer to analyze how the Optimizer uses tables and indices to obtain the result set. If the word `SORT` is displayed, you should check whether improvements to the query or the indices are possible.

Although the Optimizer does a very good job, especially since Firebird 2.0, the programmer can often offer the Optimizer hints to help improve performance; depending on the task in hand, a small change in the SQL statement can often improve the speed immensely. For example, consider using the `+0` field parameter to deactivate indices with a low selectivity, perhaps derived tables can reduce the number of reads or fetches. Other factors affecting the performance of queries, such as hardware, OS and memory configuration, index selectivity, etc. can be referred to in *Firebird administration using IBExpert*.

Special features

The IBExpert SQL Editor has two special features that allow you to:

- Create a table from query results and populate it with data.
- Move data between two registered databases.

Creating a table from query results

As everyone knows it is possible to insert data into any table by executing the INSERT statement:

```
INSERT INTO TARGET_TABLE
   SELECT FIELD_1, FIELD_2 FROM SOURCE_TABLE
   WHERE SOMETHING_FIELD <> 5
```

However this will only work if the table TARGET_TABLE already exists in the database. IBExpert enables execution of this kind of statement even if the TARGET_TABLE does not exist in the database. First IBExpert notifies the user that TARGET_TABLE doesn't exist in the database and offers to create this table using query structure. If confirmed, IBExpert creates the TARGET_TABLE and then populates it with data from the SELECT.

A small example illustrates how this works, based on a SOURCE_TABLE with the following structure:

```
CREATE TABLE SOURCE_TABLE (
   ID INTEGER,
   SOME_TEXT VARCHAR(50),
   SOME_PRICE NUMERIC(15,4),
   SOME_DATE DATE);
```

When the following statement is executed:

```
INSERT INTO TARGET_TABLE
   SELECT * FROM SOURCE_TABLE
```

and there is no TARGET_TABLE in the database, IBExpert will create TARGET_TABLE as:

```
CREATE TABLE TARGET_TABLE (
   ID INTEGER,
   SOME_TEXT VARCHAR(50),
   SOME_PRICE NUMERIC(15,4),
   SOME_DATE DATE);
```

and after that inserts into this table records retrieved with the SELECT part.

Of course, it is possible to write different INSERT statements. For example:

```
INSERT INTO [TARGET_DATABASE].TARGET_TABLE
   SELECT ID, SOME_DATE FROM TEST_TABLE
```

In this case IBExpert will create table TARGET_TABLE as

```
CREATE TABLE TARGET_TABLE (
   ID INTEGER,
   SOME_DATE DATE);
```

Moving data between databases

IBExpert allows you to move data from one database to another by executing a special statement in the SQL Editor.

Syntax

```
INSERT INTO <database_alias>.<table_name>
   [(<columns_list>)]
   <select_statement>
```

Argument	Description
database_alias	Alias of a registered database. This must be enclosed in square brackets. This argument is case-insensitive so aliases My alias and MY ALIAS are equivalent.
table_name	Name of the table to be populated with data.
columns_list	List of columns in target table. This argument is not obligatory.
select_statement	Any SELECT statement.

Examples

The following statement moves data from SOURCE_TABLE of the current database into TARGET_TABLE of the database with the alias My test DB:

```
INSERT INTO [My test DB].TARGET_TABLE
   SELECT * FROM SOURCE_TABLE
```

If the table TARGET_TABLE doesn't exist in the target database, IBExpert will create it after your confirmation with the structure of the SOURCE_TABLE.

See also:
SQL Language Reference
Firebird 2 Language Reference Guide
Database Technology Articles
SQL basics

New SQL Editor

An additional SQL Editor can be opened using Tools / New SQL Editor, the respective icon in the Tools toolbar, or [Shift + F12].

The use of multiple SQL Editor windows does not affect the list of previous SQLs found on the History page, as this list is database dependent and not window dependent.

Query Builder

For those not yet competent in SQL, the Visual Query Builder is there to make life easier! It allows you to create and edit queries with multiple tables without previous knowledge of SQL, as well as prepare and execute queries, and view the results. This feature is unfortunately not included in the Personal Edition.

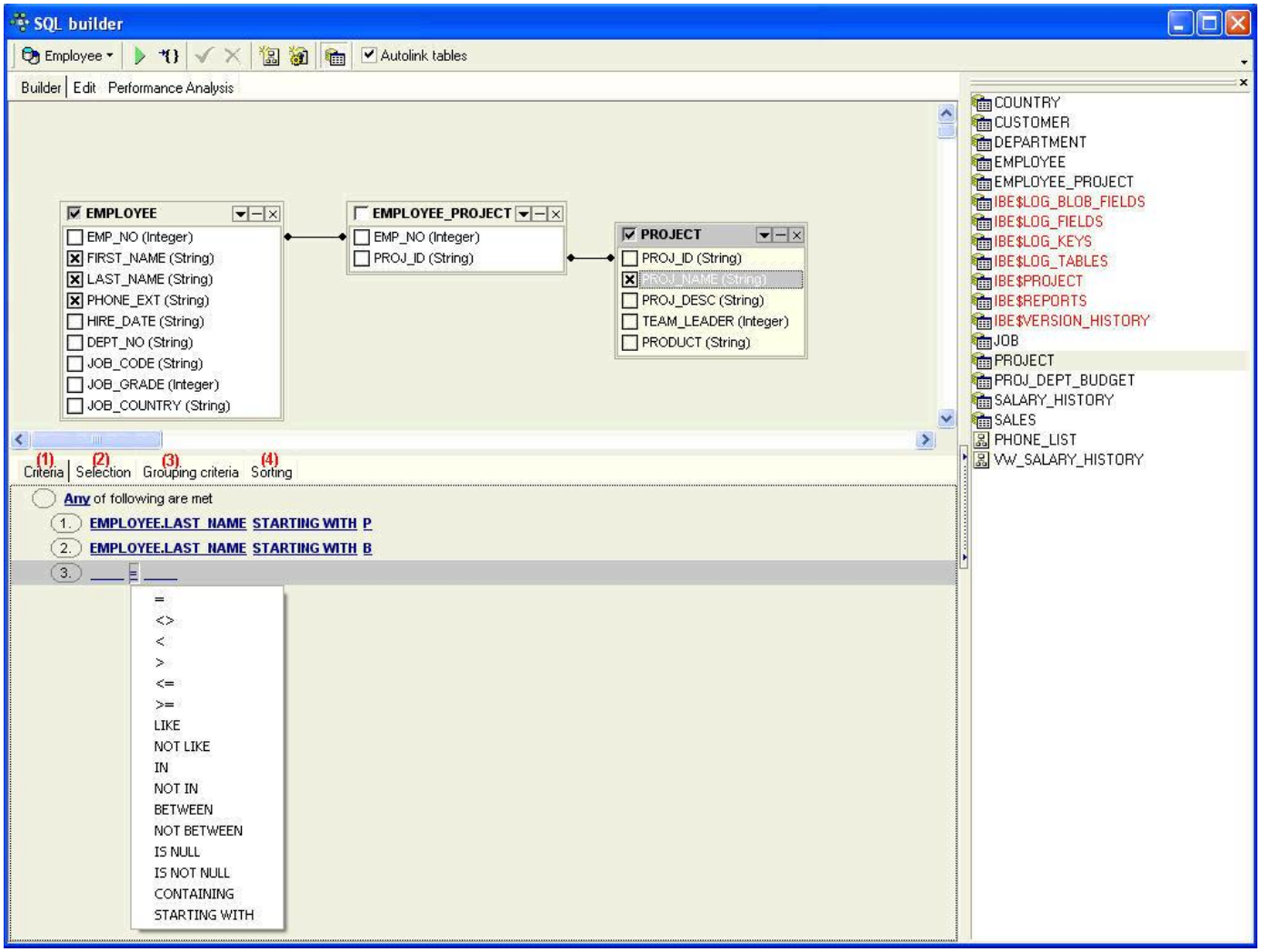
If you are new to Firebird/InterBase SQL, then please also refer to *Firebird Development using IBExpert* for a comprehensive introduction to SQL. The SQL Language Reference and the Firebird 2 SQL Reference Guide provide comprehensive references to all Firebird/InterBase SQL keywords, syntax and parameters.

The IBExpert Query Builder is started using the menu item Tools / Query Builder. It can also be started directly from the SQL Editor using [Ctrl + Shift + Alt + B] or the

icon.

A query can be built by simply moving the database objects (e.g. by dragging the desired table) from the right panel over to the left editing area. Objects may also be dragged and dropped from the DB Explorer and SQL Assistant into the code editor window. Since version 2004.2.26.1 this has been greatly improved.

When an object node(s) is dragged from the DB Explorer or SQL Assistant, IBExpert will offer various versions of text to be inserted into the code editor. It is also now possible to customize the highlighting of variables. Use the IBExpert Options menu item, Editor Options / Colors to choose color and font style for variables.



The required fields can be selected using the mouse. By clicking on the circle to the left of the table name, all fields are automatically highlighted. Tables can be linked, e.g. by key relationships, joins etc., using the mouse (click on the desired field in the first table and drag it across to the desired field in the second table). This creates a `JOIN`.

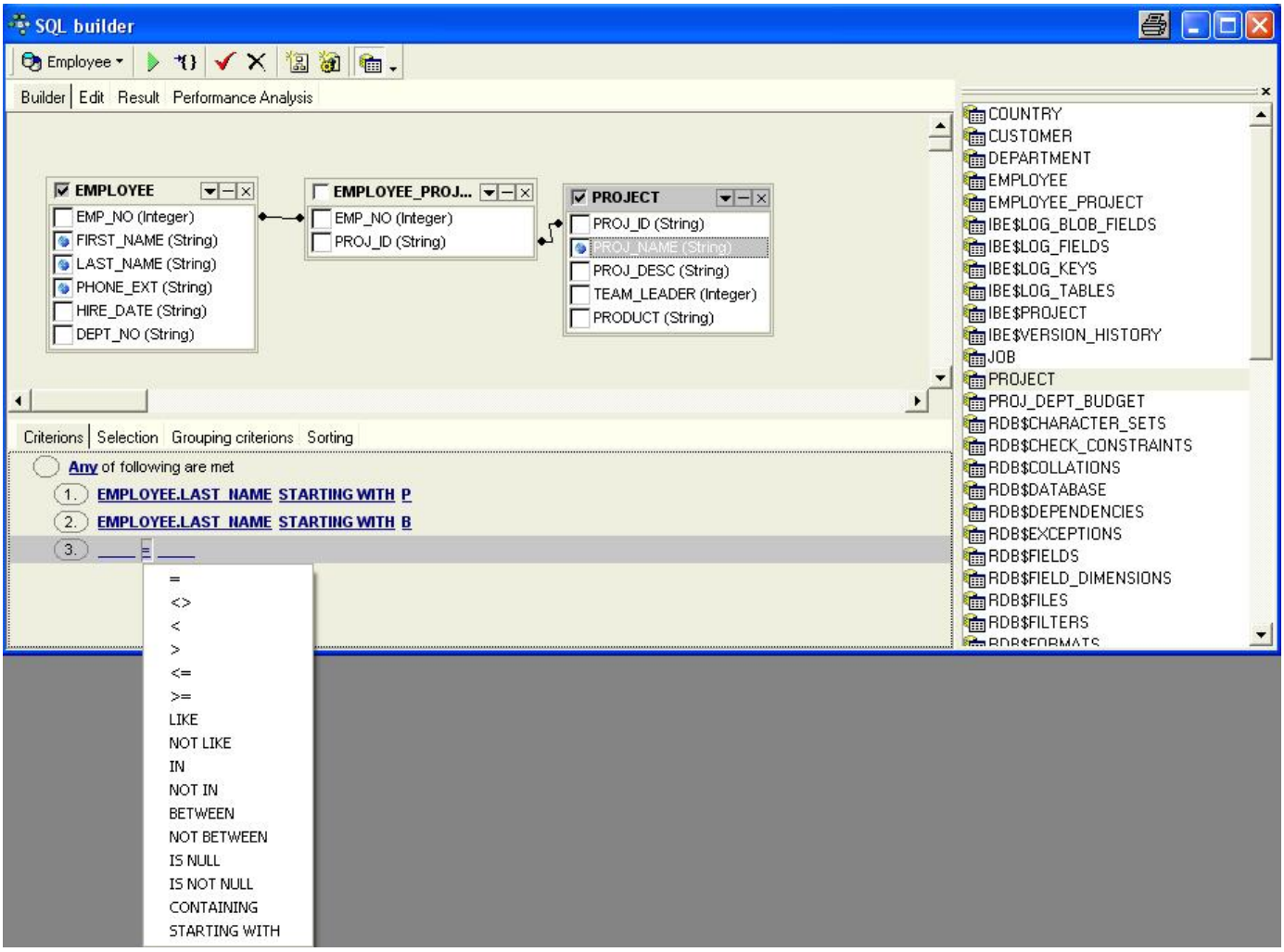
By double-clicking on the lines connecting two tables the option *Link Properties* appears, and the developer can specify from which table all of the information should be fetched (see JOIN for more information about joins).

Alternatively, a small context-sensitive menu appears when right-clicking on a line, offering not only the above mentioned option, but also the option to insert or delete point or to delete the link.

Check every field which is important for the result set and use [F9] or the respective icon to execute and view the results. For information regarding the *Results* page, please refer to SQL Editor / Results.

Conditions can be specified in the lower part of the Query Builder dialog using the options listed under the following tabs:

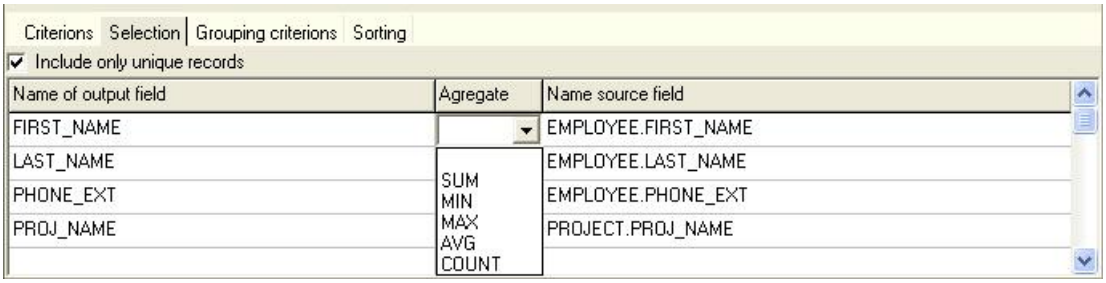
(1) Criteria



A simple condition string contains three fields: an argument, a condition and a second argument - if required for the condition. By clicking on the word *ALL* of *All of following are met*, it is possible to change this condition to *ALL*, *ANY*, *NONE*, or *NOT ALL*. By clicking on the ring to the left of *All of following are met*, it is possible to add a condition. Using [Shift + Enter] or right-clicking, fields can be selected from the specified tables. Alternatively a value can be manually entered. By clicking on the '=' sign a list of available conditions appears.

If you wish to view the SQL statement at any time, simply switch to the Edit page.

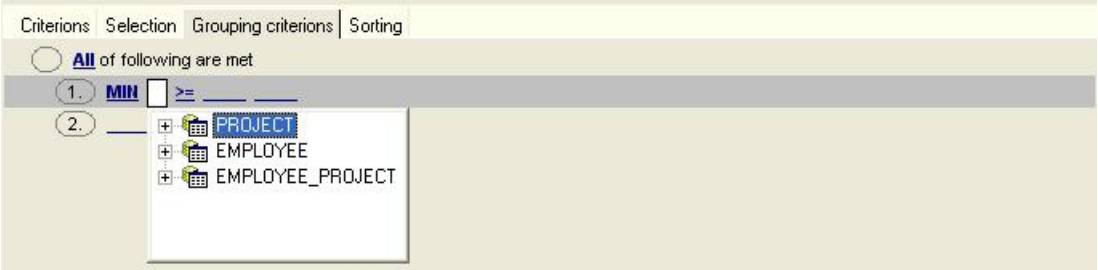
(2) Selection



An aggregate (SUM, MIN, MAX, AVG and COUNT) can be specified for individual fields if wished. For example, if a minimum or maximum order value needs to be determined; or the number of unpaid invoices. By double-clicking on a field in the builder area, the field source is automatically inserted. An output field name may be specified by double-clicking (or using the [Enter] key) on the first input field. The *Aggregate* pull-down list can be viewed by double-clicking or using the [Enter] key and downward arrow key, and an option selected.

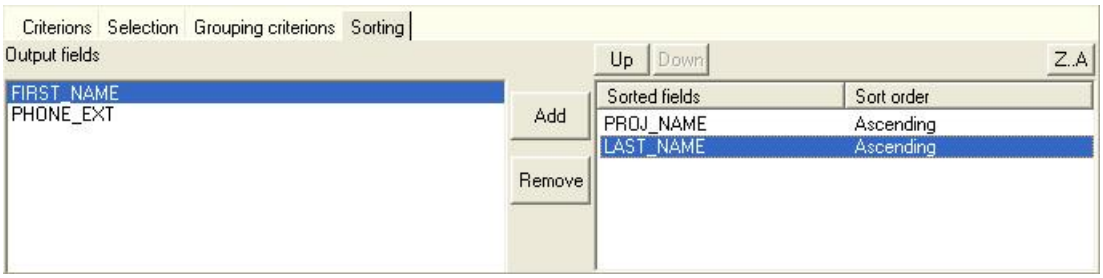
The *Include only unique records* checkbox eliminates duplicate records when checked.

(3) Grouping criteria



Again *ALL*, *ANY*, *NONE*, or *NOT ALL* of the specified conditions can be met. Here combined criteria can be determined, i.e. aggregate and comparative selection criteria.

(4) Sorting



Here the results can be sorted in ascending or descending order by one or more fields in order of priority. Simply move the field(s) to be used as the sorting criteria from the list on the left to the right-hand window, by selecting and clicking the Add button or using drag 'n' drop. Use the *A.Z -Z.A* button to specify ascending or descending order, and use the *Up* and *Down* buttons (when sorting by multiple fields) to specify sorting priority (i.e. which field should be sorted first).

When the query preparation is complete, it can be prepared [Ctrl + F9] and analyzed, and/or executed [F9] before finally committing.

In addition to the main *Builder* window, there is also an *Edit* page, displaying the query, resulting from the drag 'n' drop and condition specification in the main builder window, as SQL text. This is, in effect, the same as the SQL Editor's main Edit window. It can be edited directly, if wished, and all changes are displayed on the other Query Builder pages.

A *Results* page appears following query execution, displaying the returned data resulting from the query. A *Filter panel* can also be blended into the dialog to aid data navigation and allow further filtering. For more information, please refer to the SQL Editor's Edit page and Filter Panel.

The Plan Analyzer is displayed following query execution and shows how Firebird/InterBase approaches a query, e.g. with `SORTS`, `JOINS` etc, which tables and indices are used. The information is shown in the lower panel in a tree structure with statistics.

The *Performance Analysis* displays information showing much effort was required by InterBase/Firebird to carry out an executed query or procedure. For more information please refer to the SQL Editor's Performance Analysis.

Visual Query Builder is ideal for the beginner, although somewhat limited for more advanced work; complex queries should be performed in the SQL Editor.

See also:
Create View or Procedure from Select
SQL Language Reference
Stored Procedure
Toolbar Query Builder
Toolbar SQL Editor
Toolbar Tools
SQLBasics

DataAnalysis / OLAP

The IBExpert Tools menu item, Data Analysis, is new to IBExpert version 2004.10.30.1.

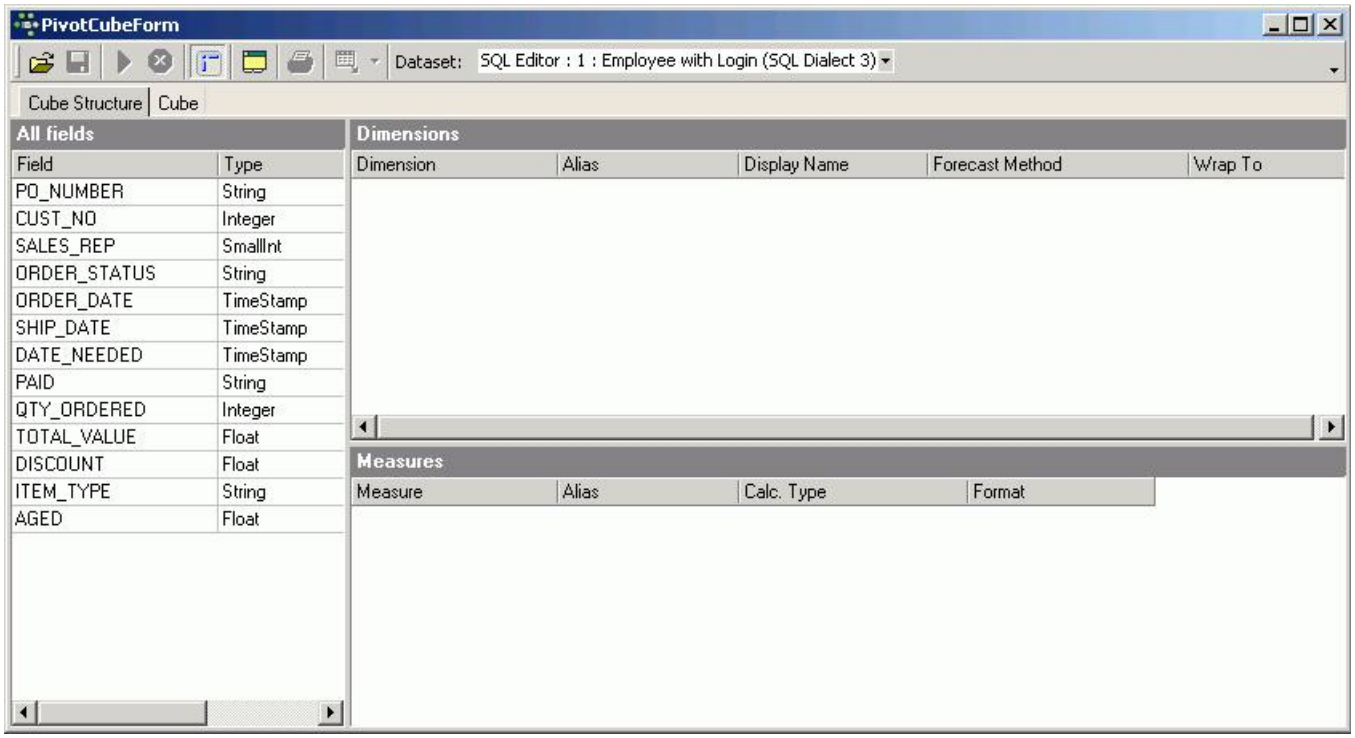
It is an ideal OLAP and data warehouse component, for analyzing data in the database quickly and easily. This sophisticated module can be used to build cubes, manage dimensions and measures, the technology being based on the building of multidimensional data sets - so-called OLAP cubes. It includes a powerful filtering system, enabling not only dimensions but also measures to be filtered. This feature is unfortunately not included in the Personal Edition.

The *PivotCubeForm* can be opened using the IBExpert Tools menu, or started directly from the SQL Editor / Results page, the Table Editor / Data page or the View Editor / Data page, using the *Data Analysis* icon:

We will illustrate the functionalities and options available in the *Pivot Cube*, using the following simple `SELECT` command, executed in the SQL Editor:

```
SELECT * FROM SALES;
```

By clicking the *Data Analysis* icon on the SQL Editor / Results page, the *PivotCubeForm* is opened:



The *PivotCubeForm* has its own toolbar (please refer to Data Analysis toolbar for further information), and contains two pages: Cube Structure and Cube.

Cube Structure

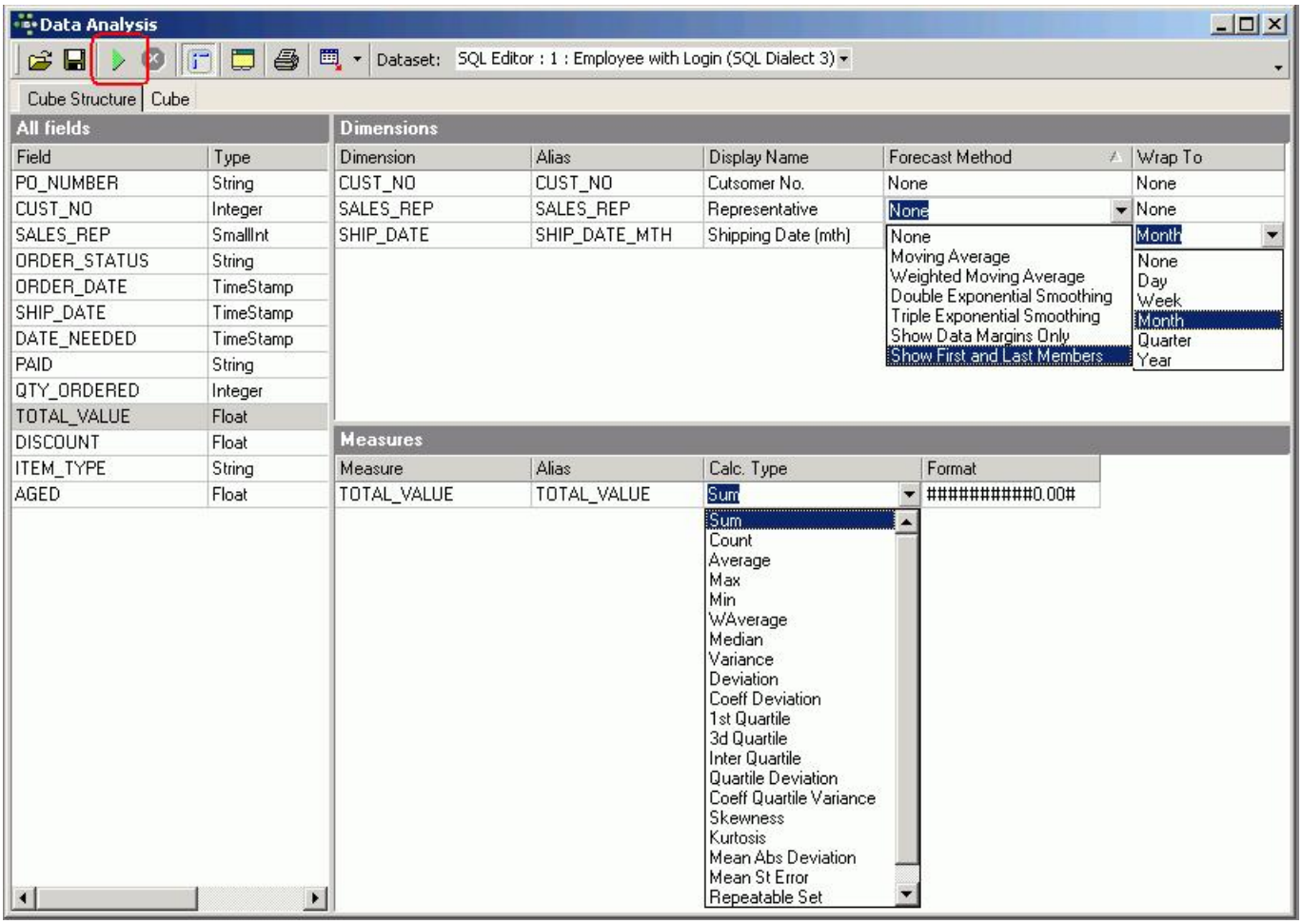
The first page has three main areas:

1. **All Fields:** This automatically displays all data set fields displayed on the SQL Editor's *Results* page.
2. **Dimensions:** what is to be analyzed and displayed. The field order is at this stage irrelevant.
3. **Measures:** which values are to be analyzed and displayed. IBExpert Data Analysis permits use of any data types as measures; the only restriction being that non-numeric data types can only use the `ctCount` aggregate type.

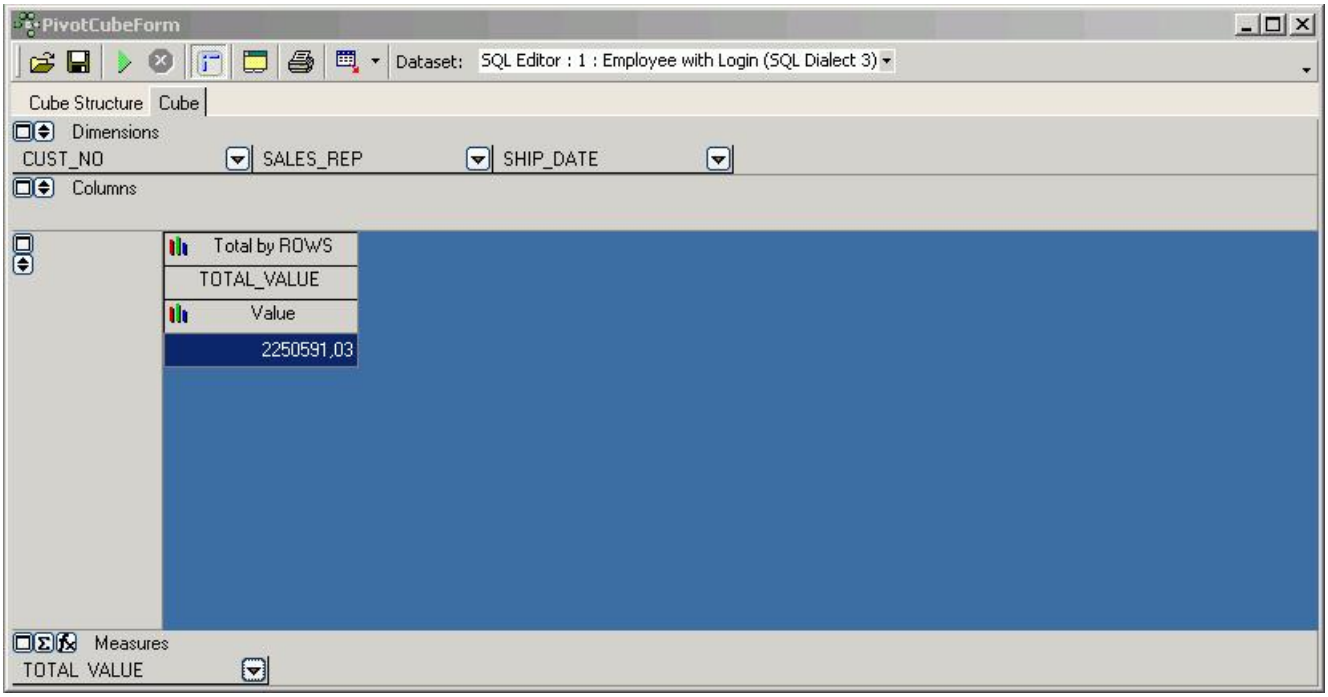
As with all IBExpert grids, columns can be sorted in ascending and descending order by simply clicking on the column headers.

Fields can be selected from the *All Fields* panel and dragged 'n' dropped into the *Dimensions* panel. For example, `CUST_NO`, `SALES_REP` and `SHIP_DATE`, the shipping date also being grouped by month. The *Alias* names and *Display Names* can be manually altered as wished, and the *Forecast Method* and *Wrap To* periods can be selected from the pull-down lists. (Simply click on the field where a selection is to be made, and click the black downward arrow on the right of the field to open the list of available options.) Multiple field selection/deselection is possible since IBExpert version 2006.12.11.

The TOTAL_VALUE field can be dragged 'n' dropped from the *All Fields* panel into the *Measures* area. Again select *Calculation Type* from the options offered in the pull-down list; the numeric *Format* can be manually altered if desired:



And then the cube can be generated using the *Build Cube* icon or [F9] (see illustration above) and displayed on the *Cube* Page:



Cube

The second page in the *PivotCubeForm* displays the cube itself in the third of four areas, so-called toolbars:

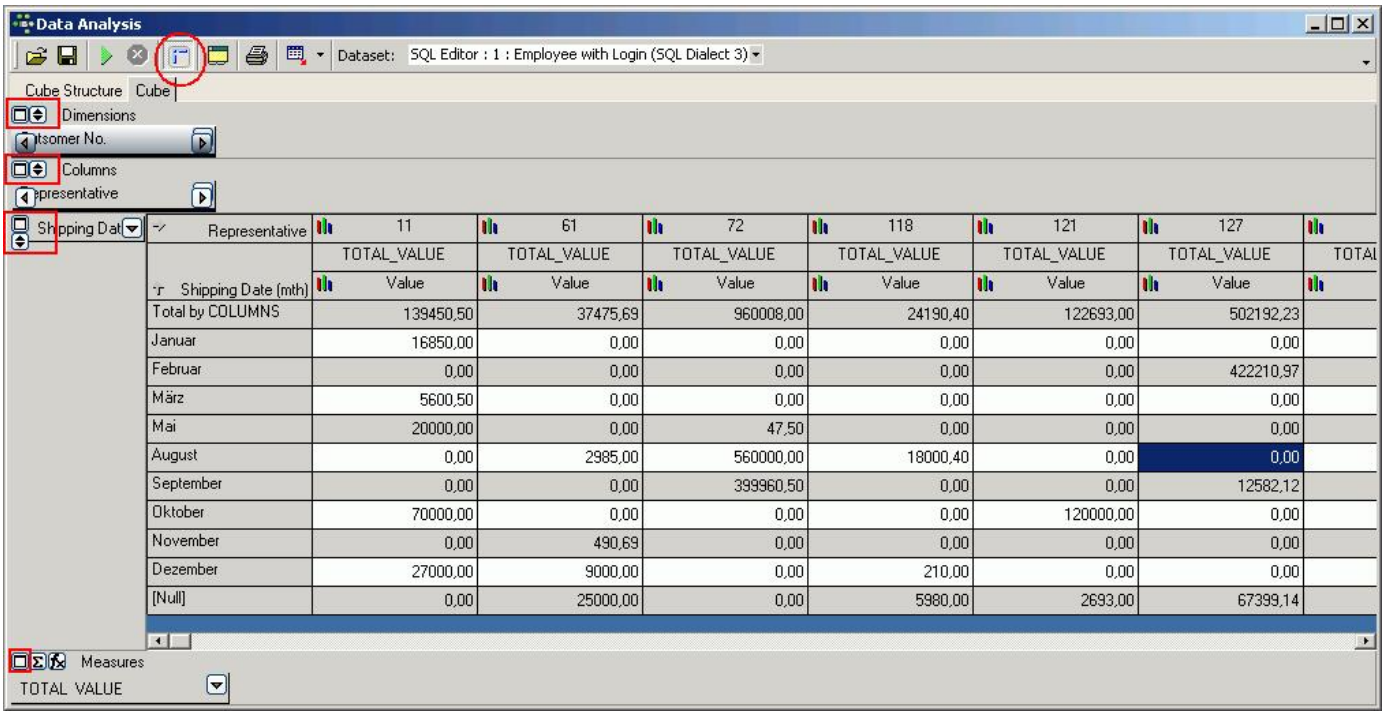
1. Dimensions
2. Columns
3. Main display area

4. Measures - the order of the items here determines how the data is displayed in the pivot grid.

These areas can all be opened or closed, by clicking on the small square buttons in the upper left-hand corner of each area (see rectangular marked symbols in the illustration below). The arrow buttons can be used to adjust the size of the expanded areas, and display/hide the filter, which allows values to be searched and viewed for individual data sets.

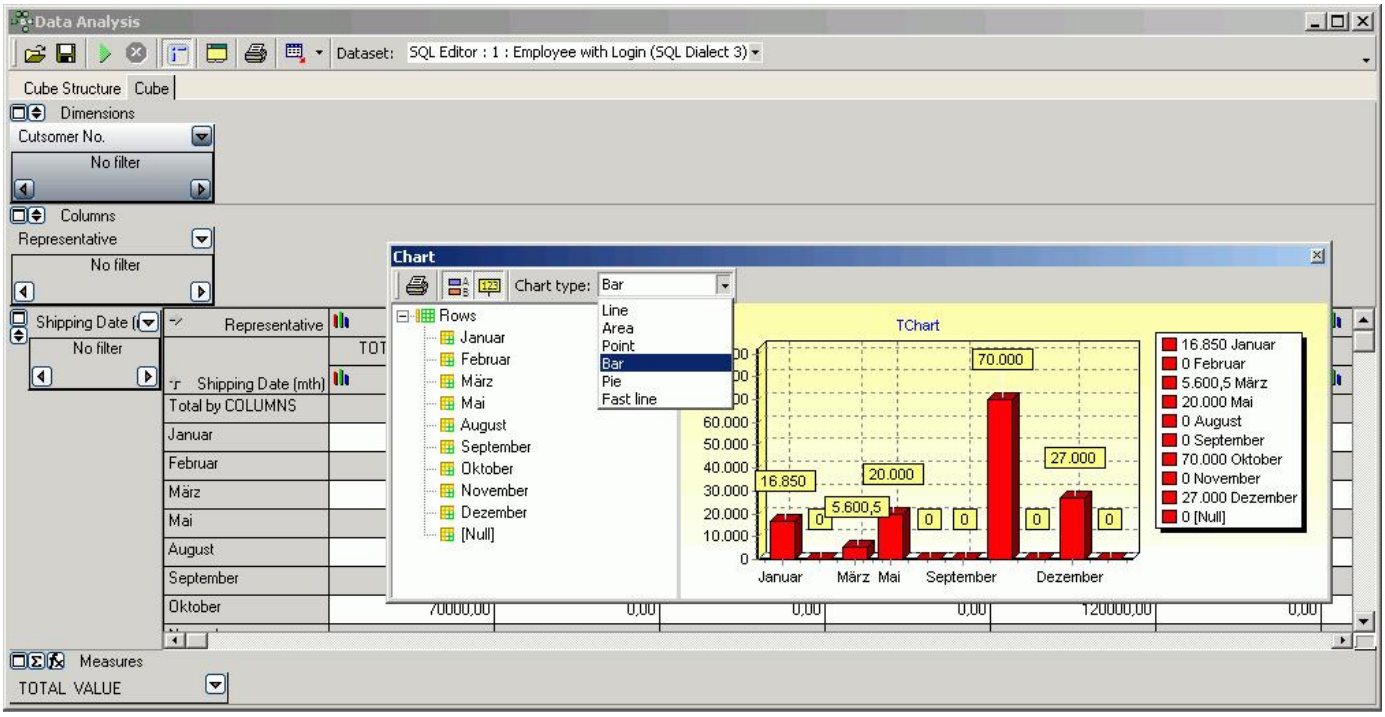
The toggle toolbars on/off icon (see circled icon below) can be used to remove these areas completely leaving just the main blue display area, or blending them in again.

It is now possible to generate a summary, for example, which customer or which sales representative has generated which sales revenue. Or even which representative (column) has generated which revenue in which month:

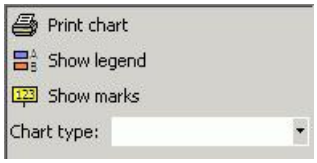


Shipping Date (mth)	11	61	72	118	121	127
	TOTAL_VALUE	TOTAL_VALUE	TOTAL_VALUE	TOTAL_VALUE	TOTAL_VALUE	TOTAL_VALUE
	Value	Value	Value	Value	Value	Value
Total by COLUMNS	139450,50	37475,69	960008,00	24190,40	122693,00	502192,23
Januar	16850,00	0,00	0,00	0,00	0,00	0,00
Februar	0,00	0,00	0,00	0,00	0,00	422210,97
März	5600,50	0,00	0,00	0,00	0,00	0,00
Mai	20000,00	0,00	47,50	0,00	0,00	0,00
August	0,00	2985,00	560000,00	18000,40	0,00	0,00
September	0,00	0,00	399960,50	0,00	0,00	12582,12
Oktober	70000,00	0,00	0,00	0,00	120000,00	0,00
November	0,00	490,69	0,00	0,00	0,00	0,00
Dezember	27000,00	9000,00	0,00	210,00	0,00	0,00
[Null]	0,00	25000,00	0,00	5980,00	2693,00	67399,14

The data can be displayed graphically with a simple mouse click. Simply click on the desired graphics icon to the left of the *Measures* (here: *Representative* or *Shipping Date (mth)*):

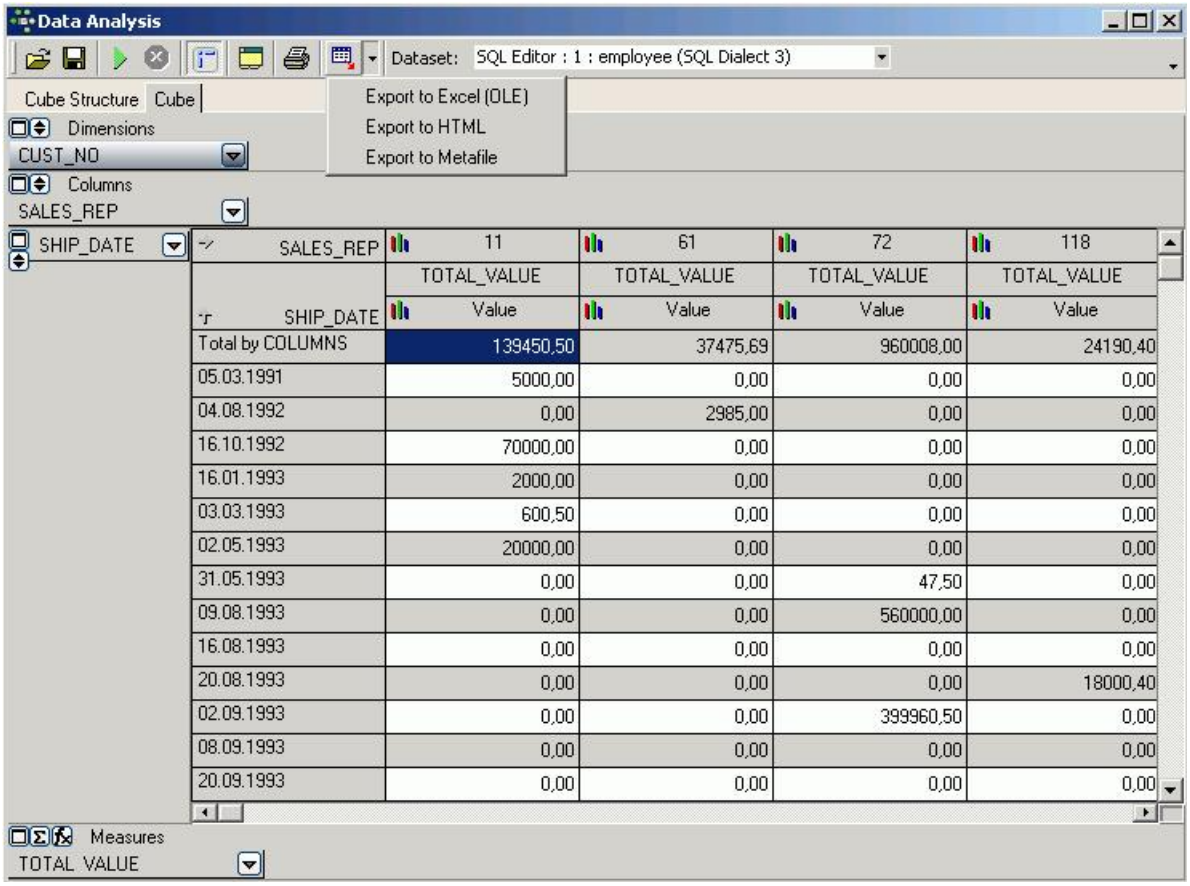


The *Graphics* window has its own mini toolbar, with the following options:



allowing the graph type to be altered, the legend and notes to be blended in or out, and enabling the graph to be printed. There are numerous options to add functional values and formulae. Please refer to Cube Manager and Calculated Measures Manager for further information.

The data and analyses generated can be saved as *.CUB files, or exported to Excel (OLE), HTML or metafile. Simply click the small black arrow directly to the right of the *Export* icon, and select from the list:

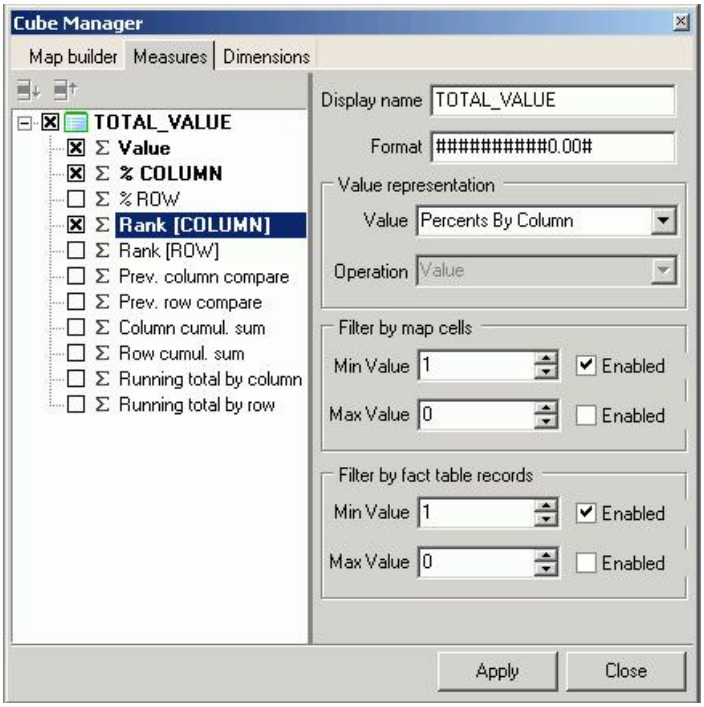


They can even be quickly and easily printed - simply click the *Print* icon (or [Ctrl + P]), to go to the *Print Preview*, where the page layout and appearance may be modified before finally printing.

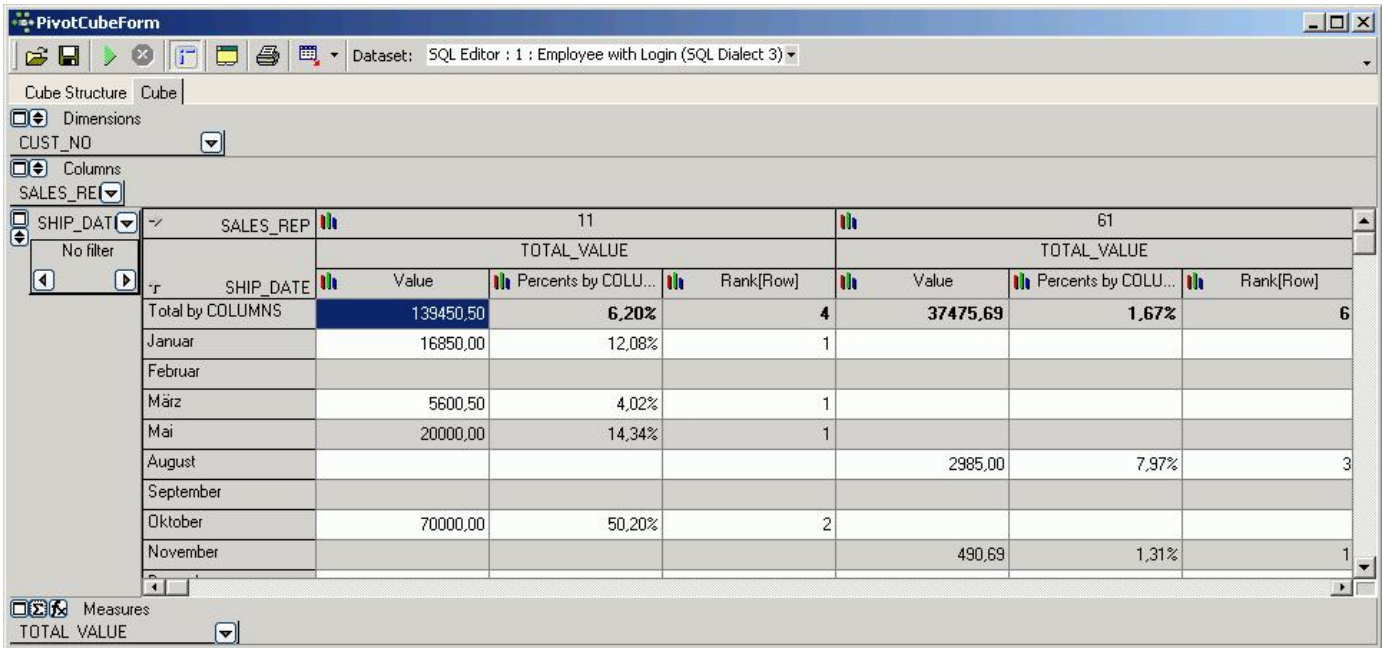
In fact, IBExpert's Data Analysis offers innumerable possibilities to define reports quickly and easily, or to simply collate the data material.

Data Analysis Cube Manager

The Cube Manager can be opened using the *PivotCubeForm* icon, or by clicking the *Sum* button in the bottom left hand corner of the *Measures* toolbar on the *Cube* page. This can be used to include certain alternative additional values. For example, alter the view to percentage column values:



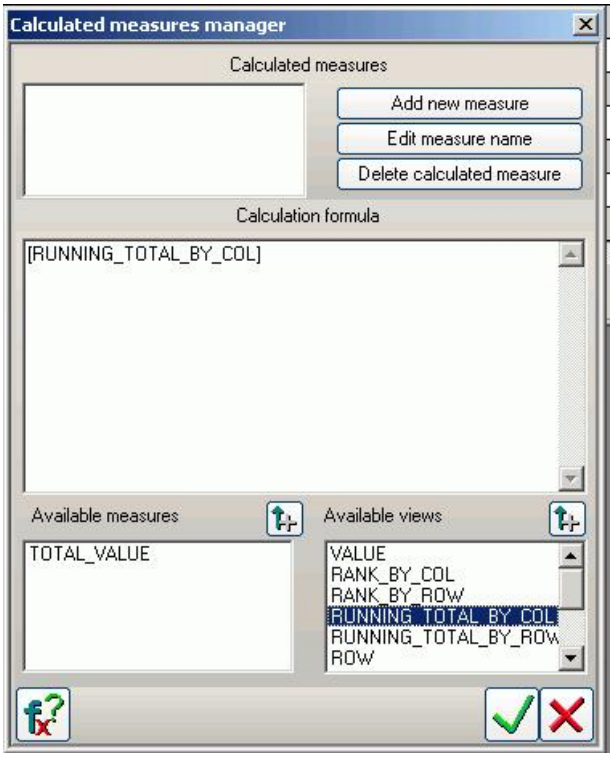
Click the *Apply* icon to view the results:



Depending on what you wish to see, it is possible to specify an ascending or descending order by simply clicking on the column headers.

Data Analysis Calculated Measures Manager

It is possible to integrate certain function values by clicking on the *Function* button in the bottom left hand corner of the Measures toolbar on the *Cube* page, to open the *Calculated Measures Manager*.



You can add new measures and edit or delete existing measures.

A new measure name can be added by clicking the *Add New Measure* button and inserting a name. A template automatically appears in the *Calculation Formula* input area. This can be completed manually, the *Available Measures* (bottom left-hand list) and *Available Views* (bottom right-hand list) can be inserted simply by double-clicking on the measure name, or clicking the [upward arrow +] button to the right of the *Available Measures* or *Available Views* headings.

When you are satisfied with your specifications, simply click the

button. You will now see both the original evaluation and the new calculated measure name displayed in the status bar. By clicking the black arrow to the right of these names, the *Cube Manager* is automatically opened, displaying the specifications made for the selected measure.

Simply re-click the *Function* button to reopen the *Calculated Measures Manager*, to make additional alterations, insertions or deletions as required.

Script Executive

The Script Executive can be used to view, edit and execute SQL scripts. It can be started from the IBExpert Tools menu, using the respective icon in the [IBExpert ToolbarsTools toolbar or using [Ctrl + F12]. It is used for SQLs covering several rows. The Script Executive can both read and execute scripts.

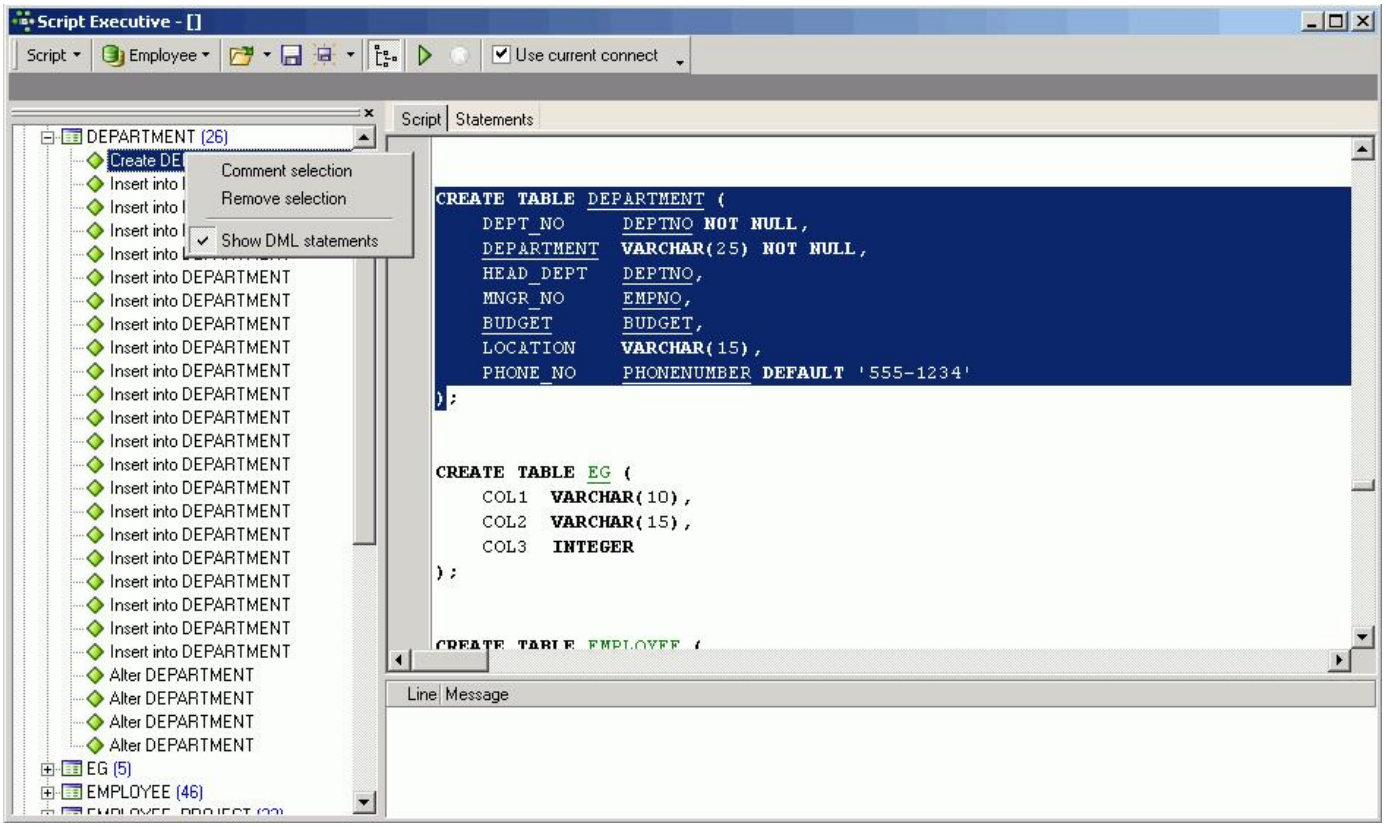
Although InterBase/Firebird can also process such procedure definitions in the SQL Editor, it is recommended to use the Script Executive for more complex work, as it can do much more than the SQL Editor. There is a wealth of script language extensions including conditional directives, and it can also be used for executing multiple scripts from a single script.

The main advantage of the Script Executive is that it displays all DDL and DML scripts of a connected database. And since IBExpert version 2006.01.29 the Script Executive always uses the default client library specified in the IBExpert Options menu item Environment Options / Preferences under *Default Client Library*, unless it is overridden using the `SET CLIENTLIB` command.

The *Script Explorer* (the left-hand panel) displays all database objects used in the current script in a tree structure. It even allows you to find a script part rapidly by clicking on the object in the tree. The *Script Explorer* can be blended in and out using the respective icon on the Script Executive toolbar. SQL scripts can be loaded from and saved to file if wished. Since IBExpert version 2007.09.25 the *Script Explorer* also displays IBEBlocks and Firebird blocks.

Objects may be dragged and dropped from the DB Explorer and SQL Assistant into the code editor window. And since version 2004.2.26.1 this has been greatly improved. When an object node(s) is dragged from the DB Explorer or SQL Assistant, IBExpert will offer various versions of text to be inserted into the code editor. It is also now possible to customize the highlighting of variables. Use the IBExpert Options menu item, Editor Options / Colors to choose color and font style for variables.

Complete scripts can be transferred from the SQL Editor or extracted directly from the Extract Metadata Editor into the Script Executive using the relevant menu items (please refer directly to these subjects for further details).



In IBExpert version 2007.12.01 the option was introduced to display DML statements in the Script Explorer tree. Simply right-click to open the context-sensitive menu and check/uncheck as wished.

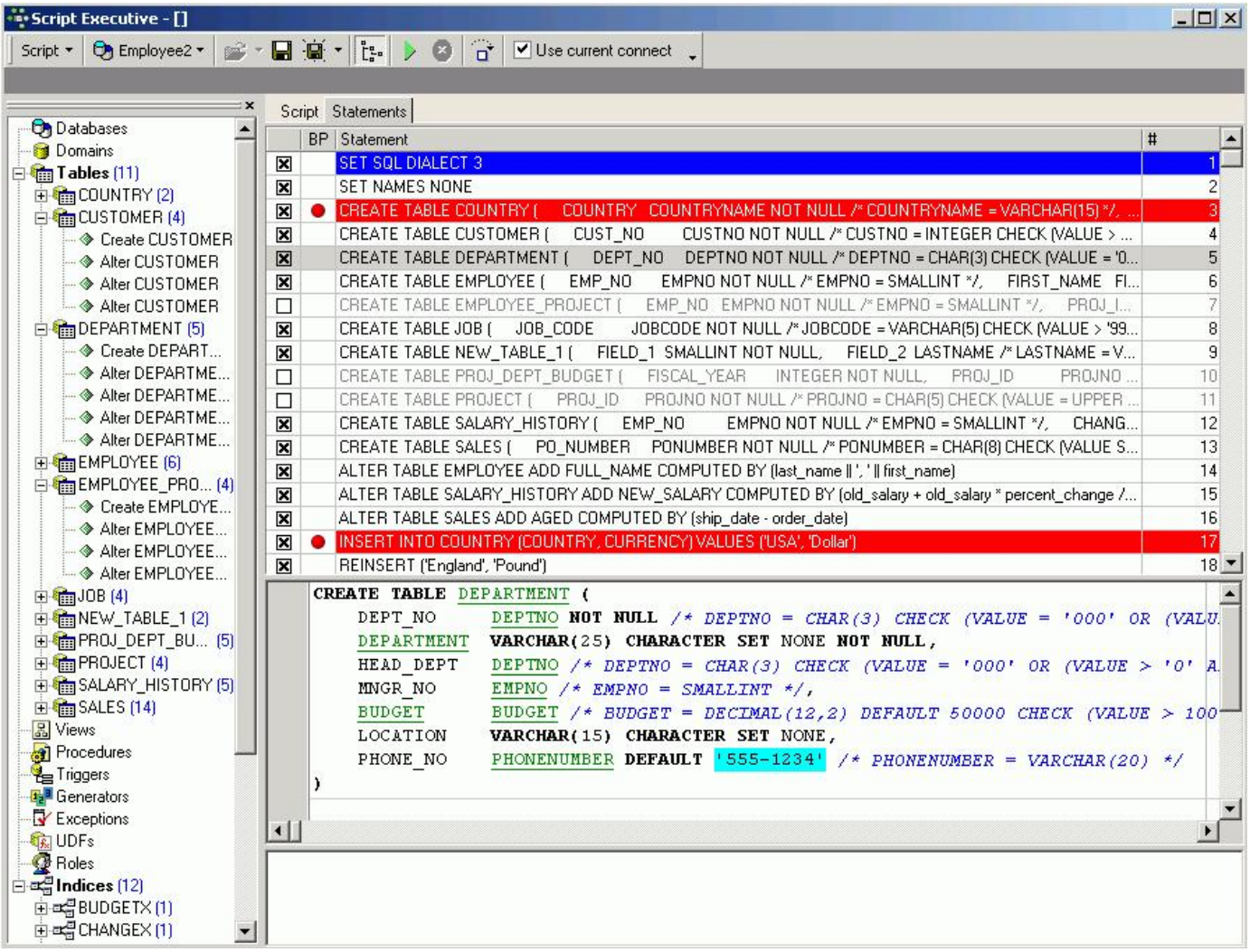
The *Script Type* may be selected from the Script Executive toolbar pull-down list (options include *InterBase/Firebird* or *MySQL*).

The *Script* page includes other features, such as code completion (please refer to Code Insight for details) - familiar from the SQL Editor. The SQL Editor menu can be called by right-clicking in the script area. Following statement execution, the *Script* page displays any errors highlighted in red. Using the

icon, the script can be executed step by step.

Any errors appearing in the lower *Messages* box may be saved to file if wished, using the right-click menu item *Save Messages Log ...*

The *Statements* page displays a list of individual statements in grid form:



These statements may be removed from the script simply by unchecking the left-hand boxes. One, several or all statements may be checked or unchecked using the right-click menu. Breakpoints can be specified or removed simply by clicking (or using the space bar) to the left of the selected statement in the *BP* column.

IBExpert version 2004.04.01.1 includes added support for the `EXECUTE BLOCK` statement (Firebird 2).

The following features were introduced in IBExpert version 2005.03.12.1:

Executing of `INSERT/UPDATE/EXECUTE PROCEDURE` statements `WITHOUT` parameters is up to 10 times faster now. Added support for the following Firebird 2 features:

```
CREATE SEQUENCE
DROP SEQUENCE
ALTER SEQUENCE
```

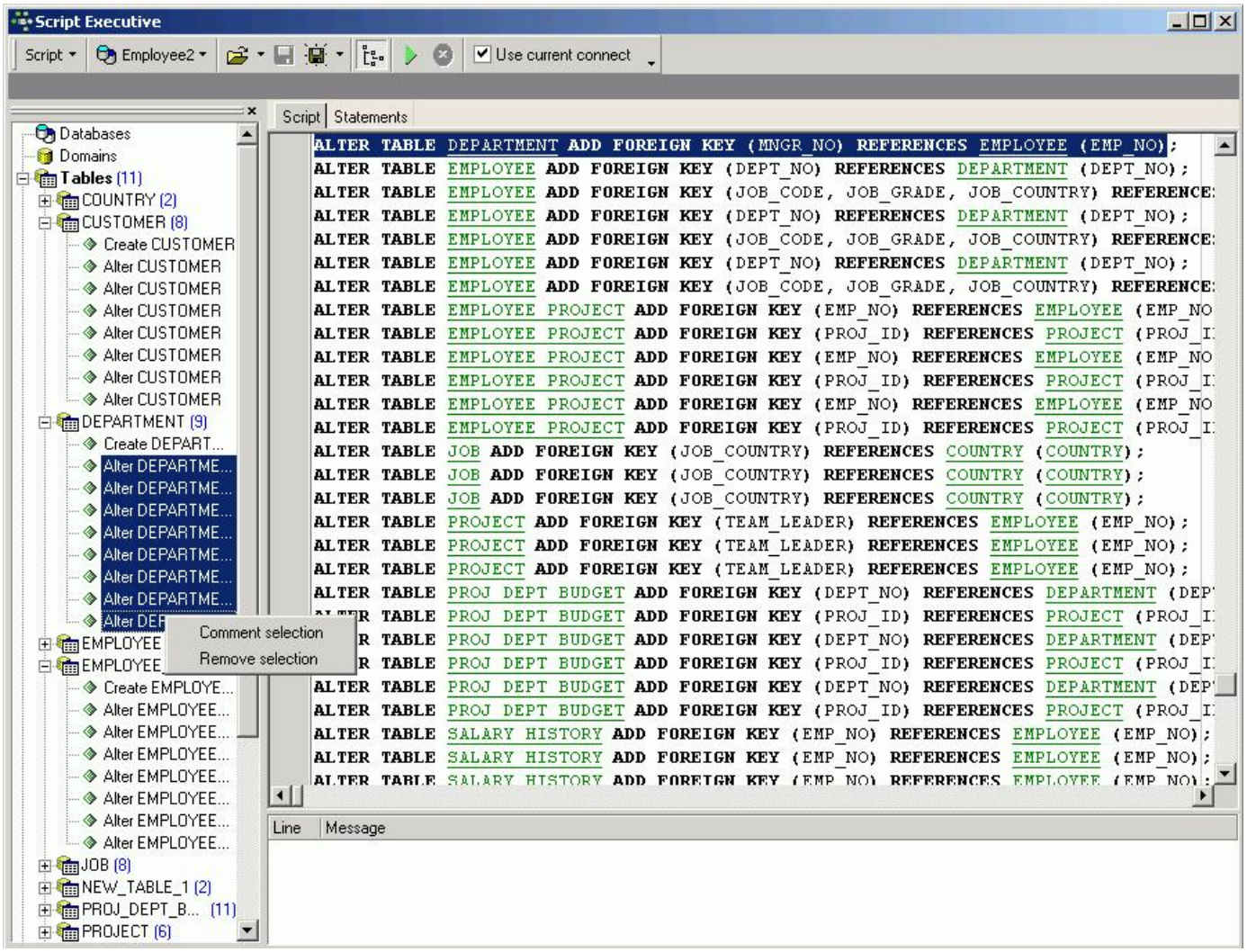
Extended syntax of `OUTPUT` command. Please refer to `OUTPUT` for futher information and examples.

Introduced in IBExpert version 2005.09.25:

- Added support for `COMMENT ON` statements (Firebird 2).
- Added possibility to delete/comment several script statements at once. Simply select the items to be deleted/commented in the *Script Explorer*, and choose the corresponding action in the right-click context menu.

Introduced in IBExpert version 2006.10.14 (also in IBEScript):

Added support for `BATCH BEGIN/BATCH EXECUTE` statements (InterBase 2007). If the server does not support this feature all statements between `BATCH BEGIN` and `BATCH EXECUTE` will be executed in the regular way.



```
ALTER TABLE DEPARTMENT ADD FOREIGN KEY (MNGR_NO) REFERENCES EMPLOYEE (EMP_NO);
ALTER TABLE EMPLOYEE ADD FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEPT_NO);
ALTER TABLE EMPLOYEE ADD FOREIGN KEY (JOB_CODE, JOB_GRADE, JOB_COUNTRY) REFERENCE
ALTER TABLE EMPLOYEE ADD FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEPT_NO);
ALTER TABLE EMPLOYEE ADD FOREIGN KEY (JOB_CODE, JOB_GRADE, JOB_COUNTRY) REFERENCE
ALTER TABLE EMPLOYEE ADD FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEPT_NO);
ALTER TABLE EMPLOYEE ADD FOREIGN KEY (JOB_CODE, JOB_GRADE, JOB_COUNTRY) REFERENCE
ALTER TABLE EMPLOYEE_PROJECT ADD FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO
ALTER TABLE EMPLOYEE_PROJECT ADD FOREIGN KEY (PROJ_ID) REFERENCES PROJECT (PROJ_I
ALTER TABLE EMPLOYEE_PROJECT ADD FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO
ALTER TABLE EMPLOYEE_PROJECT ADD FOREIGN KEY (PROJ_ID) REFERENCES PROJECT (PROJ_I
ALTER TABLE EMPLOYEE_PROJECT ADD FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO
ALTER TABLE EMPLOYEE_PROJECT ADD FOREIGN KEY (PROJ_ID) REFERENCES PROJECT (PROJ_I
ALTER TABLE JOB ADD FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY);
ALTER TABLE JOB ADD FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY);
ALTER TABLE JOB ADD FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY);
ALTER TABLE PROJECT ADD FOREIGN KEY (TEAM_LEADER) REFERENCES EMPLOYEE (EMP_NO);
ALTER TABLE PROJECT ADD FOREIGN KEY (TEAM_LEADER) REFERENCES EMPLOYEE (EMP_NO);
ALTER TABLE PROJECT ADD FOREIGN KEY (TEAM_LEADER) REFERENCES EMPLOYEE (EMP_NO);
ALTER TABLE PROJ_DEPT_BUDGET ADD FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEP
ALTER TABLE PROJ_DEPT_BUDGET ADD FOREIGN KEY (PROJ_ID) REFERENCES PROJECT (PROJ_I
      R TABLE PROJ_DEPT_BUDGET ADD FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEP
      R TABLE PROJ_DEPT_BUDGET ADD FOREIGN KEY (PROJ_ID) REFERENCES PROJECT (PROJ_I
ALTER TABLE PROJ_DEPT_BUDGET ADD FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEP
ALTER TABLE PROJ_DEPT_BUDGET ADD FOREIGN KEY (PROJ_ID) REFERENCES PROJECT (PROJ_I
ALTER TABLE SALARY_HISTORY ADD FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO);
ALTER TABLE SALARY_HISTORY ADD FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO);
ALTER TABLE SALARY_HISTORY ADD FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO);
```

Executing multiple scripts from a single script

Simply use the following syntax:

```
connect 'server:c:\my_db.gdb' ...;

input 'c:\my_scripts\f2.sql';
input 'c:\my_scripts\f1.sql';
input 'c:\my_scripts\f3.sql';
```

Create multiple CSV files from a script

The following is an example illustrating the creation of multiple csv files from a script:

```
shell del C:\list.dat nowait;    --deleting the old file
shell del C:\*.csv nowait;    --deleting the old csv files

connect 'localhost:C:\employee.fdb' user 'SYSDBA' password 'masterke';
--connect to employee example database

output 'C:\list.dat';    --record the following result as a simple text file,

based on each unique employee, we create a new output ...;select ... ;output; line in the dat file

SELECT distinct
'OUTPUT C:\'||EMPLOYEE.last_name||'.csv delimiter ´´;´´;'||
'SELECT distinct EMPLOYEE.last_name, customer.customer,customer.phone_no '||
'FROM SALES INNER JOIN CUSTOMER ON (SALES.CUST_NO = CUSTOMER.CUST_NO) '||
'INNER JOIN EMPLOYEE ON (SALES.SALES_REP = EMPLOYEE.EMP_NO) where

EMPLOYEE.last_name=´´´||EMPLOYEE.last_name||´´´;'||
'OUTPUT;'
FROM SALES INNER JOIN CUSTOMER ON (SALES.CUST_NO = CUSTOMER.CUST_NO) INNER JOIN EMPLOYEE ON

(SALES.SALES_REP = EMPLOYEE.EMP_NO);
```

```
output;      --close the dat file
input 'C:\list.dat';  --execute them
```

The data file is created automatically.

The outer query gets one record for each employee, in the inner select, all phone numbers for the employees' customers are selected.

Please also refer to IBEBlockExamples Importing data from a CSV file.

Script Language Extensions

Script language extensions are unique to IBExpert, and offer the developer a number of additional language options. These include, among others, conditional directives, DESCRIBE database objects, as well as SET, SHELL, INSERTEX, OUTPUT and RECONNECT.

Conditional Directives

Conditional directives control conditional execution of parts of the script. Four types of conditional directives are supported:

- $IFEXISTS
- $IFIBEVERSION
- $IFNOTEXISTS ($IFNEXISTS)
- $ELSE
- $ENDIF

IBExpert version 2005.12.04 introduced added support for the new conditional directive:

```
{$IfExists INDEX <index_name>}
{$IfNotExists INDEX <index_name>}
```

$IFEXISTS

This tests the existence of the specified database object or data and executes the following block of the script if the object or data do exist in the database.

Syntax

1. `{$IFEXISTS DOMAIN|TABLE|VIEW|TRIGGER|PROCEDURE|EXCEPTION|GENERATOR|UDF|ROLE object_name}`

2. `{$IFEXISTS select_statement}`

Example

The following script drops the exception `InvalidUserID` if it exists in the database:

```
{$IFEXISTS EXCEPTION "InvalidUserID"}

   DROP EXCEPTION "InvalidUserID";
```

The next script alters a procedure:

```
{$IFEXISTS SELECT RDB$PROCEDURE_NAME
           FROM RDB$PROCEDURES
           WHERE RDB$PROCEDURE_NAME = 'GETDBVER'}

   ALTER PROCEDURE GETDBVER
   RETURNS (
       VER INTEGER)
   AS
   begin
     ver = 2;
     suspend;
   end;
```

$IFIBEVERSION

The `$IfIBEVersion` conditional directive was implemented in IBExpert version 2007.07.18. This allows you to check the current version of IBExpert/IBEScript.

Syntax

```
      {$IfIBEVersion <relational_operator> <version_number>}
    ...
    ...                <relational_operator> = < | > | =< | >= | = | <> |
```

`<version_number>` - version number string without quote char.

Example

```
{$IfIBEVersion < 2007.7.16.0}
  execute ibeblock
```

```
as
begin
  ibec_ShowMessage('Please, update your version of IBExpert/IBEScript!');
end;
quit;
```

$IFNOTEXISTS ($IFNEXISTS)

This tests the existence of the specified database object or data and executes the following block of the script if the object or data does not exist in the database.

Syntax

1. `{$IFNOTEXISTS DOMAIN|TABLE|VIEW|TRIGGER|PROCEDURE|EXCEPTION|GENERATOR|UDF|ROLE object_name}`

2. `{$IFNOTEXISTS select_statement}`

Example

The following script creates a table `CUSTOMERS` if there is no such table in the database:

```
{$IFNOTEXISTS TABLE CUSTOMERS}

  CREATE TABLE CUSTOMERS (
    ID          INTEGER NOT NULL PRIMARY KEY,
    FIRST_NAME  VARCHAR(30),
    MIDDLE_NAME VARCHAR(30),
    LAST_NAME   VARCHAR(30));
```

The next script creates an exception:

```
{$IFNOTEXISTS SELECT RDB$EXCEPTION_NAME
              FROM RDB$EXCEPTIONS
              WHERE RDB$EXCEPTION_NAME = 'InvalidUserID'}

  CREATE EXCEPTION "InvalidUserID" 'Invalid User Identifier!';
```

$ELSE

Switches between executing and ignoring the script part are delimited by the previous or and the next .

Syntax

```
{$ELSE}
```

Example

The following script tests the existence of domain `DOM_BOOL` in the database. If domain `DOM_BOOL` cannot be found in the database it will be created. If domain `DOM_BOOL` already exists in the database it will be altered.

```
{$IFEXISTS DOMAIN DOM_BOOL}

  ALTER DOMAIN DOM_BOOL
  ADD CHECK (VALUE IN (0,1));

{$ELSE}

  CREATE DOMAIN DOM_BOOL AS SMALLINT
  DEFAULT 0 CHECK (VALUE IN (0,1));

{$ENDIF}
```

$ENDIF

Ends the conditional execution initiated by the last or directive.

Syntax

```
{$ENDIF}
```

Example

The following script creates a generator:

```
{$IFNOTEXISTS GENERATOR "GenUserID"}

   CREATE GENERATOR "GenUserID";

{$ENDIF}
```

Conditional Directives - the complete example

This example illustrates the use of conditional directives for upgrading databases. Let's assume there is an initial version of your database (version 1):

```
  CREATE TABLE FIRST_TABLE (
  ID    INTEGER NOT NULL,
  DATA  VARCHAR(100));

   CREATE PROCEDURE GETDBVER
   RETURNS (
     VER INTEGER)
   AS
   begin
     ver = 1;
     suspend;
   end;
```

The next script will upgrade a database of any version < 4 to version 4.

```
/***** Upgrade to version 2 *****/
{$IfNotExists select ver from GetDBVer where ver > 1}

ALTER TABLE FIRST_TABLE
ADD CONSTRAINT PK_FIRST_TABLE
PRIMARY KEY (ID);

ALTER PROCEDURE GETDBVER
RETURNS (
  VER INTEGER)
AS
begin
  ver = 2;
  suspend;
end;

{$endif}
```

```
/***** Upgrade to version 3 *****/
{$IfNotExists select ver from GetDBVer where ver > 2}
CREATE GENERATOR GEN_FIRST_TABLE_ID;

CREATE TRIGGER FIRST_TABLE_BI0 FOR FIRST_TABLE
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  new.id = gen_id(gen_first_table_id, 1);
end;

ALTER PROCEDURE GETDBVER
RETURNS (
    VER INTEGER)
AS
begin
  ver = 3;
  suspend;
end;
{$endif}
```

```
/***** Upgrade to version 4 *****/
{$IfNotExists select ver from GetDBVer where ver > 3}
CREATE EXCEPTION DELETION_NOT_ALLOWED 'You cannot delete records!';

CREATE TRIGGER FIRST_TABLE_BD0 FOR FIRST_TABLE
ACTIVE BEFORE DELETE POSITION 0
AS
begin
  exception deletion_not_allowed;
end;

ALTER PROCEDURE GETDBVER
RETURNS (
    VER INTEGER)
AS
begin
  ver = 4;
  suspend;
end;
{$endif}
```

DESCRIBE DOMAIN

This changes a domain description.

Syntax

```
DESCRIBE DOMAIN domain_name 'description';
```

Argument	Description
`domain_name`	Name of an existing domain.
`'description'`	Quoted string containing a domain description.

Description

`DESCRIBE DOMAIN` changes the description of an existing domain `domain_name`. When the IBExpert Script Executive executes this statement it modifies the value of the `RDB$DESCRIPTION` column in `DB$FIELDS` connected with the specified domain name.

Actually the following statement is executed:

```
UPDATE RDB$FIELDS
SET RDB$DESCRIPTION = :DESC
WHERE RDB$FIELD_NAME = 'domain_name'
```

where `DESC` parameter is filled with the description.

Example

```
DESCRIBE DOMAIN DOM_BOOL
'Boolean value:
 0 - FALSE
 1 - TRUE';
```

DESCRIBE EXCEPTION

This changes an exception's description.

Syntax

```
DESCRIBE EXCEPTION exception_name 'description';
```

Argument	Description
`exception_name`	Name of an existing exception.
`'description'`	Quoted string containing a new description of specified exception.

Description

`DESCRIBE EXCEPTION` changes the description of an existing exception `exception_name`. When the IBExpert Script Executive executes this statement it modifies the value of the `RDB$DESCRIPTION` column in `RDB$EXCEPTIONS` connected with the specified exception. Actually the following statement is executed:

```
UPDATE RDB$EXCEPTIONS
SET RDB$DESCRIPTION = :DESC
WHERE RDB$EXCEPTION_NAME = 'exception_name'
```

where the `DESC` parameter is filled with the description.

Example

```
DESCRIBE EXCEPTION MISSING_USER
'There is no such user!';
```

DESCRIBE FIELD

This changes a column description.

Syntax

```
DESCRIBE FIELD column_name TABLE table_name 'description';
```

Argument	Description
`column_name`	Name of an existing column of table `table_name`.
`table`	Name of an existing table.
`'description'`	Quoted string containing a column description.

Description

`DESCRIBE FIELD` changes the description of an existing column `column_name` of table `table_name`. When the IBExpert Script Executive executes this statement it modifies the value of the `RDB$DESCRIPTION` column in `RDB$RELATION_FIELDS` connected with the specified column and table names. Actually the following statement is executed:

```
UPDATE RDB$RELATION_FIELDS
SET RDB$DESCRIPTION = :DESC
WHERE (RDB$RELATION_NAME = 'table_name') AND
      (RDB$FIELD_NAME = 'column_name')
```

where the `DESC` parameter is filled with the description.

Example

```
DESCRIBE FIELD FULL_USER_NAME TABLE USERS
'Full user name.
 Computed, concatenation of FIRST_NAME, MIDDLE_NAME and LAST_NAME';
```

DESCRIBE FUNCTION

This changes an UDF description.

Syntax

```
DESCRIBE FUNCTION function_name 'description';
```

Argument	Description
function_name	Name of an existing user-defined function.
'description'	Quoted string containing an UDF description.

Description

DESCRIBE FUNCTION changes the description of an existing user-defined function function_name. When the IBExpert Script Executive executes this statement it modifies the value of the RDB$DESCRIPTION column in RDB$FUNCTIONS connected with the specified function. Actually the following statement is executed:

```
UPDATE RDB$FUNCTIONS
SET RDB$DESCRIPTION = :DESC
WHERE RDB$FUNCTION_NAME = 'function_name'
```

where the DESC parameter is filled with the description.

Example

```
DESCRIBE FUNCTION COMPARE_BLOBS
'Compares two blob values and returns 1
if both values are equal. In other case returns 0';
```

DESCRIBE PARAMETER

This changes a procedure parameter description.

Syntax

```
DESCRIBE PARAMETER parameter_name PROCEDURE procedure_name 'description';
```

Argument	Description
parameter_name	Name of an existing parameter of stored procedure.
procedure_name	Name of an existing stored procedure.
'description'	Quoted string containing a parameter description.

Description

DESCRIBE PARAMETER changes the description of an existing parameter parameter_name of a specified stored procedure procedure_name. When the IBExpert Script Executive executes this statement it modifies the value of the RDB$DESCRIPTION column in RDB$PROCEDURE_PARAMETERS connected with the specified parameter and procedure names. Actually the following statement is executed:

```
UPDATE RDB$PROCEDURE_PARAMETERS
SET RDB$DESCRIPTION = :DESC
WHERE (RDB$PROCEDURE_NAME = 'procedure_name') AND
      (RDB$PARAMETER_NAME = 'parameter_name')
```

where the DESC parameter is filled with the description.

Example

```
DESCRIBE PARAMETER USER_ID PROCEDURE CALC_TRAFFIC
'User ID';
```

DESCRIBE PROCEDURE

This changes a stored procedure description.

Syntax

```
DESCRIBE PROCEDURE procedure_name 'description';
```

Argument	Description
procedure_name	Name of an existing stored procedure.
'description'	Quoted string containing a procedure description.

Description

DESCRIBE PROCEDURE changes the description of an existing stored procedure procedure_name. When the IBExpert Script Executive executes this statement it modifies the value of the RDB$DESCRIPTION column in RDB$PROCEDURES connected with the specified procedure. Actually the following statement is executed:

```
UPDATE RDB$PROCEDURES
SET RDB$DESCRIPTION = :DESC
WHERE RDB$PROCEDURE_NAME = 'procedure_name'
```

where the DESC parameter is filled with the description.

Example

```
DESCRIBE PROCEDURE CALC_TRAFFIC
'Calculates the summary traffic';
```

DESCRIBE TABLE

This changes a table description

Syntax

```
DESCRIBE TABLE table_name 'description';
```

Argument	Description
table_name	Name of an existing table.
'description'	Quoted string containing a table description.

Description

DESCRIBE TABLE changes the description of an existing table table_name. When the IBExpert Script Executive executes this statement it modifies the value of the RDB$DESCRIPTION column in RDB$RELATIONS connected with the specified table. Actually following statement is executed:

```
UPDATE RDB$RELATIONS
SET RDB$DESCRIPTION = :DESC
WHERE RDB$RELATION_NAME = 'table_name'
```

where the DESC parameter is filled with the description.

Example

```
DESCRIBE TABLE CUSTOMERS
'Customers of our excellent application';
```

DESCRIBE TRIGGER

This changes a trigger description

Syntax

```
DESCRIBE TRIGGER trigger_name 'description';
```

Argument	Description
trigger_name	Name of an existing trigger.
'description'	Quoted string containing a trigger description.

Description

DESCRIBE TRIGGER changes the description of an existing trigger trigger_name. When the IBExpert Script Executive executes this statement it modifies the value of the RDB$DESCRIPTION column of RDB$TRIGGERS connected with the specified table. Actually the following statement is executed:

```
UPDATE RDB$TRIGGERS
SET RDB$DESCRIPTION = :DESC
WHERE RDB$TRIGGER_NAME = 'trigger_name'
```

where the DESC parameter is filled with the description.

Example

```
DESCRIBE TRIGGER USERS_BI
'Generates an unique identifier';
```

DESCRIBE VIEW

This changes a view description

Syntax

```
DESCRIBE VIEW view_name 'description';
```

Argument	Description

view_name	Name of an existing view.
'description'	Quoted string containing a view description.

Description

DESCRIBE VIEW changes the description of an existing view view_name. When the IBExpert Script Executive executes this statement it modifies the value of the RDB$DESCRIPTION column of RDB$RELATIONS connected with the specified view. Actually the following statement is executed:

```
UPDATE RDB$RELATIONS
SET RDB$DESCRIPTION = :DESC
WHERE RDB$RELATION_NAME = 'view_name'
```

where the DESC parameter is filled with the description.

Example

```
DESCRIBE VIEW ALL_USERS
'Just all users...:)';
```

INSERTEX (CSV file import)

This imports data from a CSV-file into a database table.

Syntax

```
INSERTEX INTO table_name [(columns_list)]
   FROM CSV file_name
   [SKIP n]
   [DELIMITER delimiter_char]
```

Argument	Description
table_name	Name of a table into which to insert data.
columns_list	List of columns into which to insert data.
file_name	Name of CSV-file from which to import data.
SKIP n	Allows the first n lines of CSV-file to be skipped while importing data.
DELIMITER delimiter_char	Allows a delimiter to be specified, which will be used for parsing data values.

If this argument isn't specified IBExpert will use a colon as a delimiter.

Description

INSERTEX imports data from a CSV-file into a database table. Values within the CSV-file must be separated with a colon CHAR or any other char. In the latter case it is necessary to specify a delimiter CHAR using the DELIMITER argument. It is also possible to specify non-print characters as a delimiter. For example, if values are separated with tab char (ASCII value $09) it may be specified as DELIMITER #9 or DELIMITER $9.

To ignore unwanted quotes use the QUOTECHAR '"' option.

If a table table_name is missing in the database, it will be created automatically. In this case the number of columns in the newly created table will be equal to the number of values in the first line of the CSV-file. Columns will be named F_1, F_2 etc. The data type of each column is VARCHAR(255).

If the columns_list isn't specified IBExpert will insert data from the very first column. Otherwise data will only be inserted into specified columns. It is possible to skip the first several lines of the CSV-file using the SKIP argument. This may be useful if the first line contains column captions or is empty.

Since IBExpert version 2005.02.12.1 it is possible to use the INSERTEX command in the SQL Editor.

Examples

Let's consider the use of INSERTEX in the following examples. Assume there is a CSV-file with the following data, delimited with a colon:

```
C:\Mydata.csv
=====================================================
ID:FIRST_NAME:LAST_NAME:SEX
1:John:Doe:M
2:Bill:Gates:M
3:Sharon:Stone:F
4:Stephen:King:M
=====================================================
```

The following INSERTEX statement creates a table PEOPLE (if it doesn't already exist) and fills it with data from C:\Mydata.csv:

```
INSERTEX INTO PEOPLE FROM CSV 'C:\Mydata.csv' DELIMITER ':';
```

The structure and contents of PEOPLE after the data import are shown below:

F_1 (VARCHAR(255))	F_2 (VARCHAR(255))	F_3 (VARCHAR(255))	F_4 (VARCHAR(255))
ID	*FIRST_NAME*	*LAST_NAME*	*SEX*
1	John	Doe	M
2	Bill	Gates	M
3	Sharon	Stone	F
4	Stephen	King	M

The following INSERTEX statement is almost identical to the one above, but here the first line of the CSV-file has been skipped:

```
INSERTEX INTO PEOPLE FROM CSV 'C:\Mydata.csv' DELIMITER ':' SKIP 1;
```

The structure and content of the PEOPLE table after import is shown below:

F_1 (VARCHAR(255))	F_2 (VARCHAR(255))	F_3 (VARCHAR(255))	F_4 (VARCHAR(255))
1	John	Doe	M
2	Bill	Gates	M
3	Sharon	Stone	F
4	Stephen	King	M

In the next example the PEOPLE table is created first, and then subsequently populated with the data from C:\Mydata.csv:

```
CREATE TABLE PEOPLE (
    ID         INTEGER NOT NULL,
    FIRST_NAME VARCHAR(30),
    LAST_NAME  VARCHAR(30),
    SEX        CHAR(1));

INSERTEX INTO PEOPLE FROM CSV 'C:\Mydata.csv' DELIMITER ':' SKIP 1;
```

Below the structure and content of the PEOPLE table after import:

ID (INTEGER)	FIRST_NAME (VARCHAR(30))	LAST_NAME (VARCHAR(30))	SEX (CHAR(1))
1	John	Doe	M
2	Bill	Gates	M
3	Sharon	Stone	F
4	Stephen	King	M

In the next example only three columns (ID, FIRST_NAME and LAST_NAME) are affected:

```
CREATE TABLE PEOPLE (
    ID         INTEGER NOT NULL,
    FIRST_NAME VARCHAR(30),
    LAST_NAME  VARCHAR(30),
    SEX        CHAR(1));

INSERTEX INTO PEOPLE (ID, FIRST_NAME, LAST_NAME)
    FROM CSV 'C:\Mydata.csv'
    DELIMITER ':' SKIP 1;
```

The structure and content of the PEOPLE table after import can be seen below:

ID (INTEGER)	FIRST_NAME (VARCHAR(30))	LAST_NAME (VARCHAR(30))	SEX (CHAR(1))
1	John	Doe	NULL
2	Bill	Gates	NULL
3	Sharon	Stone	NULL
4	Stephen	King	NULL

OUTPUT

This redirects the output of SELECT statements to a named file.

Syntax

```
OUTPUT [filename [DELIMITER delim_char]
                 [QUOTECHAR 'quote_char']
                 [TIMEFORMAT 'time_format']
                 [DATEFORMAT 'date_format']
                 [DECIMALSEPARATOR 'dec_sep']
                 [NULLS]
                 [FIELDNAMES]
                 [ASINSERT [INTO table]]]
```

Argument	Description
filename	Name of the file in which to save output.
DELIMITER delim_char	Determines a delimiter character which is used for separating field values. If the delimiter is not specified, or the empty string is specified as a delimiter, outswapping of the data will be carried out in the format with the fixed positions of fields. It is also possible to specify a delimiter character as a decimal or hexadecimal value of the character code. For example, to set the tab character (ASCII value $09) as a delimiter, simply specify DELIMITER #9 or DELIMITER $9.
QUOTECHAR 'quote_char'	Defines the character which will be used for quoting string values. If this argument is not specified or an empty string is specified, string values will not be quoted.
TIMEFORMAT 'time_format'	Defines the string which will be used for formatting the values of time fields and the time slice of datetime values. If the argument is not defined, time values will be unloaded in the native InterBase format (for example, 17:15:45).
DATEFORMAT 'date_format'	Defines the string which will be used for formatting values of date fields and the date part of datetime values. If the argument is not defined, date values will be unloaded in the native InterBase format (for example, 17-FEB-2001).
DECIMALSEPARATOR 'dec_sep'	Defines the decimal separator which is used when outswapping the data. If this argument is not defined, the system decimal separator is used.
NULLS	Defines how NULL values will be output. If the argument is not specified, NULLS are output as an empty string. Otherwise NULLS will be unloaded as the string <null>.
FIELDNAMES	If this argument is specified, the first line in the resulting file will be a line with names of SELECT columns.
ASINSERT	This argument allows data to be unloaded as a set of INSERT operators, i.e. to get a usual SQL script.
INTO table	It is used together with ASINSERT for redefining the name of the table in INSERT operators. If the argument is not given, the name of the first table in the record set will be used.
AsUpdateOrInsert	Produces a script containing UPDATE OR INSERT statement. Added in IBExpert version 2008.02.19.

Description

The OUTPUT operator is intended for redirecting the output of SELECT statements in an external file. With the help of the given operator it is possible to export the data easily into a file with separators or with a fixed column position. OUTPUT without parameters closes the file which was opened with the previous OUTPUT command, and resets all export customizations to default.

If ASINSERT is not specified, blob fields are ignored when outswapping the data. Using ASINSERT even blob values are exported, i.e. an additional file with the extension .lob is created, in which all blob fields are stored.

While outputting into SQL script (ASINSERT is specified) DELIMITER, QUOTECHAR, NULLS and FIELDNAMES arguments are ignored.

Examples

The following script creates a MyData.txt file in the current directory and outputs the data of the SELECT into it, with a fixed column position format. If MyData.txt file already exists in the current directory, the data will be appended to it.

```
OUTPUT MyData.txt;
SELECT * FROM MY_TABLE;
OUTPUT;
```

In the next example the data will be exported in the comma-separated values (CSV) format:

```
OUTPUT 'C:\MyData\MyData.csv' DELIMITER ';'
                              FIELDNAMES
                              QUOTECHAR '"'
                              DECIMALSEPARATOR '.';
SELECT * FROM MY_TABLE;
OUTPUT;
```

In the following script the data will be exported into SQL script as a set of INSERT operators:

```
OUTPUT 'C:\MyScripts\Data.sql' ASINSERT INTO "MyTable";
SELECT * FROM MY_TABLE;
OUTPUT;
```

The next example illustrates usage of the OUTPUT statement together with SHELL.

```
/* First create a folder C:\MyData*/
SHELL MKDIR C:\MyData;

/* Try to delete mydata.csv */
SHELL DEL C:\MyData\mydata.csv;

/* Redirect output of SELECTs into mydata.csv */
OUTPUT C:\MyData\mydata.csv DELIMITER ';'
                            DATEFORMAT 'MMMM-dd-yyyy'
                            TIMEFORMAT 'hh:nn:ss.zzz'
                            QUOTECHAR '"';

SELECT * FROM MY_TABLE;

/* Close C:\MyData/mydata.csv */
OUTPUT;

/* Try to open just created CSV-file with Windows Notepad */
SHELL notepad.exe C:\MyData\mydata.csv NOWAIT;
```

```
/* Try to open C:\MyData\mydata.csv with the application
   associated with CSV files */
SHELL C:\MyData\mydata.csv NOWAIT;
```

Example using the `AsUpdateOrInsert` option:

```
OUTPUT 'C:\MyScripts\data.sql' ASUPDATEORINSERT;
SELECT * FROM MYTABLE ORDER BY ID;
OUTPUT;
COMMIT;
```

New in IBExpert version 2.5.0.61:

1. The `NOFIELDNAMES` option is obsolete now. This means that there will be no column captions in the output file by default. If you wish to include column captions use `FIELDNAMES` option.
2. Added possibility to customize delimiter char for `INSERTEX` command (`DELIMITER` option). If the `DELIMITER` option is missing a comma will be used as the delimiter char.

New in IBExpert version 2005.03.12:

Extended syntax of `OUTPUT` command:

1.
```
output 'E:\data.sql'
as insert into mytable commit after 1000;
select * from IBE$$TEST_DATA where F_INTEGER < 3000;
output;
```

2.
```
output 'E:\data.sql'
as reinsert into mytable
commit after 2000;
select * from IBE$$TEST_DATA where F_INTEGER < 3000;
output;
```

3.
```
output 'E:\data.sql'
as execute procedure myproc;
select * from IBE$$TEST_DATA where F_INTEGER < 3000;
output;
```

`ASINSERT` option is available for compatibility.

RECONNECT

`RECONNECT` closes the current connection and creates a new one with the same parameters (database, user name, password etc.).

Syntax

```
RECONNECT;
```

REINSERT

IBExpert has introduced the new `REINSERT` statement. Directly following an `INSERT` it is possible to perform further `INSERT`s with new contents.

SET BLOBFILE

IBExpert uses an original mechanism to extract values of blob fields into a script. This allows you to store the entire database (metadata and data) into script files and execute these scripts with IBExpert. A small example illustrates the method used to extract blob values.

For example, your database has a table named `COMMENTS`:

```
CREATE TABLE COMMENTS (
   COMMENT_ID INTEGER NOT NULL PRIMARY KEY,
   COMMENT_TEXT BLOB SUBTYPE TEXT);
```

This table has three records:

COMMENT_ID	COMMENT_TEXT
1	First comment
2	NULL
3	Another comment

If the *Extract BLOBs* option is not checked, you will receive the following script:

```
CREATE TABLE COMMENTS (
   COMMENT_ID INTEGER NOT NULL PRIMARY KEY,
   COMMENT_TEXT BLOB SUBTYPE TEXT);

INSERT INTO COMMENTS (COMMENT_ID) VALUES (1);
INSERT INTO COMMENTS (COMMENT_ID) VALUES (2);
INSERT INTO COMMENTS (COMMENT_ID) VALUES (3);
```

... and, of course, you will lose your comments if you restore your database from this script.

But if the *Extract BLOBs* option is checked IBExpert will generate quite a different script:

```
SET BLOBFILE 'C:\MY_SCRIPTS\RESULT.LOB';

CREATE TABLE COMMENTS (
   COMMENT_ID INTEGER NOT NULL PRIMARY KEY,
   COMMENT_TEXT BLOB SUBTYPE TEXT);

INSERT INTO COMMENTS (COMMENT_ID, COMMENT_TEXT) VALUES (1, h0000000_0000000D);
INSERT INTO COMMENTS (COMMENT_ID, COMMENT_TEXT) VALUES (2, NULL);
INSERT INTO COMMENTS (COMMENT_ID, COMMENT_TEXT) VALUES (3, h000000D_0000000F);
```

Also IBExpert generates a special file with the extension `.lob` where blob values are stored. In the current example `result.lob` will be 28 bytes long and its contents will be the first `commentAnother` comment.

`SET BLOBFILE` is a special extension of script language that allows IBExpert's Script Executive to execute scripts containing references to blob field values.

SET CLIENTLIB

This defines the client library to be used while executing a script.

Syntax

```
SET CLIENTLIB file_name;
```

Argument	Description
`file_name`	Client library file name.

Description

`SET CLIENTLIB` defines client library which will be used while executing a script. The default client library is `gds32.dll`.

Example

```
SET CLIENTLIB 'C:\Program Files\Firebird\Bin\fbclient.dll';
```

SET PARAMFILE

`PARAM` file is an `ini`-file with `param` values.

For example, if your script contains some parameterized `INSERT/UPDATE/DELETE` statements you can define parameter values in an external file (params file):

```
param1=12-FEB-2003
param2=John Doe
param3=35
...
```

When IBEScript finds a query with parameters it looks for the values of these parameters in the specified `params` file.

SET TRPARAMS

The `SET TRPARAMS` command was implemented in IBExpert version 2007.07.12. It allows you to specify your own parameters of the script transaction instead of default ones.

Syntax

```
SET TRPARAMS '<params>';
where <params> is a list of transaction parameters separated by commas or
spaces.
     Example:
     SET TRPARAMS 'isc_tpb_concurrency, isc_tpb_nowait';
```

Note: If the current transaction is active `SET TRPARAMS` will commit it and, following that, change the transaction parameters.

SHELL

This allows execution of an operating system command.

Syntax

```
SHELL os_command [NOWAIT];
```

Argument	Description
os_command	An operating system command.
NOWAIT	Optional argument. If specified, execution of a script will be continued right after creation of the process executing the command of operating system, not waiting its completion.

Description

The SHELL operator tries to execute the command os_command. If NOWAIT is not specified, the further execution of a script stops before completion of the process created by SHELL operator. Otherwise script execution will be continued immediately after beginning the execution of the command os_command.

Examples

The following script tries to create a folder MyFolder in the current directory:

```
SHELL mkdir MyFolder;
```

The following example shows the use of the SHELL command to start Notepad.exe and the loading of C:\MyTexts\Shedule.txt file in it. It is necessary to use NOWAIT here, otherwise it is not possible to execute the script further, and it will be impossible to resume work in IBExpert until the Notepad is closed.

```
SHELL "notepad.exe C:\MyTexts\Shedule.txt" NOWAIT;
```

The next example illustrates the use of the SHELL statement together with OUTPUT.

```
/* First create a folder C:\MyData*/
SHELL MKDIR C:\MyData;

/* Try to delete mydata.csv */
/>SHELL DEL C:\MyData\mydata.csv;

/* Redirect output of SELECTs into mydata.csv */
OUTPUT C:\MyData\mydata.csv DELIMITER ';'
                            DATEFORMAT 'MMMM-dd-yyyy'
                            TIMEFORMAT 'hh:nn:ss.zzz'
                            QUOTECHAR '"';


SELECT * FROM MY_TABLE;

/* Close C:\MyData\mydata.csv */
OUTPUT;

/* Try to open just created CSV-file with Windows Notepad */
SHELL notepad.exe C:\MyData\mydata.csv NOWAIT;

/* Try to open C:\MyData\mydata.csv with the application
   associated with CSV files */
SHELL C:\MyData\mydata.csv NOWAIT;
```

See also:
DB Registration Info / Log Files / Script Executive
Extract Metadata
IBEBLOCK (EXECUTE IBEBLOCK)
IBEScript?
SQL Monitor
Stored Procedure
Trigger

IBEScript.exe

IBEScript.exe can be used to execute any valid IBExpert script in batch files. For example a scheduled import or export job can be started without the need of any user input.

IBEScript.exe can also be used to encrypt script files, so that they are unreadable for the user, but executable together with IBEScript.exe.

Since IBExpert version 2007.09.25 it is possible to work with scripts larger than 2 GB. The newest version of IBEScript.exe was released with IBExpert version 2008.05.03.

IBEScript.dll

When you want to integrate IBExpert's scripts in your own application, you can use `IBEScript.dll`. A simple example for Delphi can be found in IBExpert's `IBEScriptDll` subdirectory. It can also be used from all other programming languages that can handle DLL calls. The newest version of `IBEScript.dll` was released with IBExpert version 2008.05.03.

`IBEScript.dll` exports following functions:

- `ExecScriptFile` - executes script from file.
- `ExecScriptText` - executes script from string buffer.
- `Connect` - connects to the database if there is no `CONNECT` statement in the script.

For examples of usage of the `ExecScriptFile` and `ExecScriptText` please view the demo application found in the `IBExpert/IBEScriptDll/DemoApp` directory.

The following is an example using the `Connect` function:

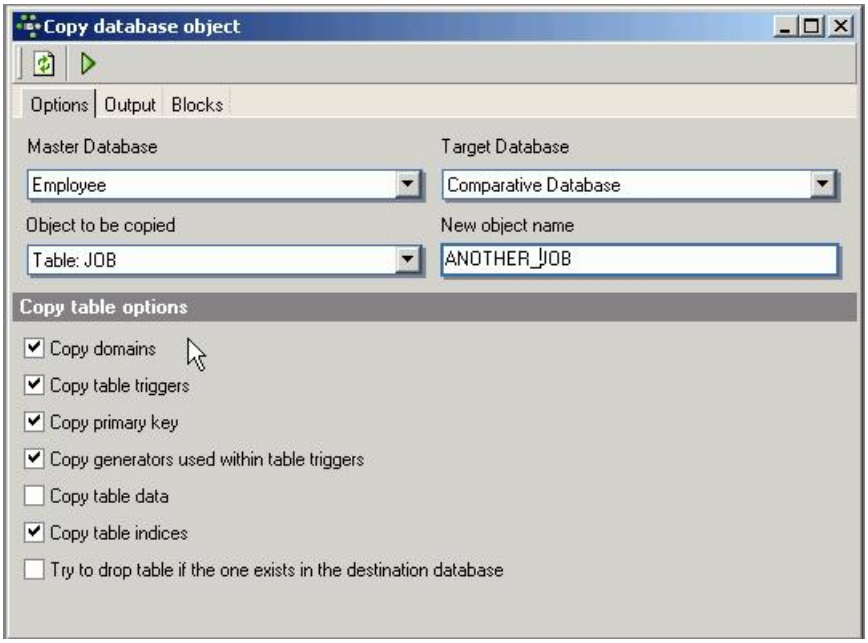
```
procedure TForm1.Button2Click(Sender: TObject);
var
  Hndl : THandle;
  ESP : TExecuteScriptProc;
  CP : TConnectDBProc;
  s : string;
  Res : integer;
begin
  ErrCount := 0;
  StmtCount := 0;
  mLog.Lines.Clear;
  s := mScript.Text;
  if Trim(s) = ´´ then
  begin
    ShowMessage('Nothing to do!');
    Exit;
  end;
  try
    Hndl := LoadLibrary(PChar('IBEScript.dll'));
    if (Hndl > HINSTANCE_ERROR) then
    begin
      ESP := GetProcAddress(Hndl, 'ExecScriptText');
      CP := GetProcAddress(Hndl, 'Connect');
      if (@ESP <> nil) and (@CP <> nil) then
      begin
        Pages.ActivePage := tsOutput;
        Res := CP(PChar('db_name=localhost:c:\empty.fdb; password=masterkey; user_name=SYSDBA;' +
                        'lc_ctype=win1251; sql_role_name=ADMIN; sql_dialect=3;' +
                        'clientlib="c:\program files\firebird\bin\fbclient.dll"'), @CEH);
        if Res = 0 then
          ESP(PChar(s), @HandleError, @BeforeExec, @AfterExec);
      end;
    end;
  finally
    if Hndl > HINSTANCE_ERROR then
      FreeLibrary(Hndl);
  end;
end;
```

Copy database object

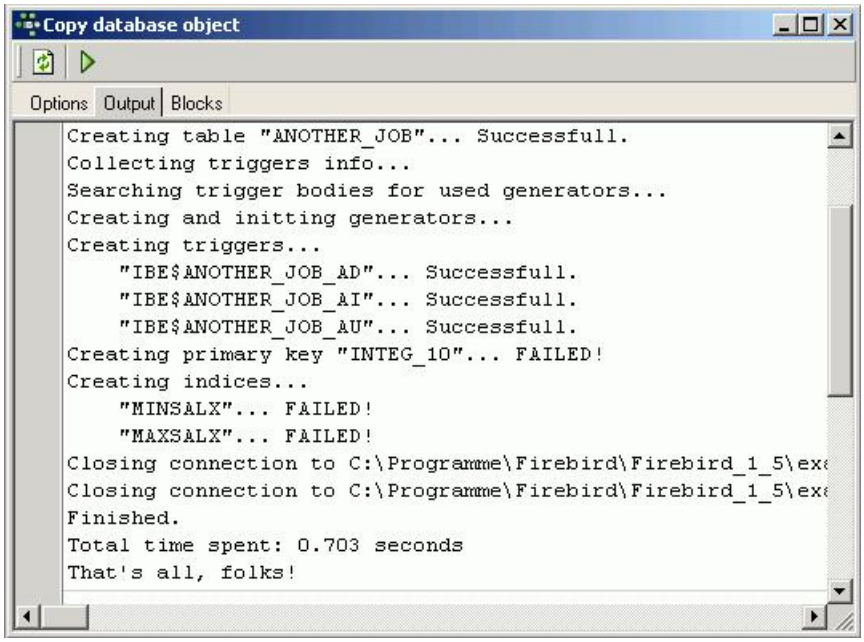
Copy Database Object was implemented in IBExpert version 2007.05.03. This feature is available as a new menu item in the IBExpert Tools menu and also in the Database Explorer context-sensitive menu: *Copy object ...*.

Simply select the database (*Master Database*) and database object (*Object to be copied*) you wish to copy, then specify the database where this object is to be copied to (*Target Database*). The original object name automatically appears in the *Newobject name* field; this can of course be altered if wished.

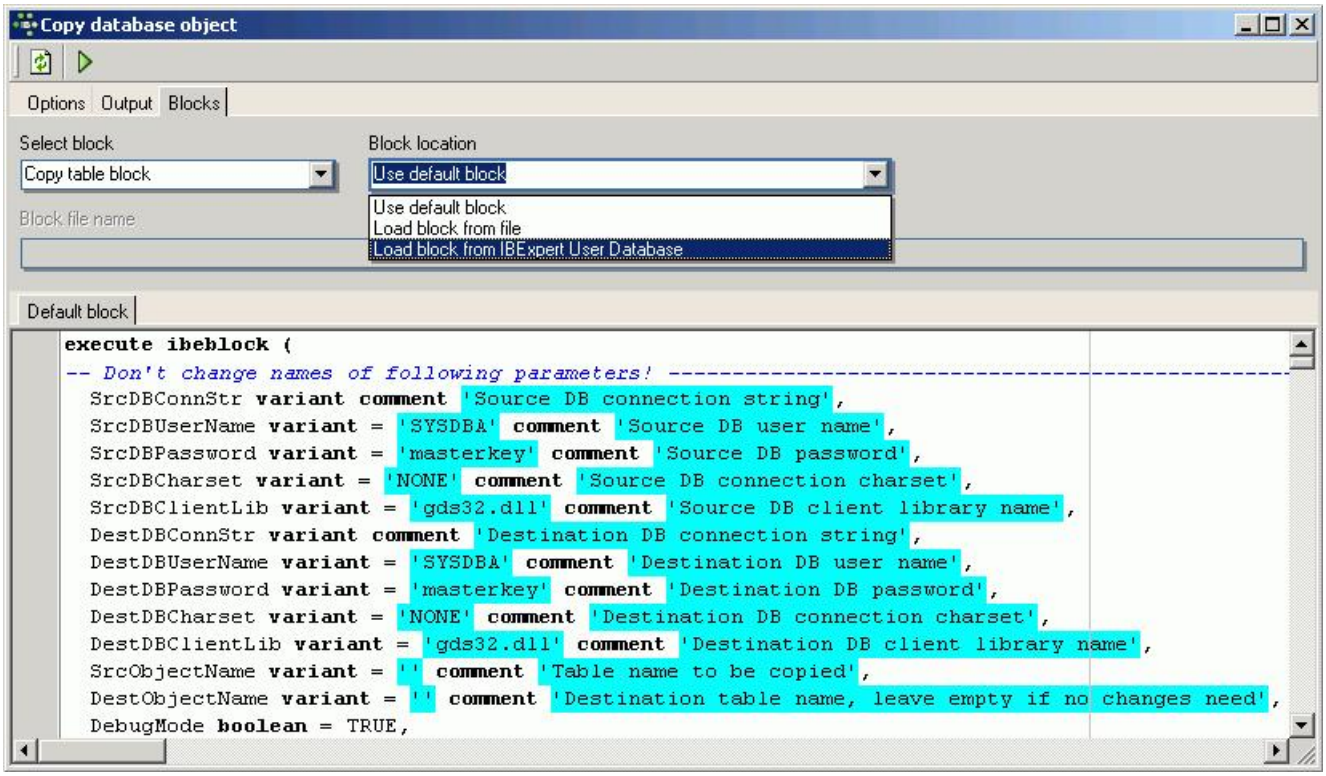
Depending upon the object selected, a number of checkbox *Copy options* are offered, including options for exactly which contents should be copied, and how IBExpert should proceed should the object already exist.



Start the copy process by clicking the green arrow icon or using [F9]. The *Output* script appears:



On the *Blocks* page, the default IBEBlock is displayed. You can of course load your own IBEBlock from file or from the IBExpert User Database. Further options include *Select block*, allowing the various database object scripts to be copied.



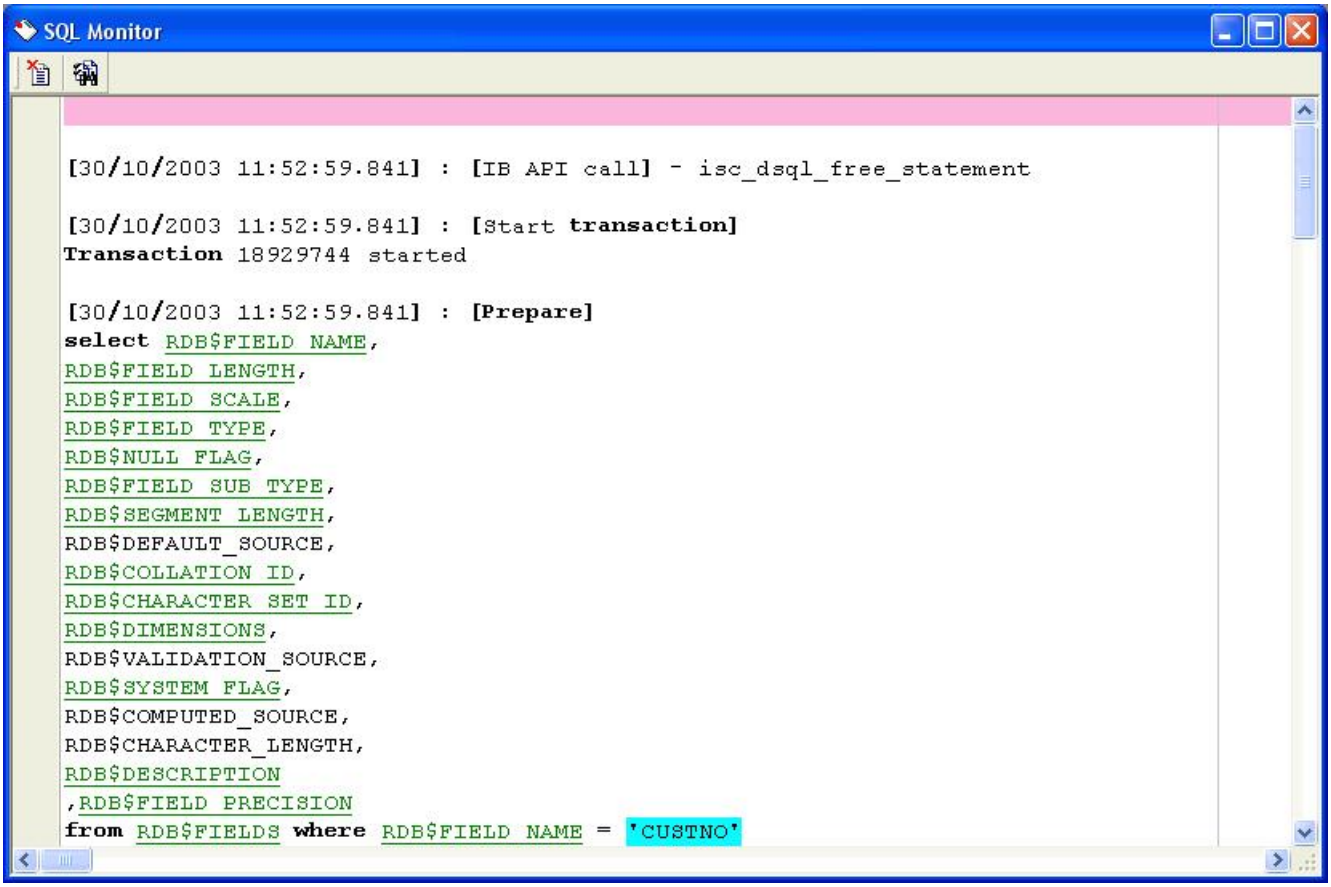
The *Copy Database Object* feature is based on IBEBlock functionality and is therefore is fully customizable.

See also:
IBEBlock

SQLMonitor

The SQL Monitor can be started in the IBExpert Tools menu, using the respective icon in the Tools toolbar or using the key combination [Ctrl + M].

The SQL Monitor can be used if a detailed protocol is required. Once opened, it logs everything performed in IBExpert, allowing the user to view all actions as SQL code.



It provides detailed background information for those wishing to learn and analyze the way IBExpert works. It is also an ideal tool for analyzing certain problems or error messages that can otherwise not easily be solved.

The SQL Monitor always includes a timestamp, regardless of whether this option is checked in the Database Registration Info / Log Files or not.

The SQL code cannot be edited directly; it can however be copied to clipboard, saved to file or printed, using the right-click SQL Editor menu. Further operations, such as Incremental Search, are explained under SQL Editor Menu.

Please note that the SQL Monitor is not able to log all SQL calls to the database server; it only logs IBExpert calls.

Please refer to SQL Monitor Options for details of customization.

See also:

SQL Monitor Options

The *Monitor Options* icon:

allows the user to specify exactly what should be monitored or not monitored:

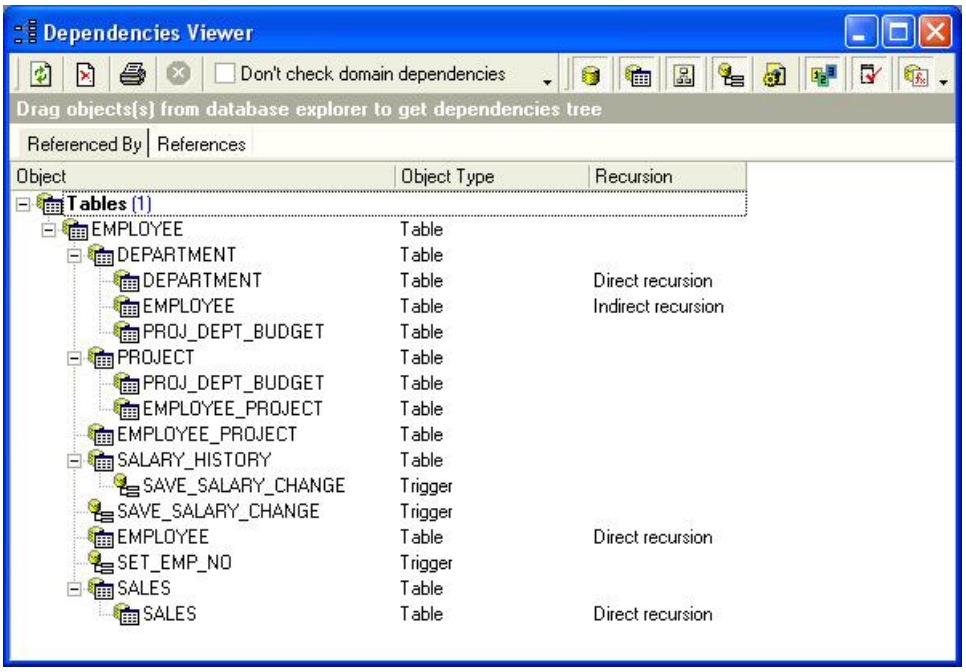
- **Connect/Disconnect:** whether the database connection should also be protocolled.
- **Prepare / Execute / Fetch:** which phases of the SQL queries should be monitored.
- **Transactions:** whether each individual transaction should be monitored.
- **Services:** monitoring of the individual commands at API level
- **API calls:** direct InterBase/Firebird calls (ICE files). This option may really only be of interest to hardcore C programmers!

See also:

Dependencies Viewer

The IBExpert Dependencies Viewer is an ideal tool for ascertaining any dependencies upon an object or an object's dependency upon other objects - particularly important before deleting objects!

It can be found in the IBExpert Tools menu.

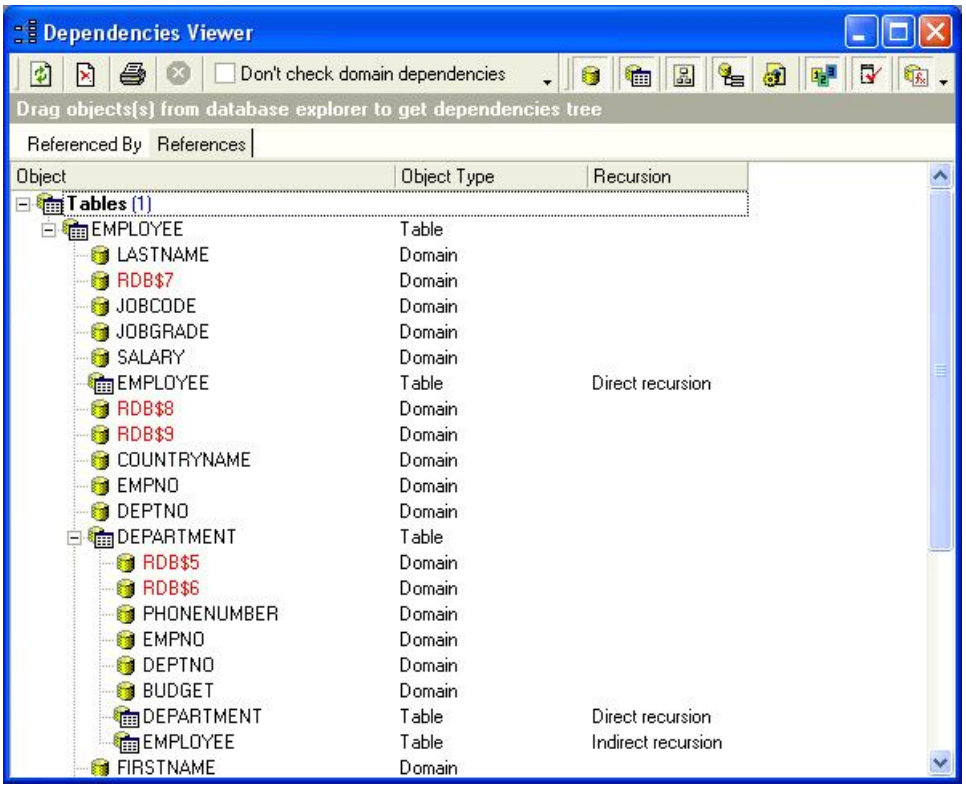


Database objects can be simply moved from the DB Explorer into the Viewer using drag 'n' drop.

IBExpert version 2006.12.11 introduced the possibility to run the SP/Trigger/View Analyzer for selected objects, using the dependencies tree context-sensitive menu item, Database Analyzer.

The *Referenced By* page displays which objects reference the selected object, i.e. the higher-ranking objects (in the above illustration `EMPLOYEE`) are referenced by the subordinate objects (in the above example: `DEPARTMENT`, `PROJECT`, `EMPLOYEE_PROJECT`, `SALARY_HISTORY`, `EMPLOYEE` (references itself = direct recursion), `SET_EMP_NO` and `SALES`).

The *References page*:



shows which objects are used by the selected object. In the above example, this includes, among others, the `EMPLOYEE` and `DEPARTMENT` tables.

It is possible to specify whether domains should be displayed or not, by simply checking the *Don't Show Domains* box in the toolbar. As it is possible for domains to reference other domains, and each table field is based either on a user-defined or system domain, this may slow work with the Dependencies Viewer if it is not checked.

Further object display criteria are offered by the icons in the toolbar (please refer to | Dependencies Viewer toolbar for details).

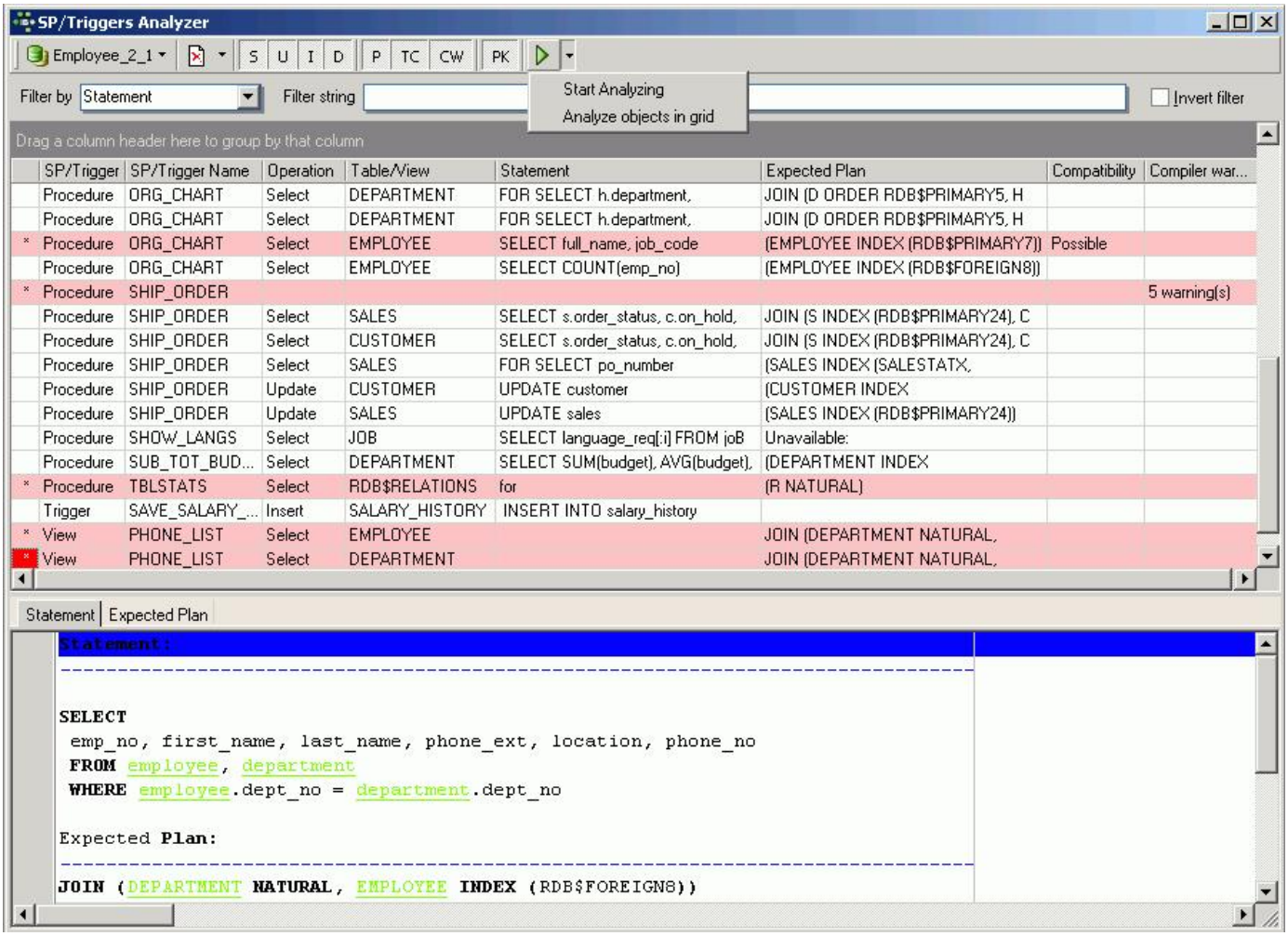
- **Direct recursion** indicates that an object references itself.
- **Indirect recursion** indicates that an object references itself indirectly via one or more other objects, for example EMPLOYEE references itself indirectly via DEPARTMENT (each employee belongs to a department; each department has a manager, who is an employee).

Double-clicking on any of the objects in the Viewer opens the respective object dialog.

SP/Triggers/Views Analyzer

The Stored Procedure/Trigger/Views Analyzer can be found in the IBExpert Tools menu. (This feature is unfortunately not included in the Personal Edition.)

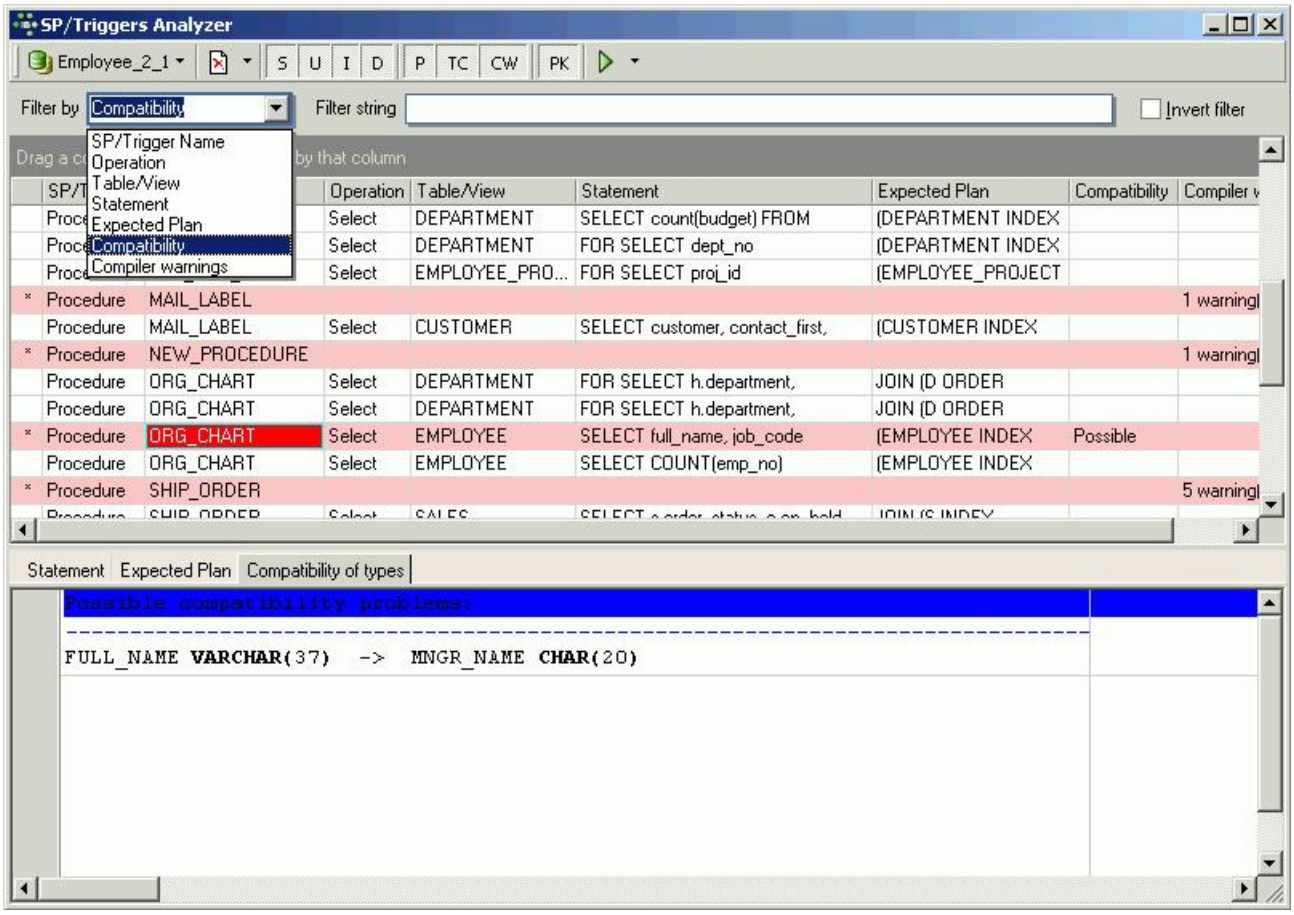
It allows the user to view and analyze how the database performs individual operations/statements in a stored procedure, trigger or view. For example, certain indices perhaps may not be used by the database server as the statistics are too high; this can be solved simply by using the IBExpert Database menu item Recompute selectivity of all indices to update the selectivity. Or when backing up an older InterBase version and restoring to a new InterBase/Firebird version, the procedures and triggers appear not to work as it is often necessary to first Recompile all stored procedures and triggers (also found in the IBExpert Database menu).



The database to be analyzed can be selected from the pull-down list of *all connected databases* (the first toolbar item). By clicking on the *Start Analyzing* icon, it loads all stored procedures and triggers for the active database.

They are all automatically analyzed, i.e. each procedure/trigger is split up into its individual statements (the first SQL row is displayed in the *Statement* column; the full code is displayed in the lower *Statement* window). All statements with any sort of problems (no index, compiler warning etc.) are highlighted, and need looking at more closely.

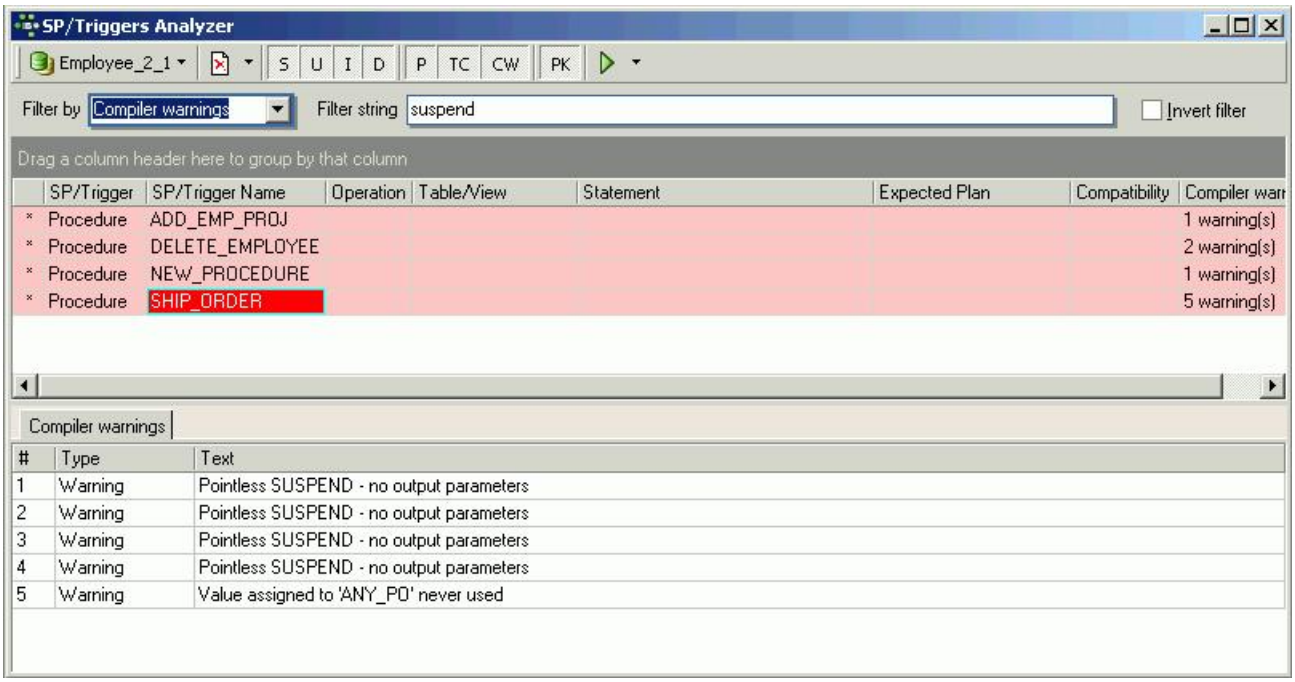
The indices used for each operation are displayed in the right-hand *Expected Plan* column; details are displayed in a tree form in the lower *Expected Plan* window. Possible compatibility problems are indicated in the *Compatibility* column with details in the *Compatibility of Types* window below:



The last column displays compiler warnings, again with details in the lower window (see illustration below).

The user can specify exactly what he would like to analyze by deactivating or activating the toolbar icons:

- **S** All `SELECT` statments are selected, analyzed and displayed.
- **U** All `UPDATE` statments are selected, analyzed and displayed.
- **I** All `INSERT` statments are selected, analyzed and displayed.
- **D** All `DELETE` statments are selected, analyzed and displayed.
- **P** Analysis of plans and indices.
- **TC** Analysis of the compatibility of types of return values and variables for `SELECT...INTO` and `OR SELECT...INTO` statements.
- **CW** Displays all compiler warnings.
- **PK** Checks primary keys.



The analysis results can be filtered by the criteria listed in the drop-down *Filter by* list:

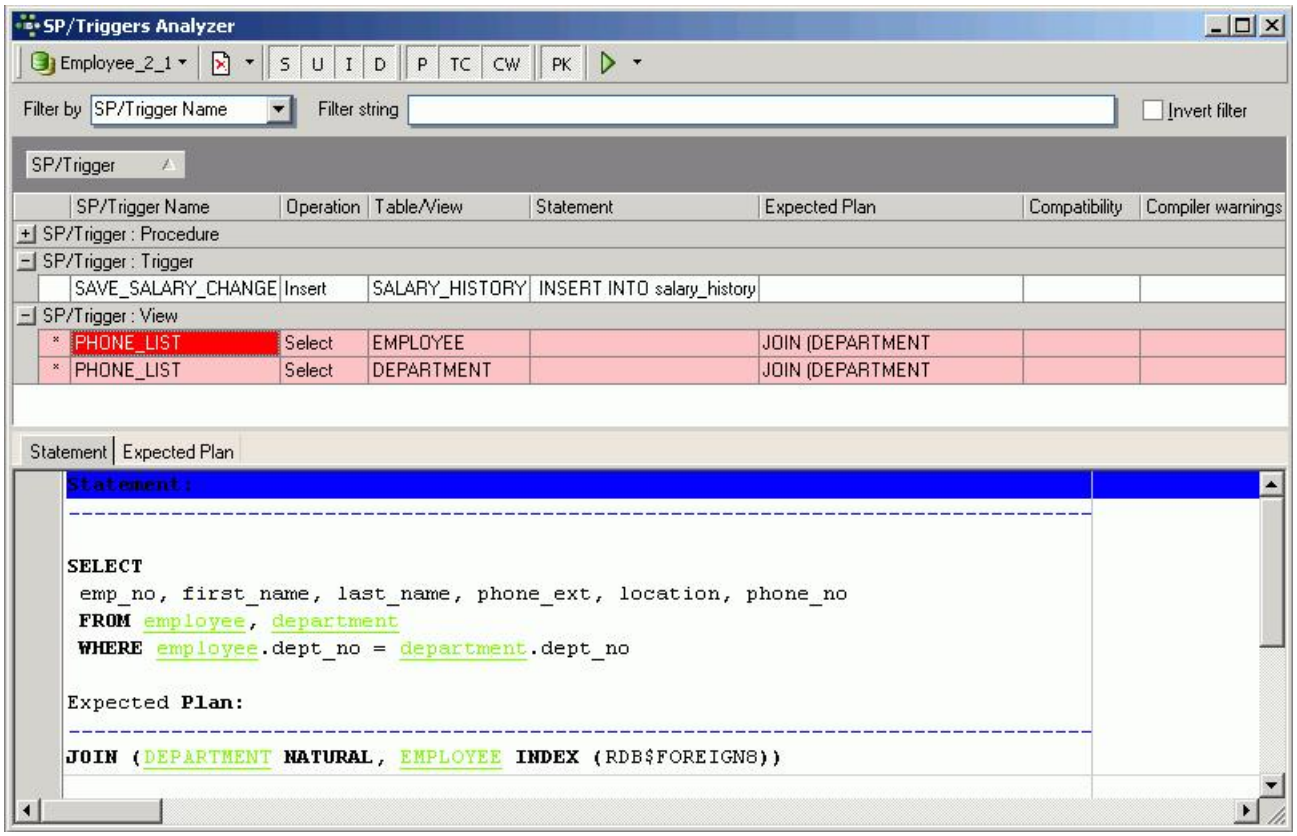
- SP/Trigger name
- Operation
- Table View
- Statement
- Expected Plan
- Compatibility
- Compiler warnings

and supplemented by the user-specified filter string to the right, to search for specific objects, operations or problems. This filter can even be inverted (check box option on the right).

As with all IBExpert grids the contents can be sorted by clicking on the desired column header (e.g. sort according to *Name*, *Table/View*, *statement* etc.). By clicking on the left-hand column header (the unnamed column to the left of the *SP/Trigger* column), the red highlighted objects (i.e. those with any sorts of problem that need looking at more closely) are grouped together.

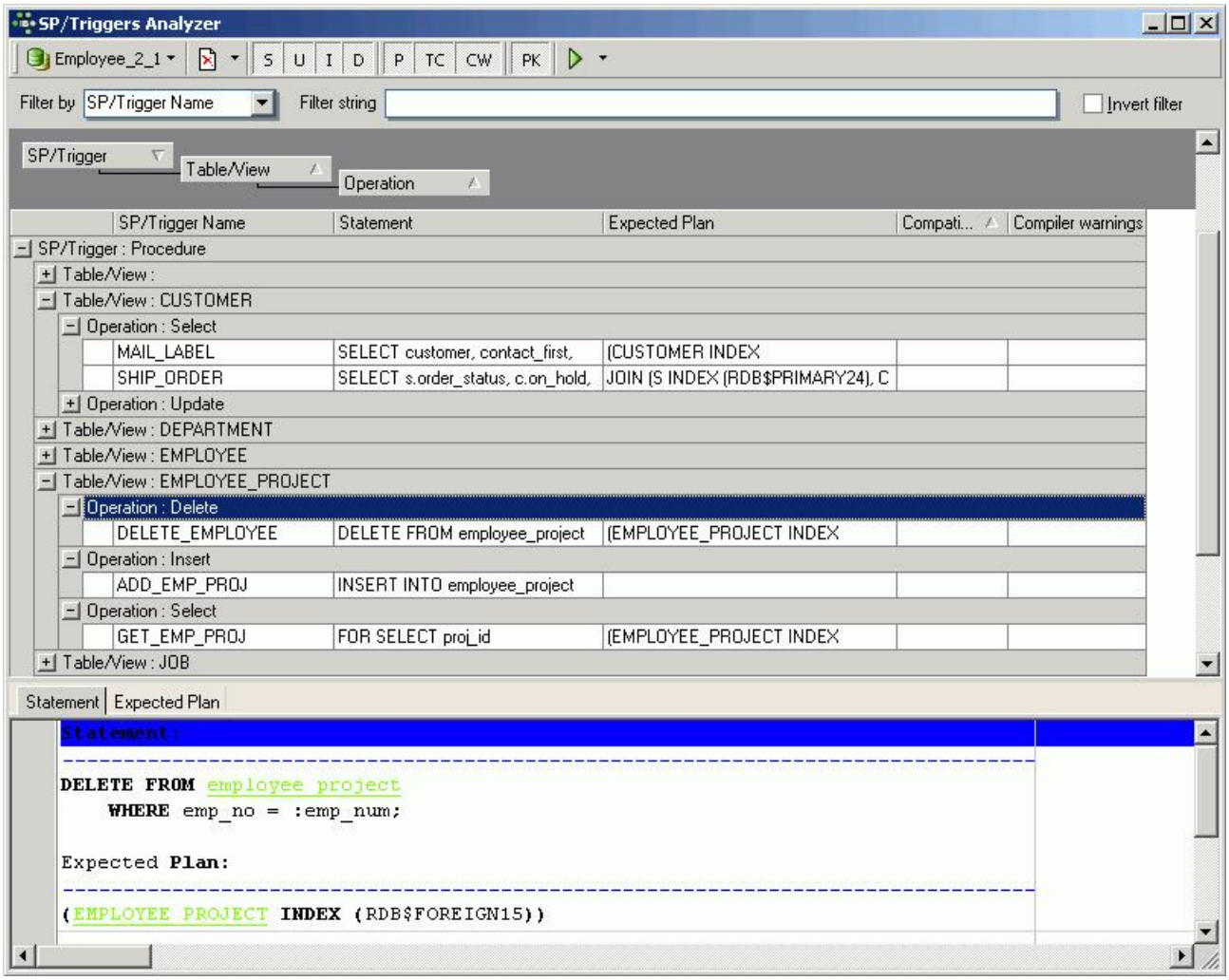
The Procedure, Trigger, Table or View editors can be quickly started by double-clicking on a selected field, allowing the user for example, to quickly and easily insert an index.

Column headers can also be dragged to the gray area below the toolbar, to group by the column selected:



The above illustration displays all stored procedures and triggers grouped by the procedure or trigger name. By clicking '+' or '-', or double-clicking on the list name, the individual operations can be easily blended in or out.

It is also possible to group by more than one criteria:



The lower window displays the SQL text for a selected operation on the *Statement* page, in the lower half of the window. The statements can easily be copied and inserted into a text editor or the IBExpert SQL Editor, using the context-sensitive right-click menu (please refer to SQL Editor Menu for further details).

In case it is of interest, the SP/Triggers/Views Analyzer was realized using the Developer Express component.

See also:
Debug Procedure or Trigger

Database Comparer

The IBExpert Database Comparer can be found in the IBExpert Tools menu. Unfortunately it is not included in the Personal Edition.

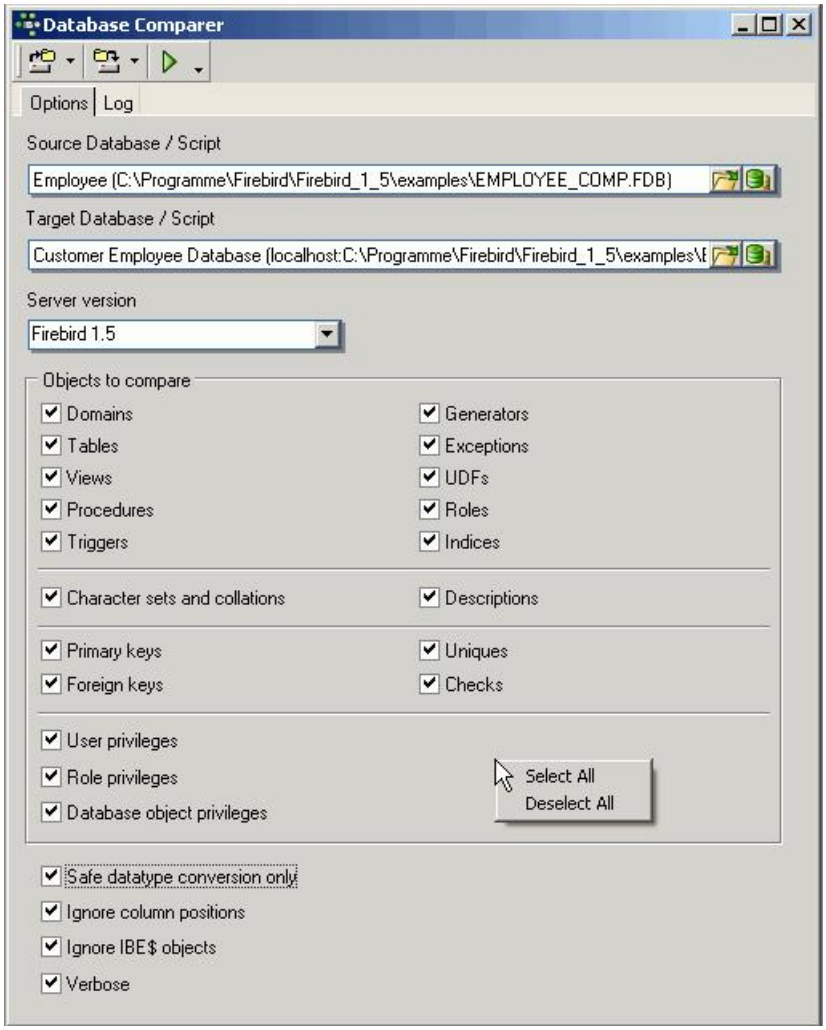
It allows developers to compare database versions or database SQL scripts. This is particularly useful, for example, before installing an updated client application, which contains new tables, procedures, exceptions, etc. etc., as it is possible to compare the databases, and - by analyzing the resulting script, view both the changes to the software, as well as those data changes made by the client, erasing any irrelevant alterations, and applying those which are relevant, by executing the script.

Options page

On the *Options* page, first select the *Source (Master/Reference) Database* or *SQL script*, by clicking the icons to the right of the path/file input area, to specify drive, path and database name. This is the reference database, to which the second database is to be compared. Then select the *Target (Comparative) Database* or script, i.e. the database which needs to be assessed and altered in order to conform with the reference database. Instead of searching for the path and directory of the databases you wish to compare, you can simply drag 'n' drop both databases from the DB Explorer into the respective fields in the Database Comparer dialog.

Scripts can also be selected and compared (since IBExpert version 2004.04.01.1). And it is possible to store into or load from an external file (using the toolbar icons at the top of the dialog), and use this together with IBECompare (IBExpert command-line tool). Since IBExpert version 2006.10.14 when settings are saved into an `INI` file, IBExpert also saves the server version.

The *Server version* (introduced in IBExpert version 2005.12.04) offers a drop-down list to allow specification of the Firebird or InterBase server version and therefore which syntax should be used while comparing the two selected databases.



There are a number of options, which can be checked if they should be included in the comparison. Since IBExpert version 2007.07.18 all the options can be selected or deselected simply and quickly using the right-click context-sensitve menu. These options include:

- **Objects to compare**
  - Domains
  - Tables
  - Views
  - Procedures

    - Triggers
    - Generators
    - Exceptions
    - UDFs
    - Roles
    - Indices

- Character sets and collations
- Descriptions

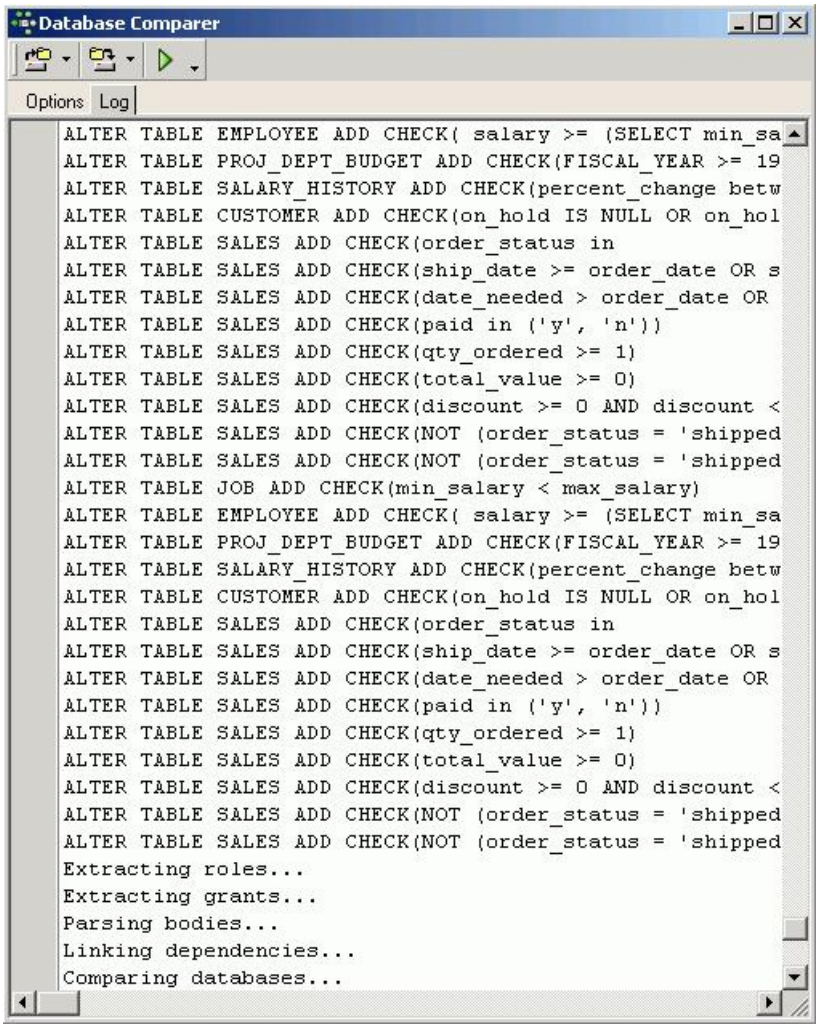
- **Keys and constraints**
    - primary keys
    - foreign keys
    - uniques
    - checks

- **Privileges**
    - User privileges
    - Role privileges
    - Database object privileges

- **Miscellaneous**
    - Safe datatype conversion only: introduced in IBExpert version 2006.12.1. If this option is enabled, only safe datatype conversion `ALTER <column> TYPE <new_type>` will be performed. Otherwise system tables are updated directly. InterBase 2007 is also supported.
    - Ignore column positions
    - Ignore IBE$ objects
    - Verbose: this displays each step that IBExpert performs and when, allowing a detailed comparison.

Array fields are also supported since IBExpert version 2005.12.04.

After selecting all features to be (or not to be) compared, click the *Compare* icon to start the comparison:



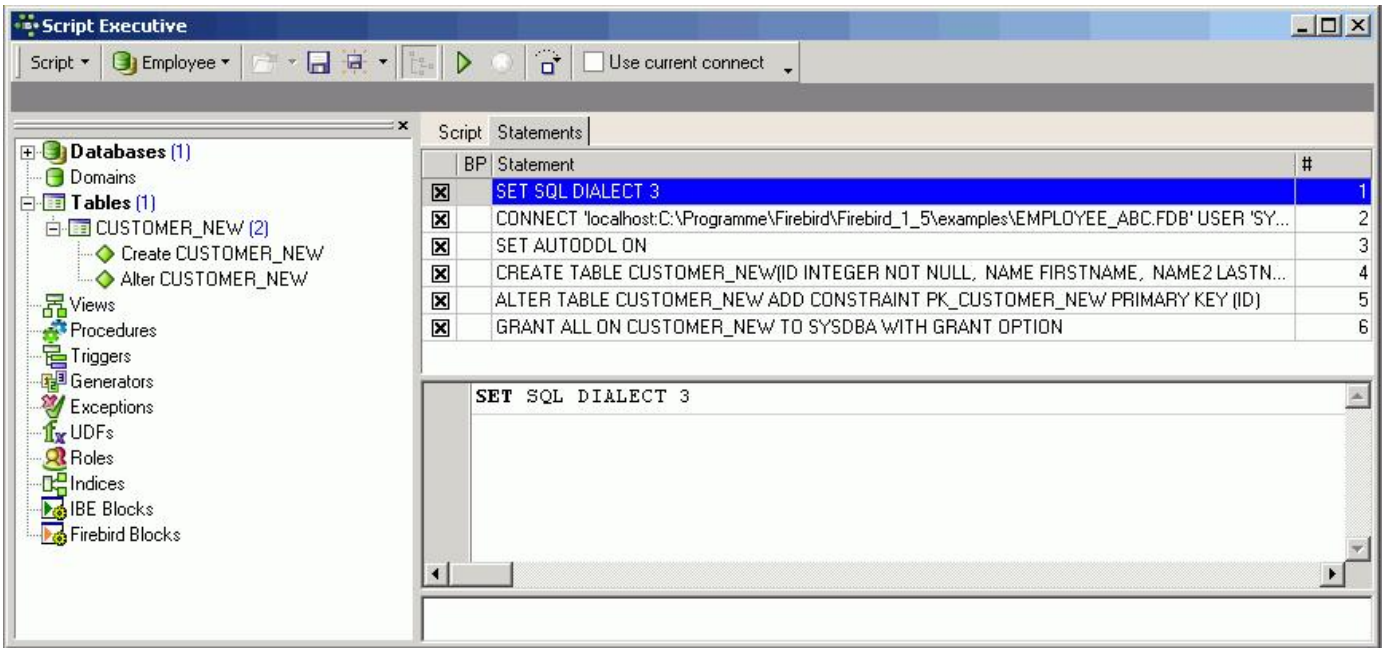
Log page

The *Log* page logs the comparison, which can be halted and restarted at any time by using the *Stop* and *Compare* icons.

The results are automatically loaded in the Script Executive. Here it is easy to see which operations need to be performed, in order to make the comparative database identical to the reference database.

Since IBExpert version 2005.08.02 there is added support for new Firebird 2.0 features such as `SELECT ... FROM (SELECT ...)`, `IS DISTINCT FROM` etc.

Statements page



It is simple to unselect or select individual statements using point and click. Please refer to Script Executive for further details. By executing all SQL statements the comparative database becomes identical to the master database.

Please note that certain alterations may cause serious problems with your database, due to restrictions and limitations in Firebird/InterBase. For example, changing a datatype from `CHAR` to `INT`. Also: Firebird seems to have problems with certain dependencies. For example, when dropping a view with dependent procedures, the Firebird server removes records from `RDB$DEPENDENCIES` and doesn't recreate them when the view is recreated.

We at IBExpert are aiming to generate comments for all such items that cannot be modified. Please mail us (documentation@ibexpert.com) if you incur problems which are not yet reported by IBExpert.

Support was introduced for Firebird 2.1 in IBExpert version 2007.12.01 .

See also:
Table Data Comparer

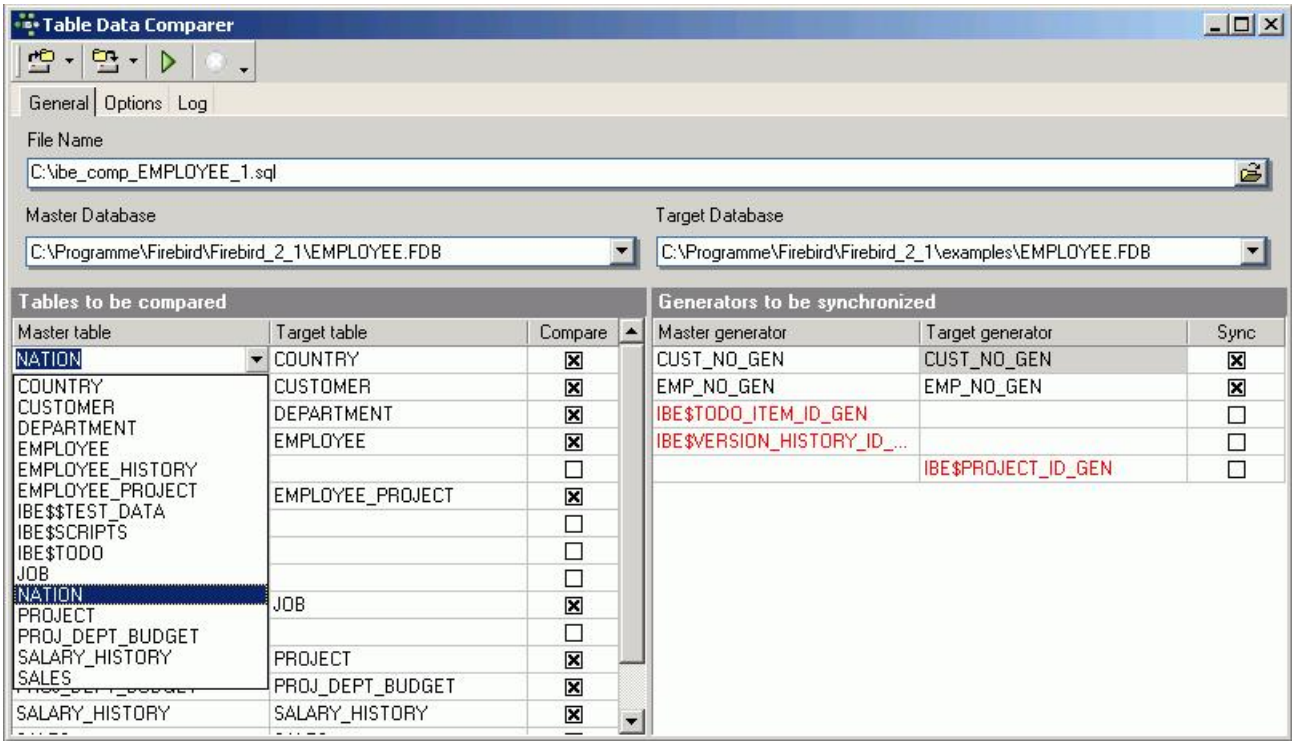
Table Data Comparer

The Table Data Comparer can be found in the IBExpert Tools menu. It allows you to compare data of two tables in different databases and obtain a script detailing all discrepancies which includes corresponding INSERT, UPDATE and DELETE statements. This feature is unfortunately not included in the Personal Edition.

General

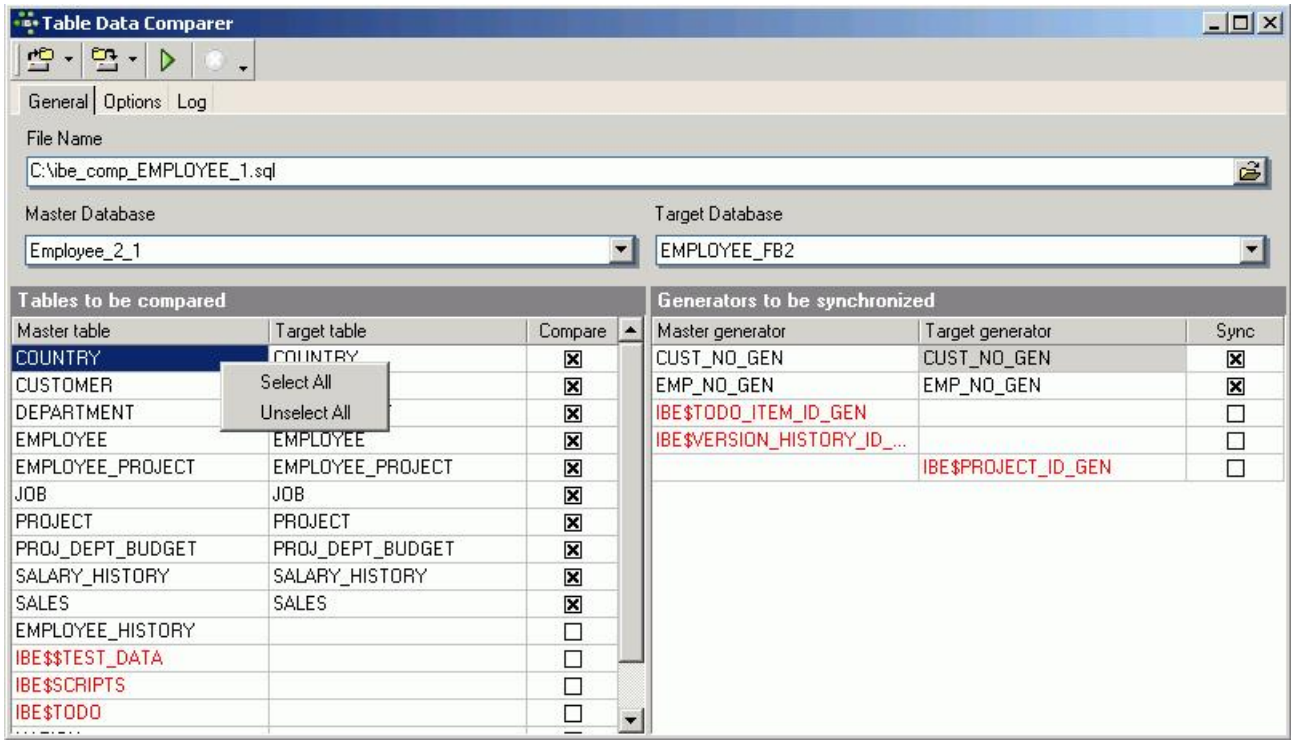
The *General* page displays the default file path and name for the resulting comparison script. This can of course be altered as wished.

The first step is to select the *Master* or *Reference Database* from the pull-down list of all registered databases. This is the reference database, to which the second database and its table(s) are to be compared. Then select the *Target Database*, i.e. the database whose table(s) need to be assessed and altered in order to conform with the reference database and table(s). The databases and tables must already exist.



Then select the tables to be compared. Tables with the same name in both databases are listed next to each other in the *Tables to be compared* list. If you wish to compare tables with different names, click the arrow to the right of the table field and select the desired table from the list of all tables in this database. Tables with different names must have the same structure. An error is raised if there is no primary key defined for the reference table.

To select all tables use the right-click context-sensitive menu.



As you will see in the illustration, system tables are not selected, even when using this funtion.

Selected generators/sequences can also be synchronized as part of the table comparison.

If you wish you can save your current settings into a file and load previously saved settings from file using the toolbar icons.

Options

The *Options* page allows:

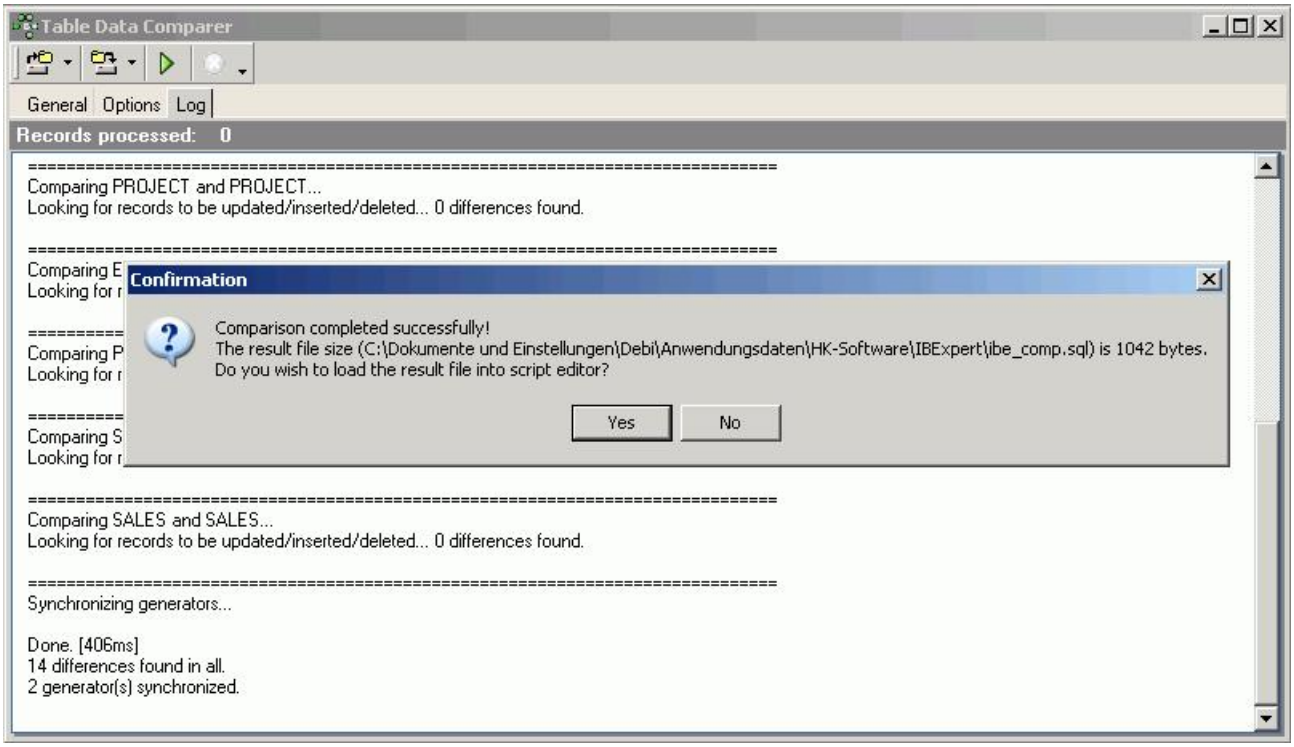
- Selection of `INSERT`, `UPDATE` or `DELETE` records.
- Option to include milliseconds into time/timestamp values.
- The options *Use ALTER SEQUENCE instead of SET GENERATOR* and *Use UPDATE OR INSERT instead of INSERT/UPDATE* are relevant for Firebird 2.0 and Firebird 2.1 respectively.

To start the table comparison simply click the *Compare* button (green arrow) or [F9].

Log

The Table Data Comparer resolves dependencies between master and detail tables while creating the script.

The resulting log:



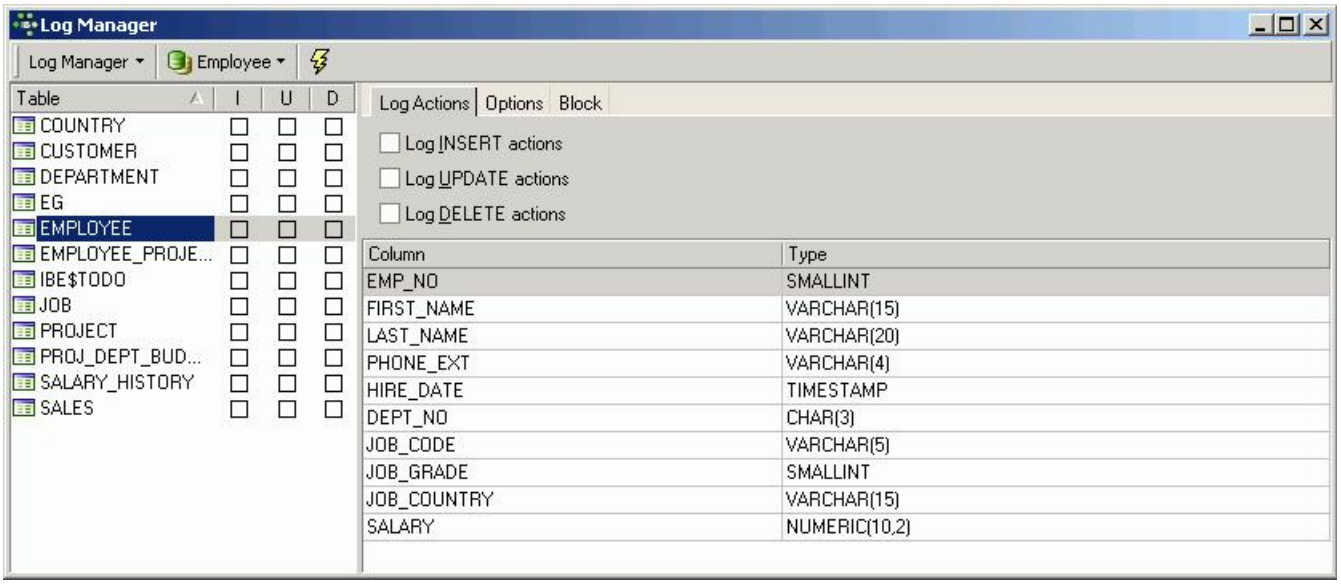
displays whether the database connections were successful, records searched, time taken and the number of discrepancies found. The resulting script file may then be loaded into the Script Executive if wished.

See also:
Database Comparer
IBECompare

Log Manager

The IBExpert Log Manager can be found in the IBExpert Tools menu. This tool is new to IBExpert version 2.5.0.47. This feature is unfortunately not included in the Personal Edition.

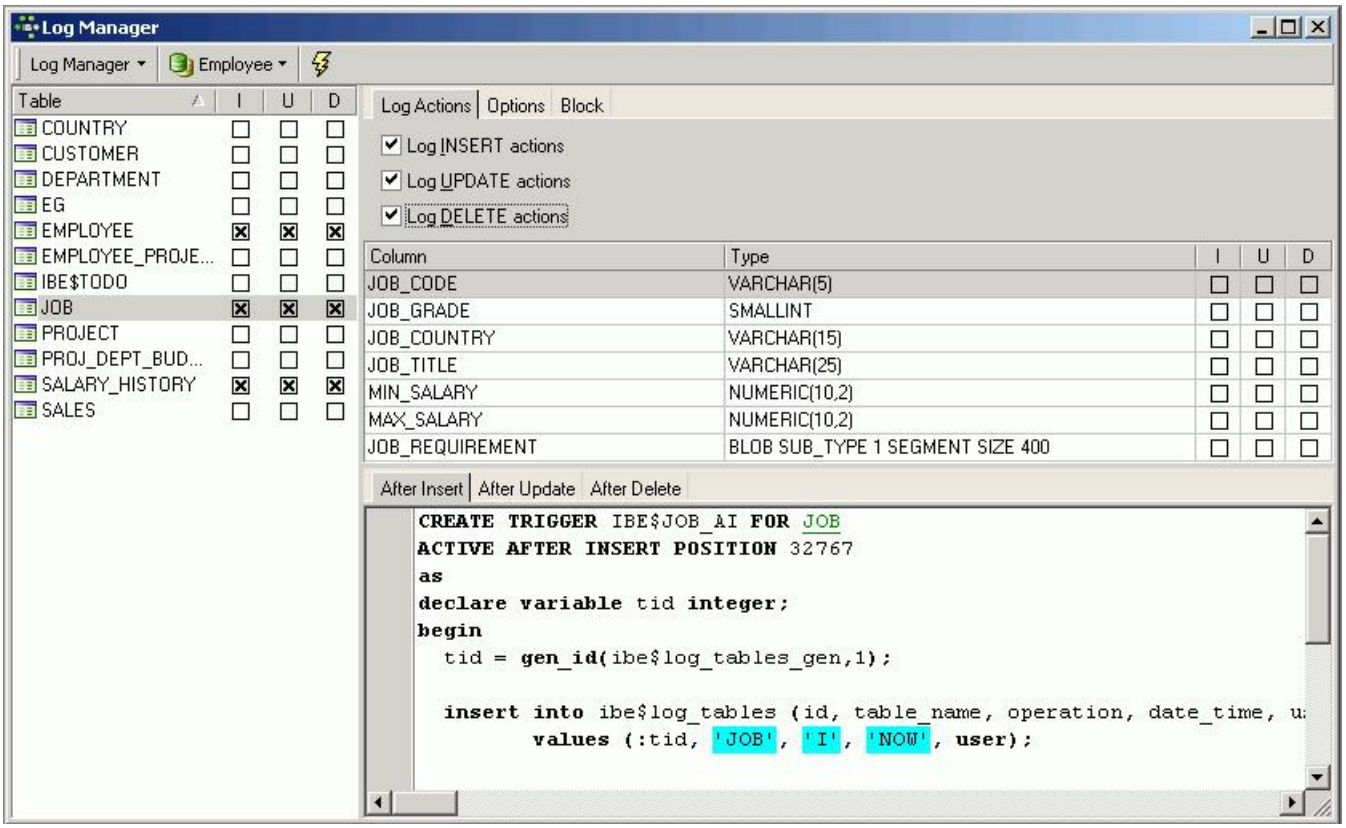
Select the database to be logged from the drop-down list of registered databases. When initially opened, the *Log Actions* page displays check options for logging INSERT, UPDATE and DELETE actions,



below which the selected table's fields and field types are displayed. The logging options, for example which INSERT, UPDATE and DELETE actions on which tables, can be checked individually or alternatively, the Log Manager pull-down menu can be used to either *Prepare All Tables* or to *Unprepare All Tables*. Take into consideration however, that when all actions on all tables are to be logged, this could slow the database performance somewhat.

Please note: all tables which are to be logged must be prepared for logging and committed, before any transactions can be logged! When new tables are added to a database, the log needs to be updated (simply select the transaction types which should be logged by double-clicking on the check boxes and compile).

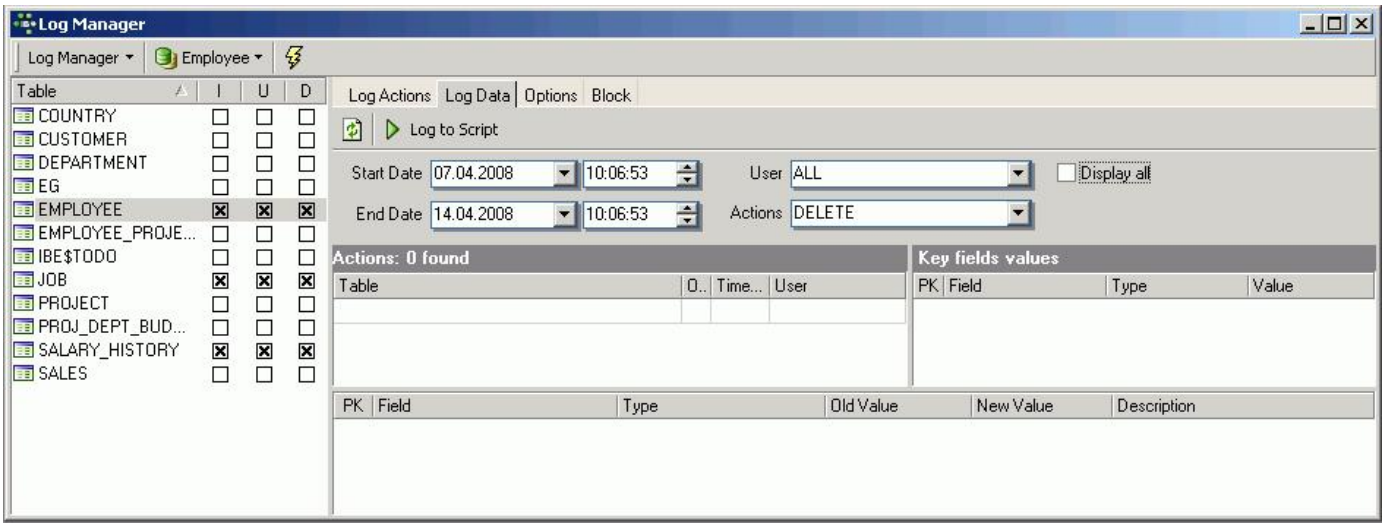
Once the actions have been selected, the *Log Actions* page displays the SQL code:



which can be copied to clipboard, if wished, using the right-click SQL Editor Menu.

New in version 2004.6.17 - templates have been added for data logging triggers. These can be altered as wished using the IBExpert IBExpert Options menu item, General Templates (Data Logging Triggers).

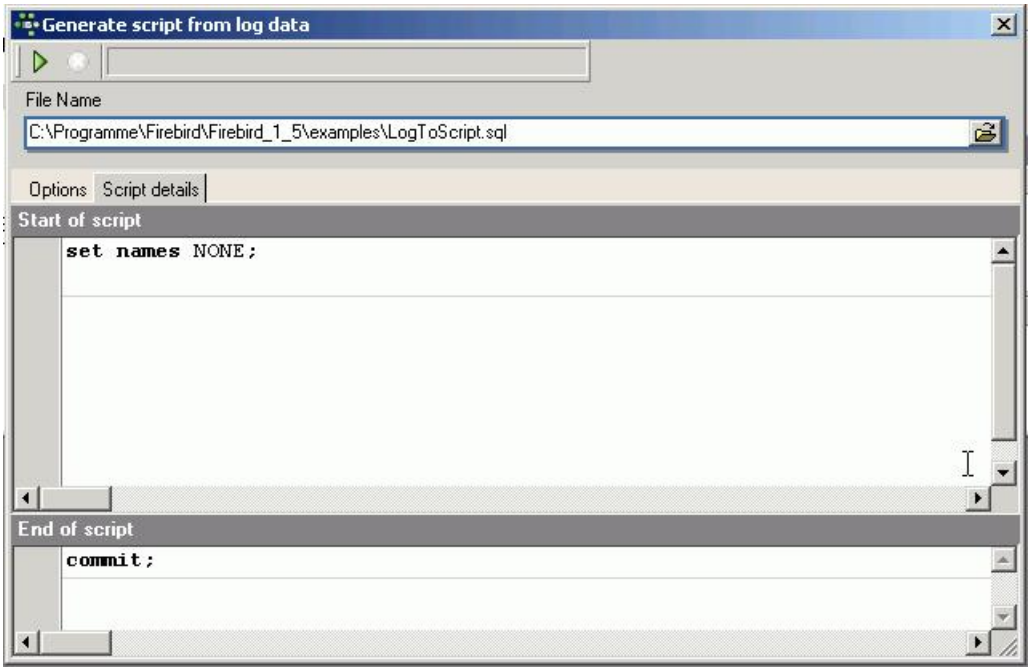
The *Log Data* page displays the new and old values:



In IBExpert version 2004.12.12.1 a new feature was added, allowing you to generate a log script for several tables simultaneously. Simply select the required tables using the [Ctrl + Shift] keys. And since IBExpert version 2005.02.12.1 64-bit IDs are now used when working with SQL Dialect 3 databases.

If a system error message appears when clicking on this page, stating that an IBExpert system table is missing, open any table from the DB Explorer and click on the *Logging* page in the Table Editor. You will then be automatically asked, whether IBExpert should generate certain system tables. After confirming and committing, you should have no further problems!

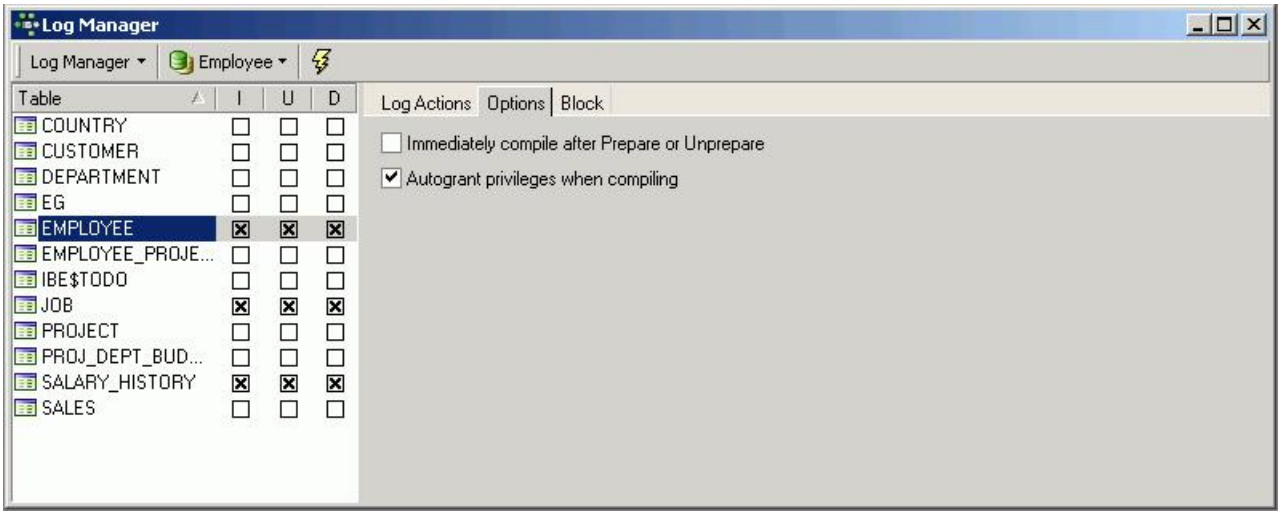
On the *Log Data* page the following can be user-specified: *Start Date*, *End Date* (both with timestamp), individual or all *users* and individual or all *actions*. The specified log can also be logged to file if wished, by clicking on the *Log to Script* button, which produces a new dialog box:



where the *Script File Name* can be specified, and on the *Options* page, how often a `COMMIT`-command should be inserted. Finally the *Script Details* page enables the user to write his own *Start of Script* and *End of Script*.

This Log file can even be used as a sort of replication. This is because, as opposed to the logging specified in the Database Registration, which only logs all IBExpert actions, the Log Manager logs all actions and operations on the database itself, including those of all users.

Back to the Log Manager Editor, the *Options* page:

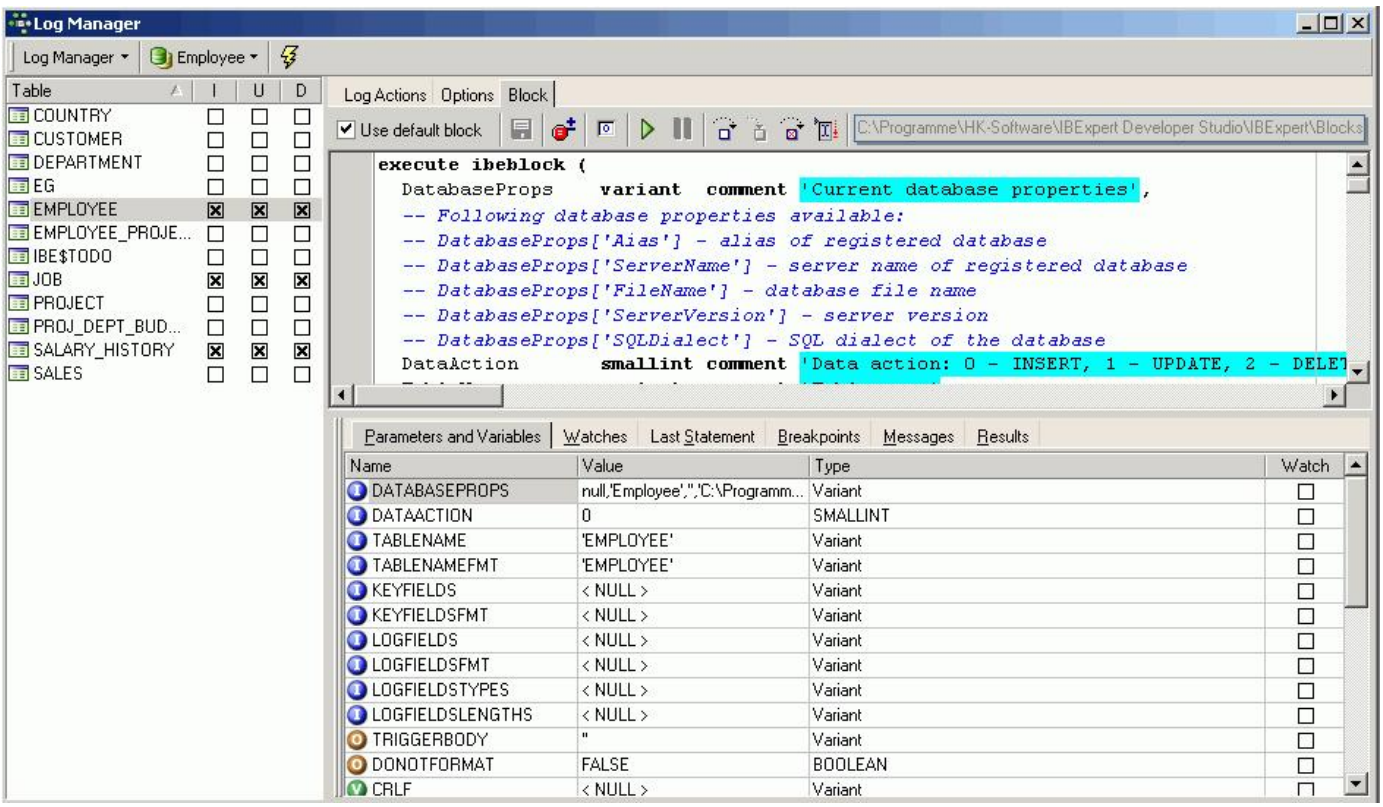


allows the user to specify the following options:

- *Immediately compile after Prepare or Unprepare*
- *Autogrant privileges when compiling* (generally this should be activated).

The item *Allow comparing BLOBS in AFTER UPDATE trigger* introduced in IBExpert version 2004.1.22.1, is now obsolete, because all actions with Blob fields are now customizable using trigger templates.

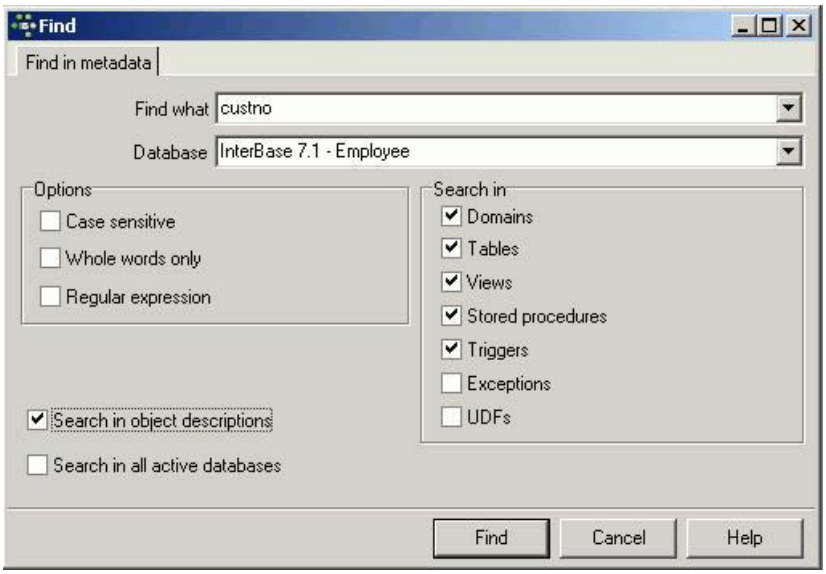
IBExpert version 2007.12.01 saw the introduction of the logging of trigger bodies based on the IBEBlock feature:



Search in metadata

The Search in Metadata option can be found in the IBExpert Tools menu, using the respective icon in the Tools toolbar, or started using the key combination [Shift + Alt + F]. It is identical to the Edit menu's Find option - *Find in Metadata* page.

This option is useful for finding individual words/digits or word/digit strings in metadata (and since IBExpert version 2004.8.5 also in object descriptions). It even searches for and displays field names, as opposed to the DB Explorer Filter, which only searches for object names. The *Find Metadata* dialog offers a number of options:

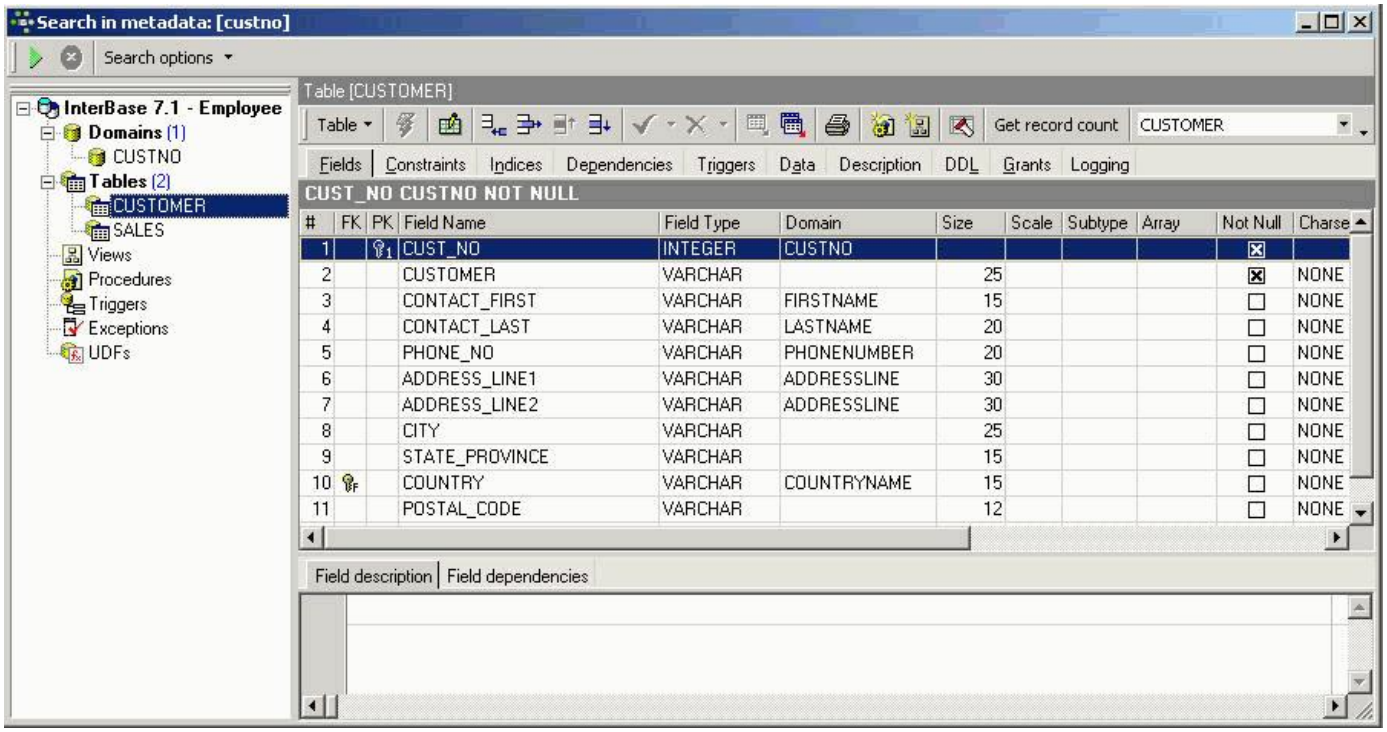


Here the user can specify what he is looking for; the pull-down list displays previous search criteria. A single active database may be selected from the second pull-down list; alternatively the *Search in all Active Databases* option can be checked, in the bottom left-hand corner of the dialog.

Further *Search* options include:

- **Case sensitive:** differentiates between upper and lower case
- **Whole words only:** as opposed to whole or parts of words
- **Regular Expression:** recognizes regular expressions in the search string.
- **Search in:** determines which object types should be searched - domains, tables, views, stored procedures, triggers, exceptions, UDFs.

After clicking on the *Find* button, a new *Search* dialog is opened:



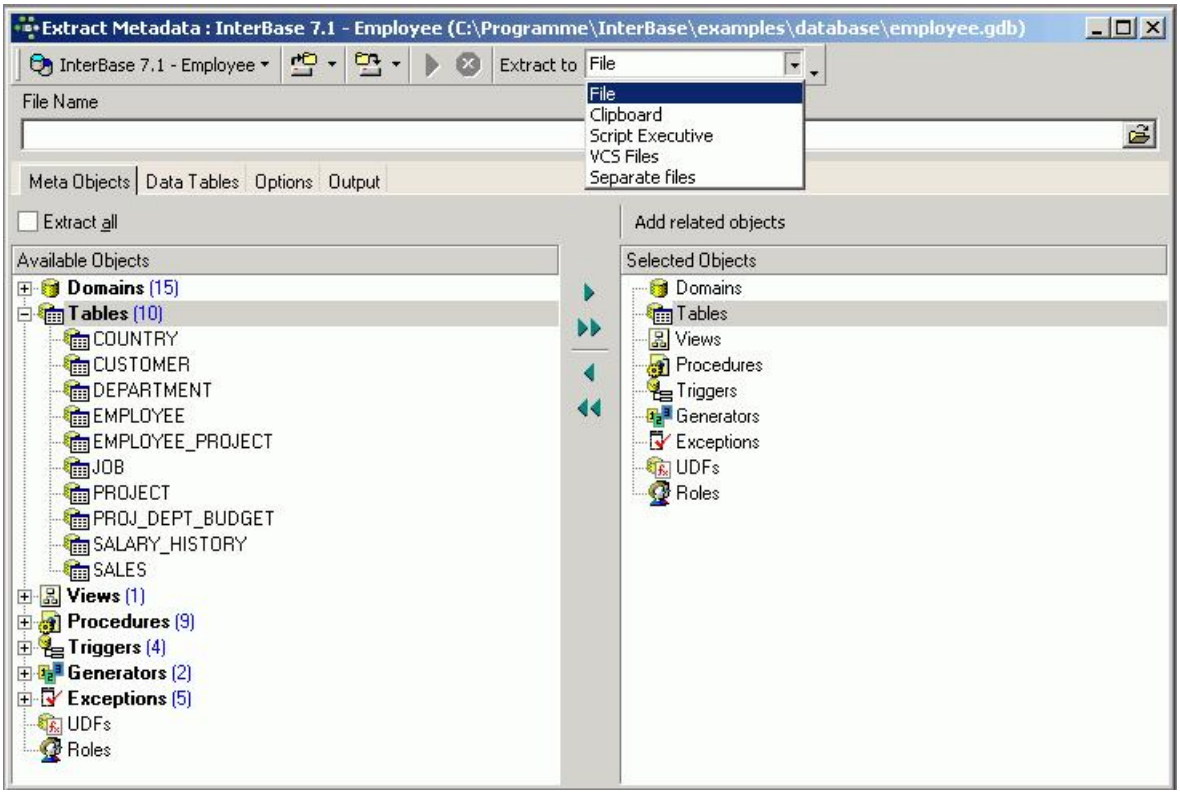
The *Search Options* button in the toolbar can be used to restart the *Find* dialog, in order to specify new *Search* conditions. The arrow to the right of this produces a drop-down overview of the search criteria specified.

The results of the Metadata Search are displayed in the usual IBExpert tree form, sorted by database object type. By clicking on an object, the object editor is opened in the *Search in Metadata* dialog, and can be edited as wished. Alternatively, a double-click on the tree object opens the object editor.

Extract metadata

The Extract Metadata menu item can be found in the IBExpert Tools menu, or started using the respective icon in the Tools toolbar.

The Extract Metadata module can be used to generate a partial or full database metadata script, including table data, privileges and objects descriptions if wished. It allows the user to extract metadata to file or clipboard. It is even possible to extract blob data and, since IBExpert version 2006.12.11, array fields' data (as blob data into a LOB file).



IBExpert version 2004.04.01.1 introduced the possibility to extract table data into separate files (`TABLE_1.sql`, `TABLE_2.sql`, `TABLE_3.sql` etc.). This version also includes support for default values of input parameters (Firebird 2). This option is particularly useful with extremely large scripts, as problems are often encountered executing scripts larger than 2 GB.

And since version 2004.1.22.1, it is also possible to extract `date`/`timestamp`/`time` values with ANSI-prefixes:

```
INSERT INTO MY_TABLE (DATE_FIELD, TIME_FIELD, TIMESTAMP_FIELD)
VALUES (date '01.01.2004', time '12:15:45',timestamp '01.01.2004 12:15:45');
```

IBExpert version 2006.10.14 altered the formatting of TIME values to `HH:MM:SS.zzz`.

Support for the InterBase 7.5 temporary tables feature was added in IBExpert version 2004.12.12.1, and IBExpert version 2006.06.05 introduced support for the Firebird 2.0 `NULL` clause.

IBExpert version 2007.02.22 introduced support for secondary database files information; the corresponding `ALTER DATABASE` statements are included into the result script as comments.

First a database needs to be selected from the toolbar's pull-down list of all registered databases. The toolbar's *Extract to* options include:

- File
- Clipboard
- Script Executive (default)
- VCS Files (previously, before IBExpert version 2004.9.12.1, named *Separate Files*)
- Separate Files (new to IBExpert version 2004.9.12.1)

The *Separate Files* mode extracts metadata (and data if specified) into a set of files: two files with metadata (`_ibe$start_.sql` and `_ibe$finish_.sql`), files containing table data (one or more files for each database table) and a `runme.sql file`, that consists of a number of `INPUT <file_name>` statements in the correct order.

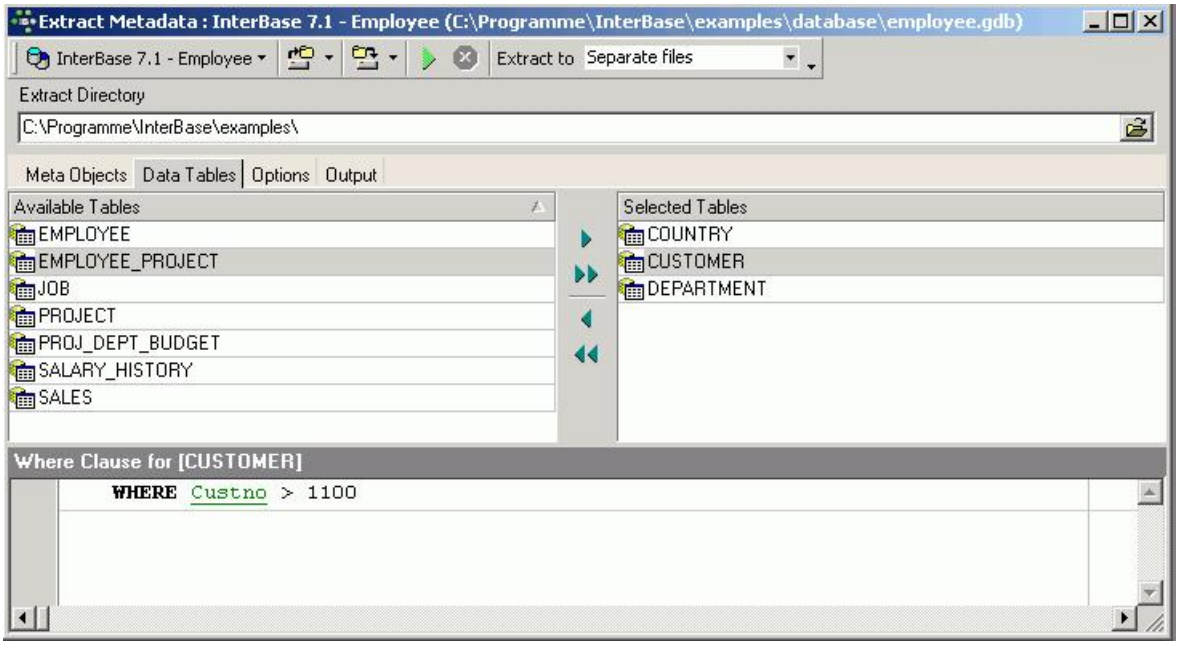
If either the *File*, *VCS Files* or *Separate Files* options are chosen, it is of course necessary to specify a file path and name (`*.sql` or *Metadata Extract Configuration* `*.mec`).

Meta Objects Page

The first dialog page *Meta Objects* displays the (please refer to this subject for further information).

Data Tables Page

The *Data Tables* page can be used to specify whether data should also be extracted. This allows both user-defined and system tables to be selected - either all or individually:

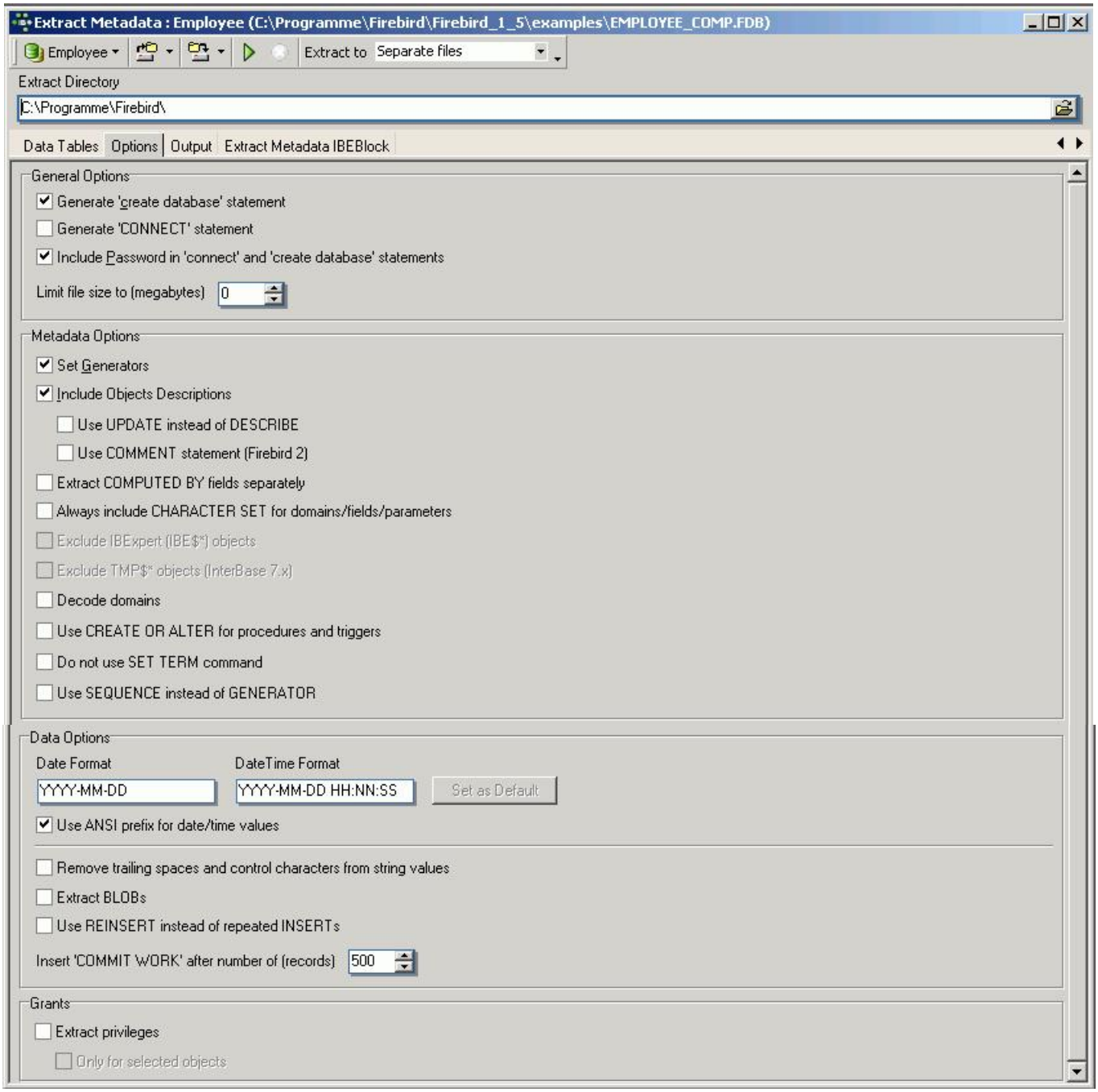


again using the <, >>, > or >> buttons, drag 'n' dropping or double-clicking.

By selecting one of the tables in the *Selected Tables* list, it is possible to add a `WHERE` clause, if wished.

Extract Metadata Options Page

The *Extract Metadata Options* page offers a wide range of further options:



These include:

General Options

- **Generate 'CREATE DATABASE' statement:** this determines whether a `CREATE DATABASE` statement should be included at the beginning of the generated script. If this option is unchecked, the `CONNECT` statement will be included instead.
- **Generate 'CONNECT' statement:** specifies the `CONNECT` statement.
- **Include password into 'CONNECT' and 'CREATE DATABASE' statements:** this determines whether the password should be included into the `CREATE DATABASE` or the `CONNECT` statement in the resulting SQL script.
- A **Limit File Size** option was added in IBExpert version 2004.9.12.1. This defines the maximum file size of the resulting script(s). When this option is specified and the maximum file size is reached, IBExpert automatically creates the next file with suffixes 0001, 0002 etc.

Metadata Options

- **Set Generators:** If this option is checked, the `SET GENERATOR` statement for each generator will be included into the resulting script.
- **Include object descriptions:** this determines whether database objects descriptions should be included into the generated script. See How does IBExpert extract objects descriptions? for more details.
  - **Use UPDATE instead of DESCRIBE:** New to IBExpert version 2005.04.24.1, this option allows you to check the new Firebird 2 feature *Extract Metadata / Use UPDATE instead of DESCRIBE*. If it is enabled, IBExpert will generate an `UPDATE RDB$xxx SET RDB$DESCRPTION ...` statement instead of `DESCRIBE` while extracting metadata. You first need to check the option, *Use UPDATE instead of DESCRIBE when extracting object descriptions*, found in the IBExpert Database menu item, Register Database or Database Registration Info under Additional / Extract Metadata. By default it corresponds to the value specified in the Database Registration Info.
  - **Use COMMENT statement (Firebird 2):** was introduced in IBExpert version 2005.09.25. This forces object descriptions to be extracted as a set of `COMMENT` statements.
- **Extract COMPUTED BY fields separately:** this option can be used to specify whether computed fields should be extracted separately (useful if there are bugs in the database; realistically however this option is seldom used).
- **Always include the CHARACTER SET for domains/fields/parameters**.

- **Exclude IBExpert (IBE$*) objects:** check option.
- **Exclude TMP$* objects (InterBase 7.x):** check option.
- Since version 2004.2.26.1 there is also the added option **Decode domains**. If enabled, the domain types will be inserted as comments just after domain names. For example:

```
   CREATE TABLE Z (
B BOOL /* INTEGER DEFAULT 0 CHECK (VALUE IN(0,1)) */
   );
```

- **Use CREATE OR ALTER for procedures and triggers:** compliant to Firebird 2.x. Introduced in IBExpert version 2007.09.25.
- **Do not use SET TERM command:** `SET TERM` is not necessary for scripts executed by IBExpert/IBEScript but may be necessary when working with other tools. Introduced in IBExpert version 2007.09.25.
- **Use SEQUENCE instead of GENERATOR:** compliant to Firebird 2.x.

Data Options

- **Date Format:** this can be used to specify the date format and datetime format, with options to use an ANSI prefix for date/time values and to set the specified format as default.
- **Remove trailing spaces and control characters from string values**
- **Extract Blobs:** IBExpert cannot "read" blobs; it therefore uses a detour to make a reference to a separate database file containing such blobs. Only IBExpert has been able to do this so far. Other products only extract the definition of the blobs, and not the contents themselves.
- **Use REINSERT instead of repeated INSERTs:** uses the IBExpert `REINSERT` command, to insert multiple data records.
- **Insert 'COMMIT WORK' after number of (records):** this option defines the number of records before inserting the `COMMIT` statement into the script. The default value is `500`, i.e. 500 `INSERT` commands are performed and then committed.

Grants

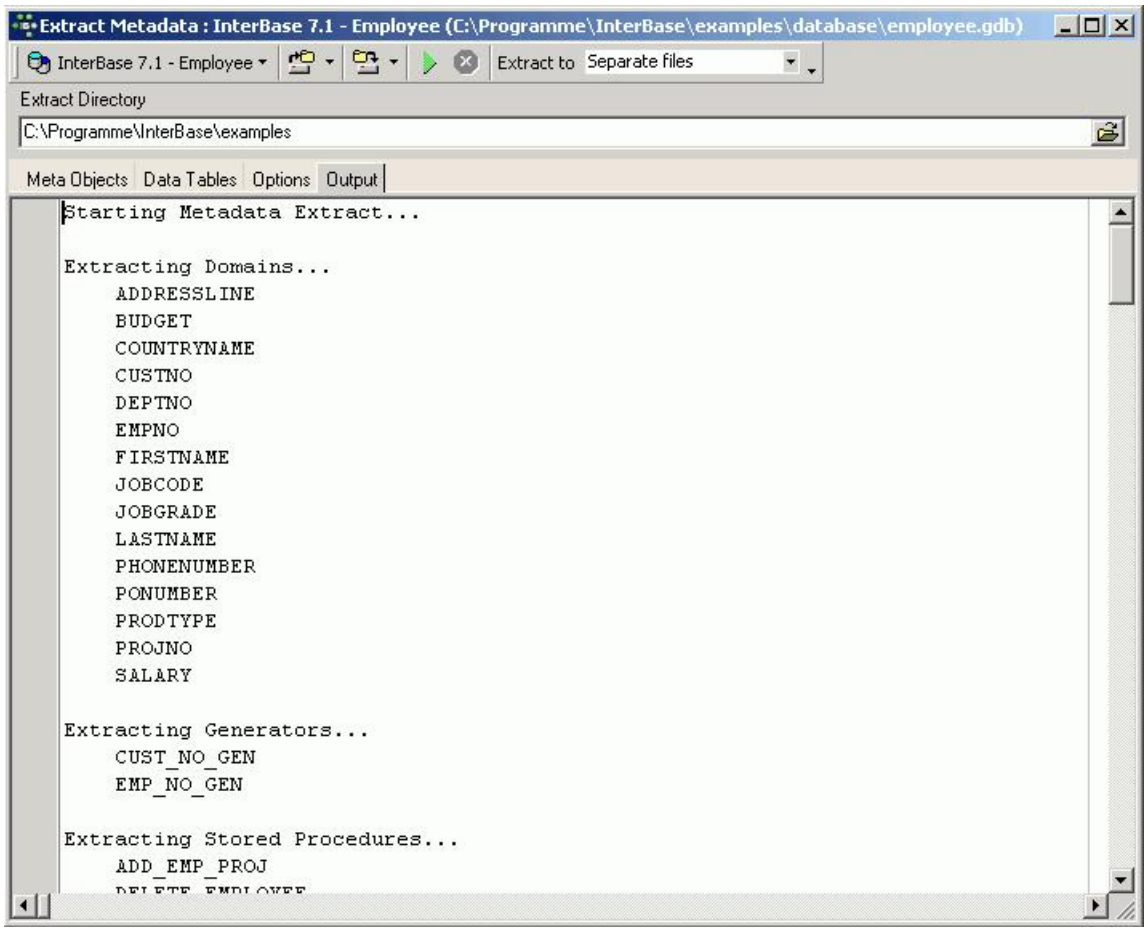
- **Extract privileges:** for all or only for selected objects.

Finally, if wished, use the toolbar icon *Save Configuration to File* or the key combination [Ctrl + S] to save this configuration as a template for future use. The next time round, the template can be quickly and easily loaded using the *Load Configuration* icon (or [Ctrl + L]); the template specifications amended if necessary, and the extract started!

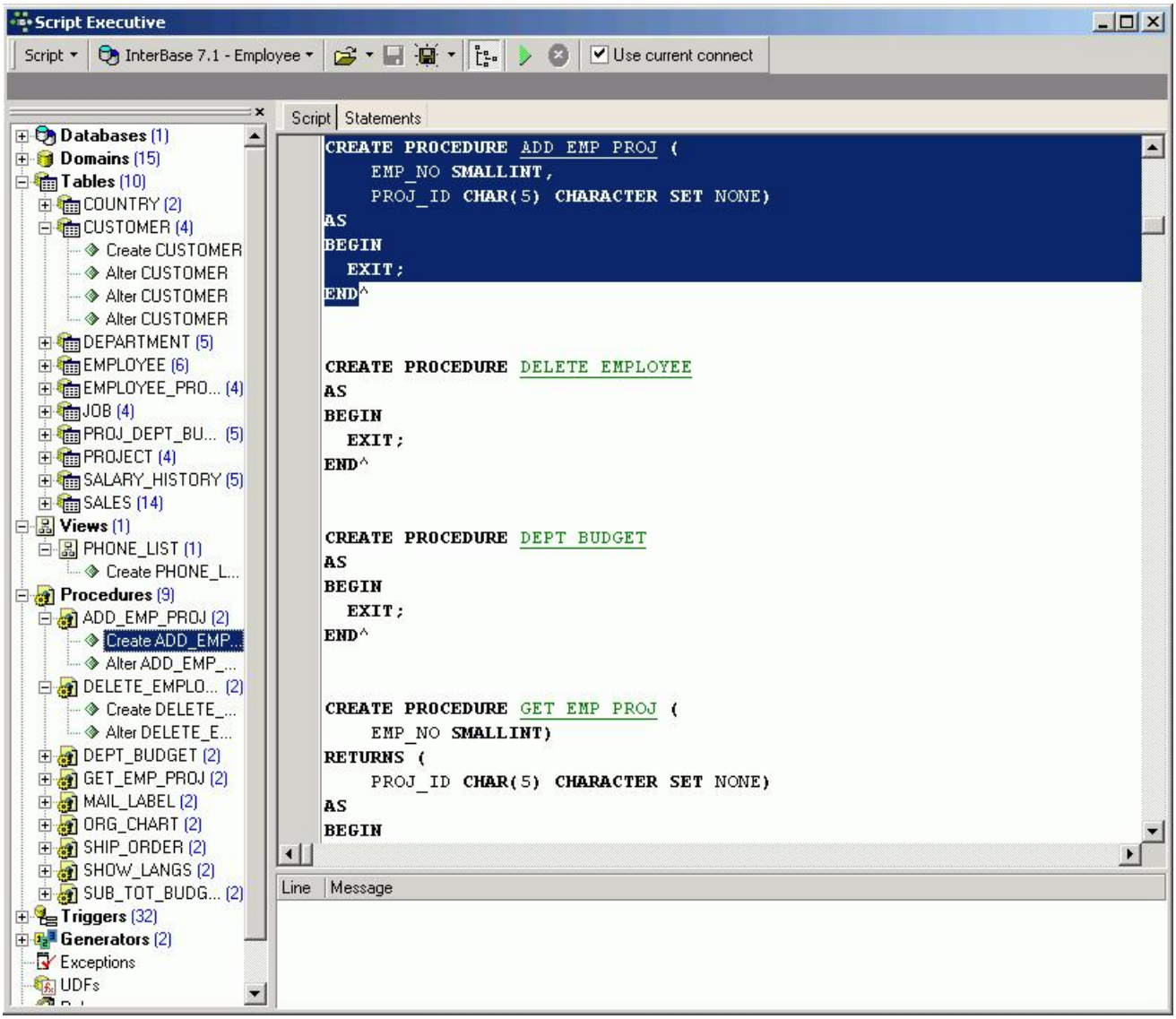
Once all objects have been selected, and all options specified, the extract can be started using the green > button or [F9].

Output Page

The *Output* page displays the IBExpert log during the extraction. Following completion, if a file was specified, IBExpert asks whether the file should be loaded into the script editor. Since IBExpert version 2007.09.25 it is possible to create scripts larger than 2 GB.



If the Script Executive has been specified as the output option, the Script Executive is automatically loaded. The object tree on the left-hand side can be opened to display the individual statements relating to an object. By clicking on any of these statements, IBExpert springs to that part of SQL code, which is displayed on the right:



The statements display what IBExpert is doing and in which order. The script displays the creation of all objects, and then the subsequent insertion of the content data, using the `ALTER` command.

Extract Metadata is a great tool, and can be useful in a variety of situations. For example, it can be used to perform an incremental backup, should it be necessary for example, to back up just the `EMPLOYEE` table every evening.

Any number of configurations may be saved in various formats:

- **Metadata extract configuration (*.mec):** this allows you to quickly and simply load a specified configuration in the *Extract Metadata* dialog.
- **IBEBlock (*.ibeblock):** new to IBExpert version 2006.06.05, this enables you to save the current settings as an `EXECUTE STATEMENT` statement. IBExpert creates a valid IBEBlock with the `ibec_ExtractMetadata` function, which may be used later in scripts.
- **All files (*.*)**.

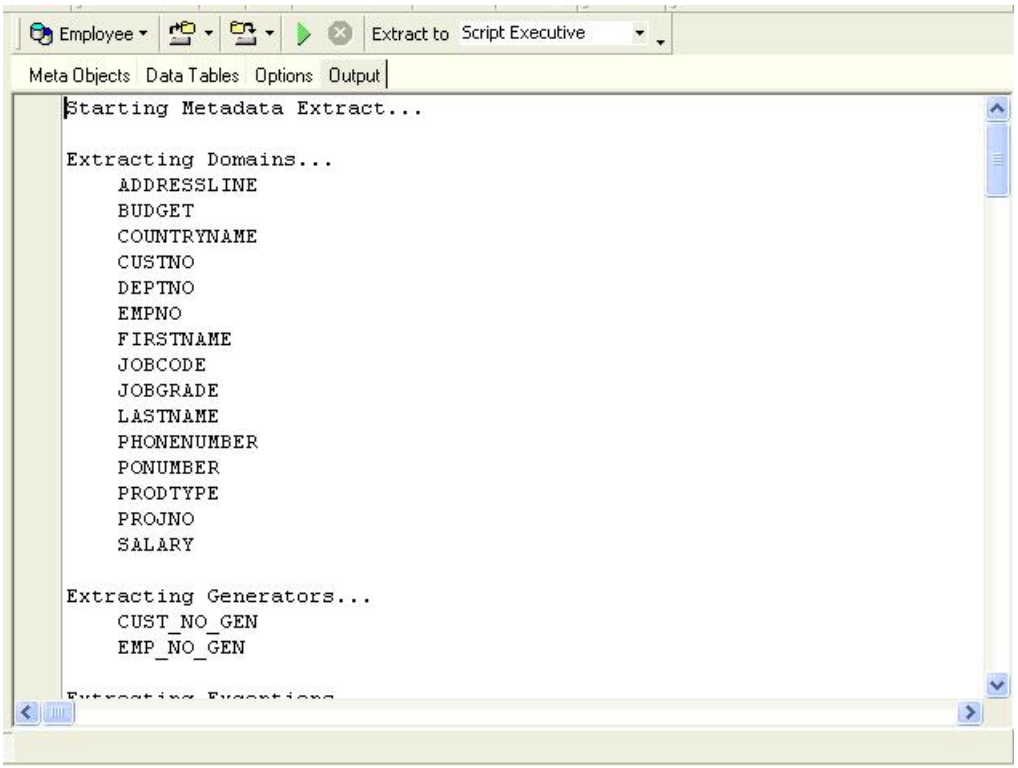
Metadata

Metadata includes the definition of the database and database objects such as domains, generators, tables, constraints, indices, views, triggers, stored procedures, user-defined functions (UDFs), blob filters. Metadata is stored in system tables, which are themselves part of every InterBase/Firebird database.

Metadata includes all those SQL statements necessary to recreate the database object. It includes the following elements:

- `CREATE DATABASE` statement
- `CREATE DOMAIN` statements
- `CREATE TABLE` statements
- declarative referential integrity using the `ALTER TABLE` statement
- `CREATE GENERATOR` statements
- `CREATE VIEW` statements
- check constraints using `ALTER TABLE` statements
- `CREATE EXCEPTION` statements
- procedure definitions using `CREATE PROCEDURE` or [[DDL - Data Definition Language[[#Alter |`ALTER PROCEDURE`

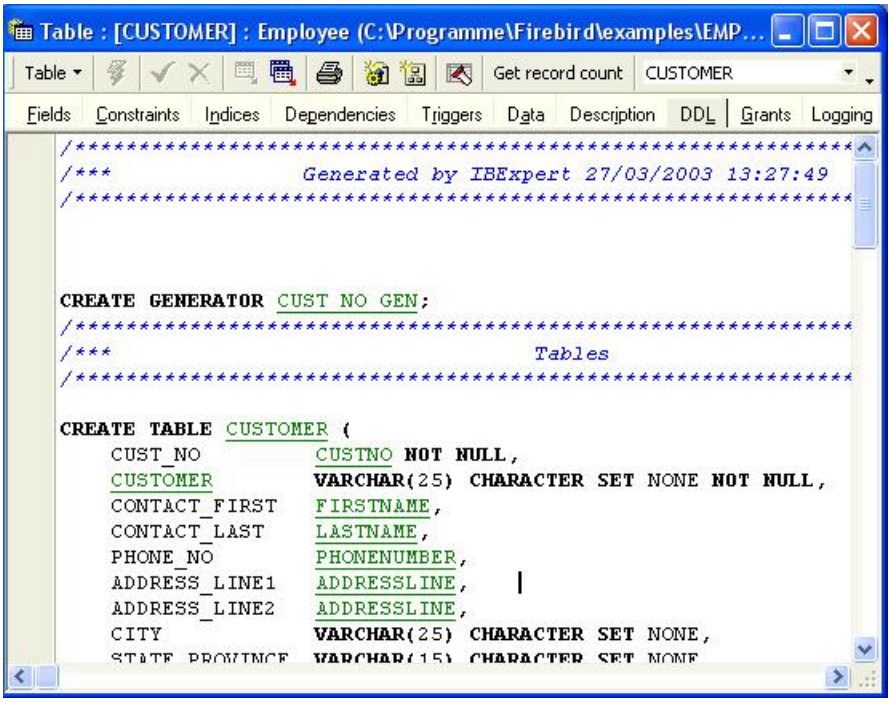
- trigger definitions using CREATE TRIGGER statements
- granting of user authorizations for tables, views and stored procedures.



Metadata for a table includes all domains and generators used by these tables plus the CREATE TABLE statement. It does not include any referential integrity definitions from this table to other tables or from other tables to this table.

Metadata for a view only includes the CREATE VIEW statement.

The current metadata definitions can be viewed on the DDL page in the individual object editors.



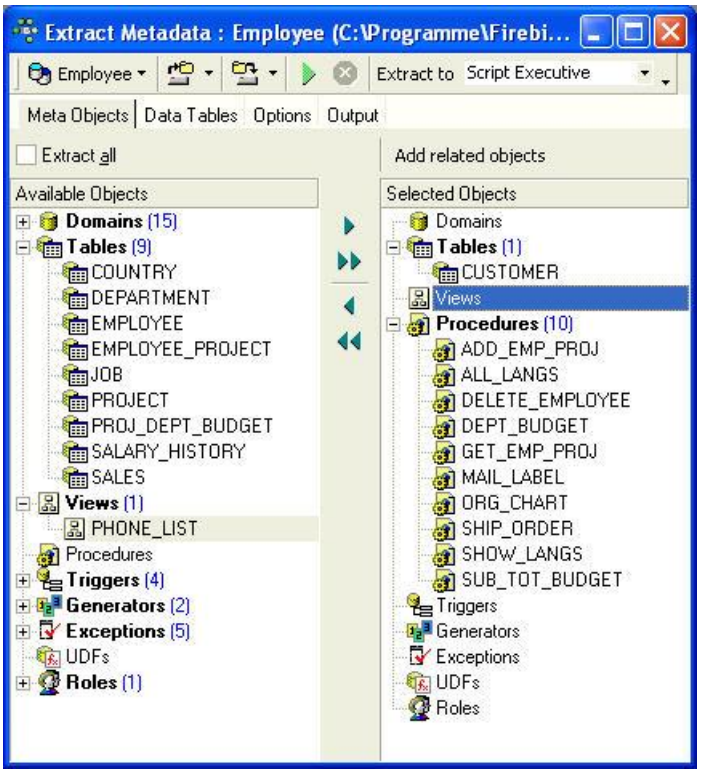
The IBExpert menu item Tools / Extract Metadata can be used to extract all metadata for a database. The resulting script can be used to create a new empty database. When the *Options Data Tables* and *Options - Extract Blobs* are used, the script contains the complete database with all data.

Select Objects Tree

The *Select Objects Tree* dialog can be found in the following editors:

- Extract Metadata Editor on the first page, *Meta Objects*,
- Generate HTML Documentation Editor, also on the *Objects* page,

- Print Metadata dialog.



The *Select Objects Tree* feature offers the user the choice whether to extract all database objects (check option), or specify individual objects, (using the < or > buttons, drag 'n' dropping the object names or double-clicking on them), or object groups (using the << or >> buttons, drag 'n' dropping the object headings or double-clicking on them).

Multiple objects can be selected using the [Ctrl] or [Shift] keys. There is even the option to *Add Related Objects* by using the button above the *Selected Objects* window.

How does IBExpert extract objects descriptions?

IBExpert uses a special extension of script language that enables it to extract objects' descriptions into script and then execute one using the Script Executive.

How does IBExpert extract blobs?

IBExpert uses an original mechanism to extract values of blob fields into a script. This allows you to store an entire database (metadata and data) in script files and execute these scripts with IBExpert. The following small example illustrates out method to extract blob values.

For example, a database has a table named `COMMENTS`:

```
CREATE TABLE COMMENTS (   COMMENT_ID INTEGER NOT NULL PRIMARY KEY,   COMMENT_TEXT BLOB SUBTYPE TEXT);
```

This table has three records:

COMMENT_ID	COMMENT_TEXT
1	First comment
2	NULL
3	Another comment

If the *Extract BLOBs* option is unchecked you will get the following script:

```
CREATE TABLE COMMENTS (   COMMENT_ID INTEGER NOT NULL PRIMARY KEY,   COMMENT_TEXT BLOB SUBTYPE TEXT);

INSERT INTO COMMENTS (COMMENT_ID) VALUES (1);
INSERT INTO COMMENTS (COMMENT_ID) VALUES (2);
INSERT INTO COMMENTS (COMMENT_ID) VALUES (3);
```

... and, of course, you will lose your comments if you restore your database from this script.

But if the *Extract BLOBs* option is checked, IBExpert will generate a somewhat different script:

```
SET BLOBFILE 'C:\MY_SCRIPTS\RESULT.LOB';
```

```
CREATE TABLE COMMENTS (
   COMMENT_ID INTEGER NOT NULL PRIMARY KEY,
   COMMENT_TEXT BLOB SUBTYPE TEXT);

INSERT INTO COMMENTS (COMMENT_ID, COMMENT_TEXT) VALUES (1, h0000000_0000000D);
INSERT INTO COMMENTS (COMMENT_ID, COMMENT_TEXT) VALUES (2, NULL);
INSERT INTO COMMENTS (COMMENT_ID, COMMENT_TEXT) VALUES (3, h000000D_0000000F);
```

IBExpert also generates a special file with the extension `LOB`, where blob values are stored. In the current example `result.lob` will be 28 bytes long and its contents will be `First commentAnother comment`.

`SET BLOBFILE` is a special extension of script language that allows the IBExpert Script Executive to execute scripts containing references to blob field values.

Obtain current generator values

There are two methods to obtain the current generator values in a database. The first is using the IBExpert menu item Tools / Extract Metadata, where there is an option to set generators on the *Options* page.

In Firebird this can also be done using a stored procedure:

```
CREATE PROCEDURE GET_GENERATORS
RETURNS (
    GENERATOR_NAME CHAR(31),
    CURR_VAL BIGINT)
AS
declare variable sql varchar(100);
BEGIN
  FOR
    select r.rdb$generator_name generator_name, cast(0 as bigint) curr_val from rdb$generators r
    where r.rdb$generator_name not containing '$'
    INTO :GENERATOR_NAME,
         :CURR_VAL
  DO
  BEGIN
    sql='Select gen_id('||GENERATOR_NAME||',0) from rdb$database';
    execute statement :sql into :curr_val;
    SUSPEND;
  END
END
```

Database repair using Extract Metadata

The Firebird core package has no dump tool. So it's important to analyze your metadata scripts to trace what started to go wrong, where and when.

If your backups are failing regularly on the same table(s) due to irreparable data damage, and you've not been able to solve the problem using `GFIX`, this is an alternative way to save at least all remaining healthy data and the database itself.

First attempt to restrict the problem to as few data sets as possible, using the `SELECT` command on the table ID field.

1. Then use the IBExpert Tools menu item, Extract Metadata. Connect to your database and select all tables for metadata and data.
2. *Extract into* - select *separate files* from the drop-down list.
3. Extract all objects and data from all tables.
4. If any error occurs on specific data, add a `WHERE` condition for the table concerned. For example, click on the table name in the right-hand column of *Selected Objects* and add your `WHERE` clause to exclude the range of damaged data, e.g. `WHERE ID>1000 AND ID<1100`.
5. Generate your script.
6. Delete the original database file.
7. If required, add the missing data as far as possible from an older extract file or backup copy of the database.
8. Execute `runme.all.bat` (don't forget to add the path to `IBEScript.exe`. This starts IBExpert's `IBEScript`, `runme.all.sql`, which loads the files from `IBE$Start`, then the data files and finally `IBE$Finish`.

This will create a new database with all objects and data, even including blob data.

IBE$Start funs the operations such as creating the database and metadata. Tables are generated, without any primary keys, foreign keys, constraints, triggers, etc. This is followed by a series of insert commands, using the IBEBlock function, REINSERT. IBE$Finish then inserts all primary keys, foreign keys etc.

You can, of course carry all this out at script level, using ibec_ExtractMetadata.

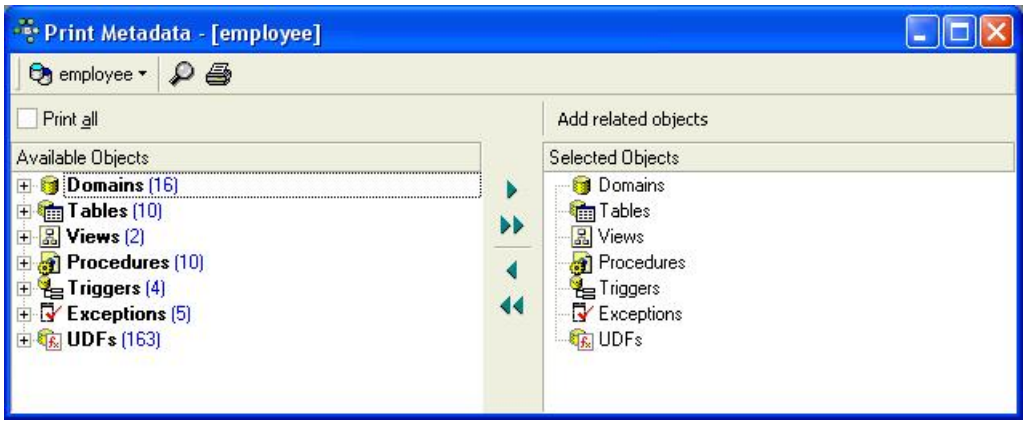
This method can of course also be used, if you wish to make an alteration to an existing database, for example, update from SQL dialect 1 to 3, or specify a character set if no default character set was specified at the time of database creation. For example, to alter the default character set from NONE to ISO8859_1, simply open IBE$Start, search CHARACTER SET NONE and replace with CHARACTER SET ISO8859_1, and then run the runme.all.sql script, as mentioned above.

See also:
IBEExtract
IBEScript
ibec_ExtractMetadata

Print metadata

Print Metadata prints the database metadata, along with dependencies, description, and other options for any database object or object group, providing a quick and yet extremely comprehensive database documentation. The information is printed as a report, using IBExpert's report templates. Using the Report Manager, these reports can also be customized (the *Print Metadata* standard report templates can be found in the `IBExpert\Reports\ directory`). This is of particular importance for those businesses working according to DIN certification/ISO standards.

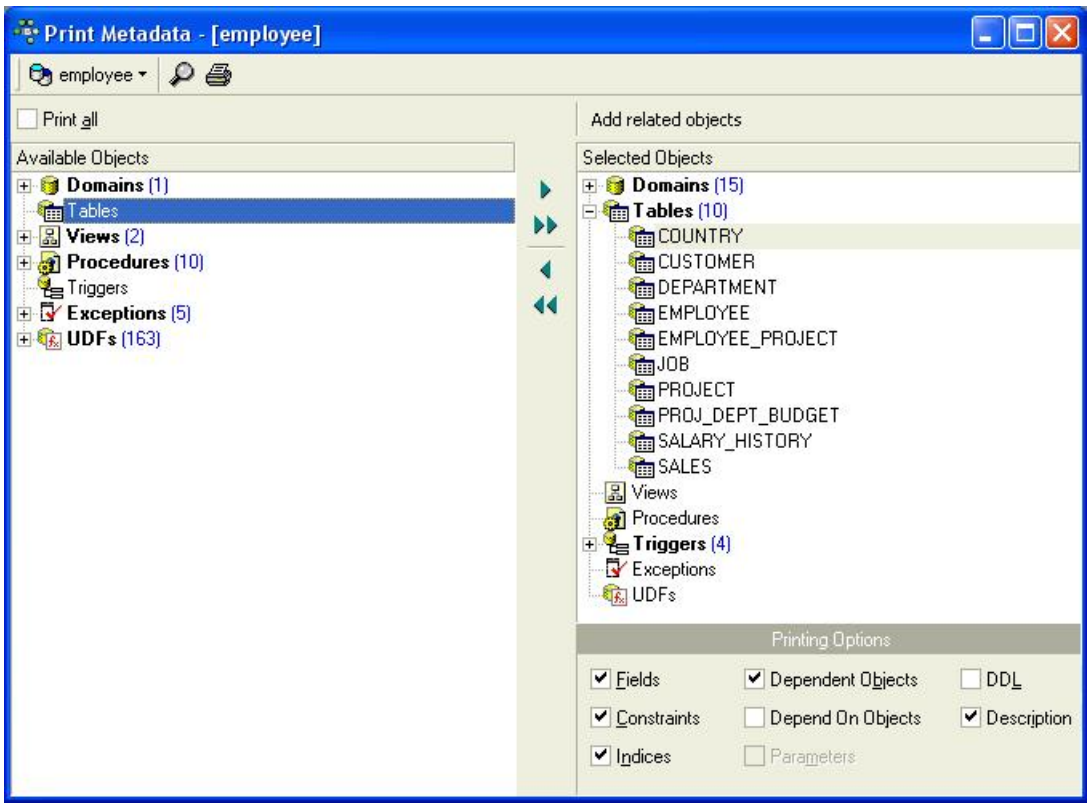
The Print Metadata menu item can be found in the IBExpert Tools menu, or started using the *Printer* icon in the Tools toolbar.



The *Print Metadata* Editor is similar to the *Extract* Metadata Editor. First select one of the registered databases using the top left toolbar button. Then select the objects to be printed. It is possible to check *Print All*, or specify individual database objects (using the < or > buttons, drag 'n' dropping the object names or double-clicking on them), or object groups (using the << or >> buttons, drag 'n' dropping the object headings or double-clicking on them). Multiple objects can be selected using the [Ctrl] or [Shift] keys.

There is even the option to *Add Related Objects* by using the button above the *Selected Objects* window.

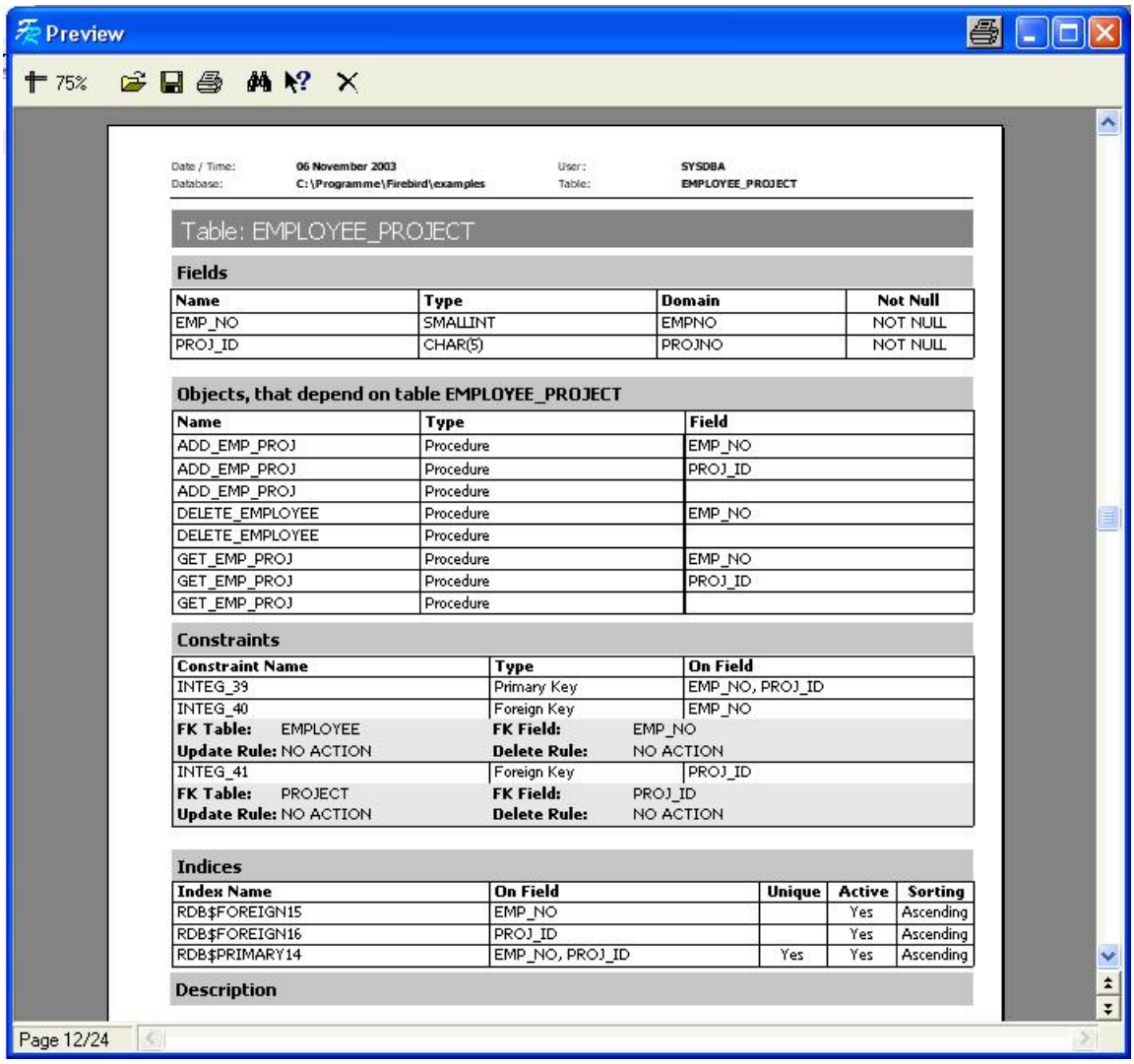
When one of the selected database objects or object groups is highlighted, a number of check options appear in the lower right panel. These include:



- fields
- constraints
- indices
- dependent objects
- depend on objects
- parameters
- DDL
- description

In order to print a complete database documentation it is of course necessary to select all database objects, and then check all options for each object group. This could however lead to difficulties in the case of very large databases, despite the Report Manager's amazing speed!

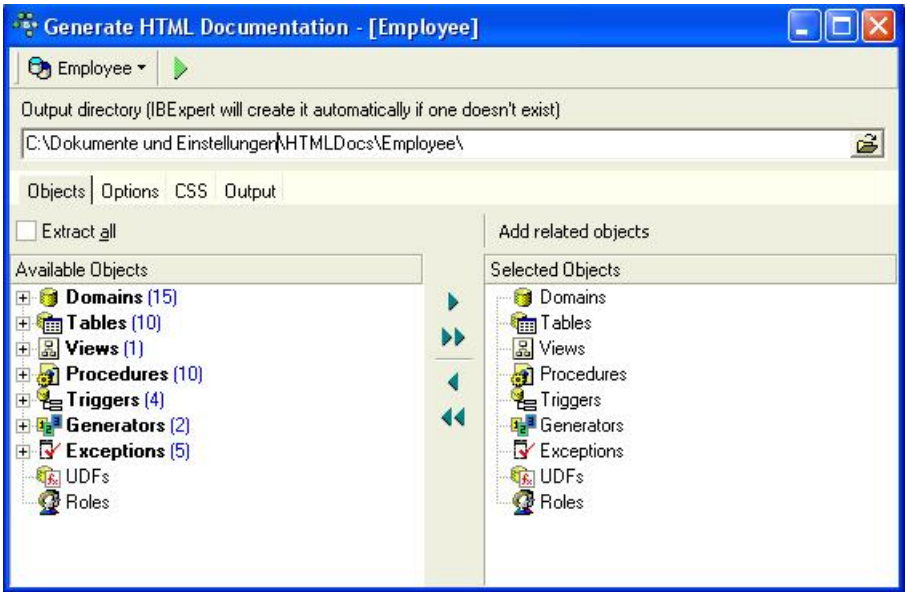
It is possible to print the report directly from this dialog or preview it first, using the *Magnifying Glass* icon.



This opens the *Fast Report Preview* page, which displays the report as it will be printed, and furthermore offers options such as saving the report to file and searching for text.

Generate HTML documentation

Using the IBExpert Tools menu, HTML documentation can be generated for a named, connected database. This option is an excellent feature for software documentation, particularly if an object description was always inserted as objects were created. For those working with an older version of IBExpert: versions before 2.5.0.47 do not include all of the features detailed here.

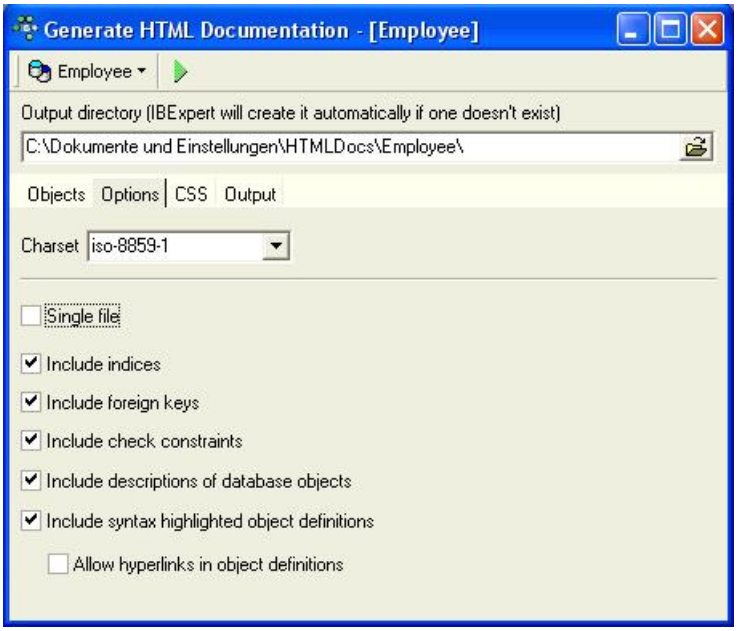


The toolbar displays the selected connected database. The pull-down lists offers a choice of all connected databases.

The default output directory can be overwritten if wished.

The *Generate HTML Documentation* Editor is similar to the *Extract Metadata* Editor. The *Objects* page allows single or groups of database objects to be selected for the HTML documentation. Database objects can be specified individually using the < or > buttons, drag'n'dropping the object names or double-clicking on them, or object groups may be specified using the << or >> buttons, drag 'n' dropping the object headings or double-clicking on them. Multiple objects can be selected using the [Ctrl] or [Shift] keys. Alternatively the *Extract All* box can be checked, allowing documentation to be generated for the complete database.

There is even the option to *Add Related Objects* by using the button above the *Selected Objects* window.



The *Options* page lists a series of check boxes including:

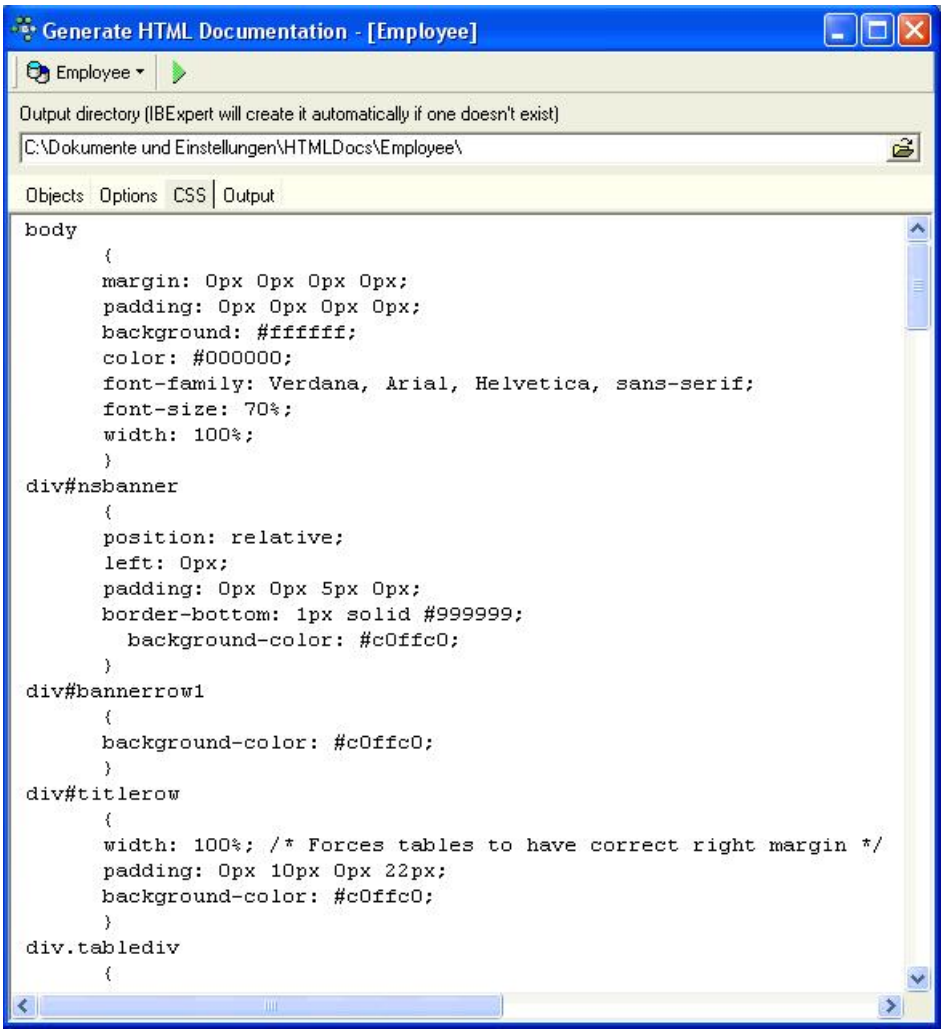
- single file (i.e. whether one complete file, as opposed to several smaller files should be generated)

and whether:

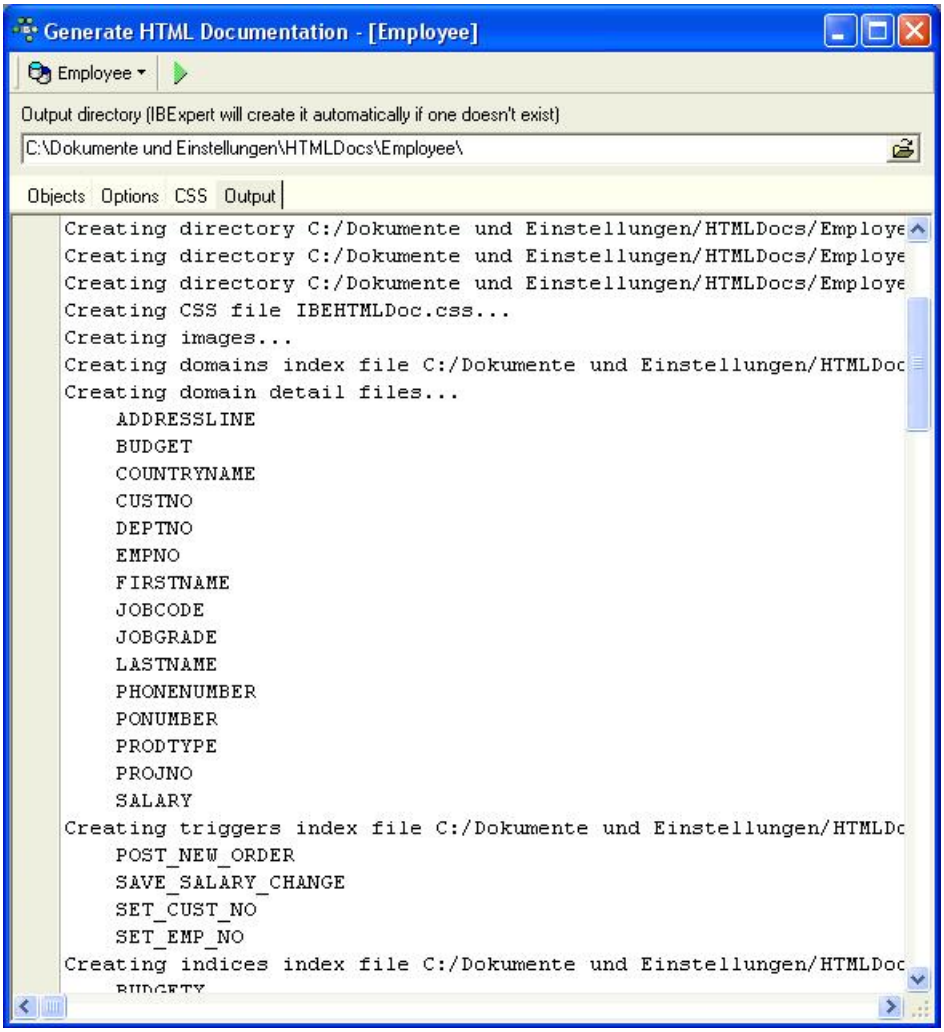
- indices

- foreign keys
- check constraints
- database object descriptions
- syntax highlighted object definitions
- hyperlinks in object definitions

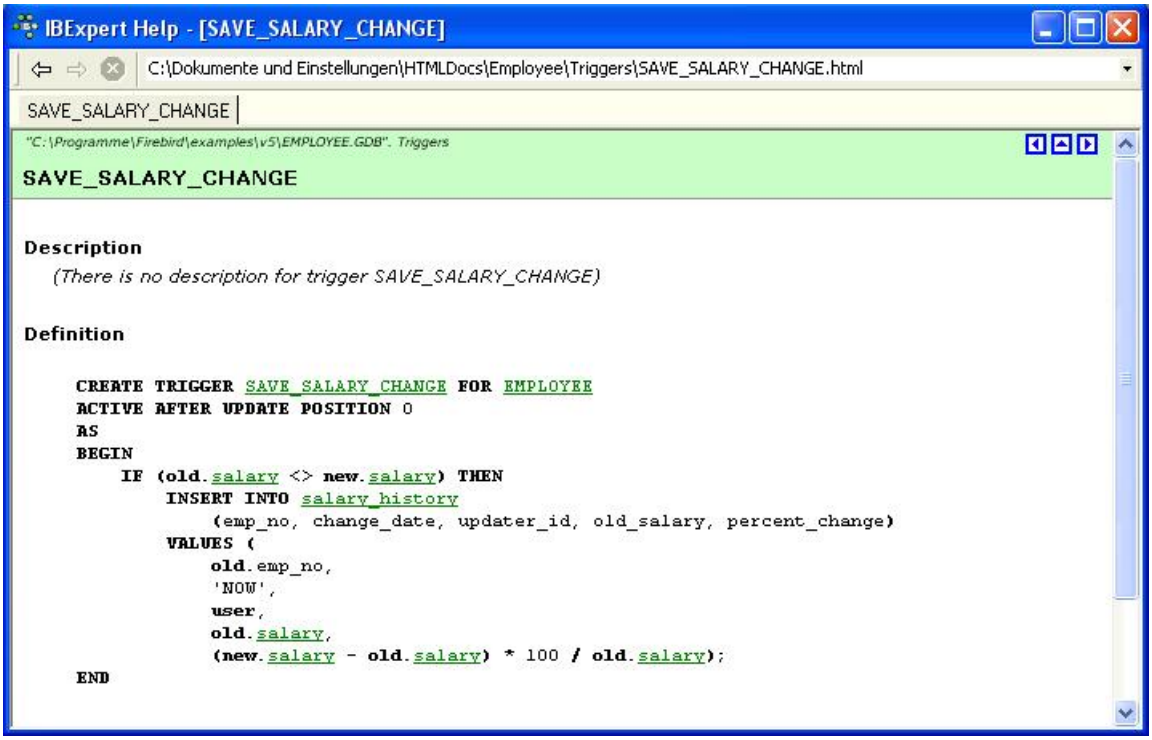
should be included.



The *CSS* or cascaded style sheets page displays the code for the HTML page template.



The *Output* page displays the code used to generate the HTML documentation.



By clicking on one of the object subjects, such as triggers, a table of all such objects (i.e. all triggers) for this database appear. Clicking on the individual objects then automatically displays the description (if existent) and the definition.

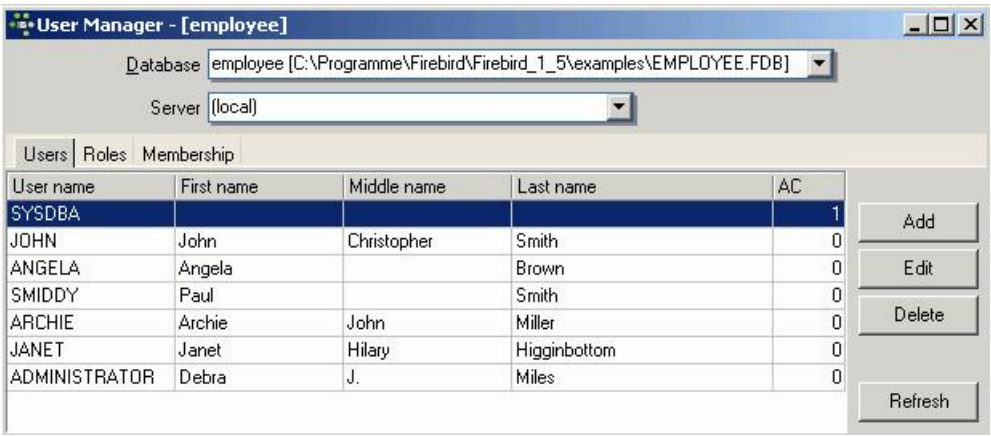
CSS - Cascaded Style Sheets

Cascaded style sheets (CSS) are an option included in the Generate HTML Documentation menu (second page in the main dialog). With knowledge of HTML these style sheets can be adapted as wished.

User Manager

The User Manager administrates database users and their roles. Here individual users can be allocated database and server access. The User Manager applies to the database server and not the individual database (please refer to database security and Server security ISC4.GDB / SECURITY.FDB for further information).

To start the User Manager select the IBExpert Tools / User Manager menu item, or click the relevant icon in the Tools toolbar. The *User Manager* Editor displays all databases (drop-down list) on the current connected InterBase/Firebird server. The server connection may be altered using the pull-down list.



Select the database and server (local or remote) to administrate.

User rights for the database

All users must be logged in, in order to access the server. What they are actually allowed to do on the server is then determined using the InterBase/Firebird GRANT and REVOKE commands (see Grant Manager for further details), or the front-end program.

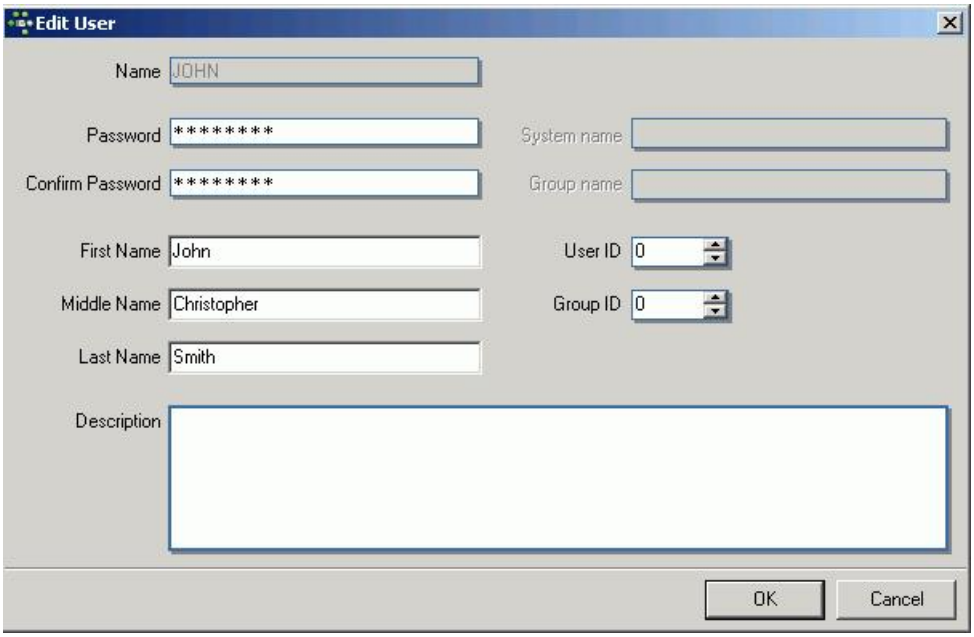
Please note! To create, edit and delete users and roles you should have the rights of server administrator.

Users page

On the *Users* page, a full list of users registered for the named server connection is displayed. Even if the selected database is not currently connected, the user list can still be seen. This is because the users are registered directly in the security database on the server, and can therefore be granted rights for all databases on this server. Since version 2.5.0.61 there is the additional column *AC* (*Active Users*) displayed in the users list. It shows how many active connections a user has to the specified database. This works only with active databases. And since version 2005.02.12.1 the *Refresh* button has been added to refresh a list of all users.

You may be asked for a password, when selecting an unconnected database, in order to ascertain your authority.

A user can be added by the SYSDBA (not a database owner, as users are created for all databases on the server). Simply click the *Add* button, and complete the *NewUser* form:



Support for the InterBase 7.5 embedded user authentication was added in IBExpert version 2004.12.12.1.

Again, only the SYSDBA or is allowed to edit or delete users. When editing, only the user name used for logging in may not be changed. It is here that a new password may be entered, if the user has forgotten his old one; or a change of name be input, for example, if a user marries.

This list contains currently existing users. To add, edit or delete users click buttons at the right of the list. In the *Add / Edit User* window set the user name and password and (optionally) his first, middle and last name.

Password

The password is always user-oriented. Passwords are stored encrypted in the server database. When a user enters his password, this is passed onto the server, which compares the string entered with the string of the encrypted password stored on the server. The password is *NEVER* passed on from the server to the client.

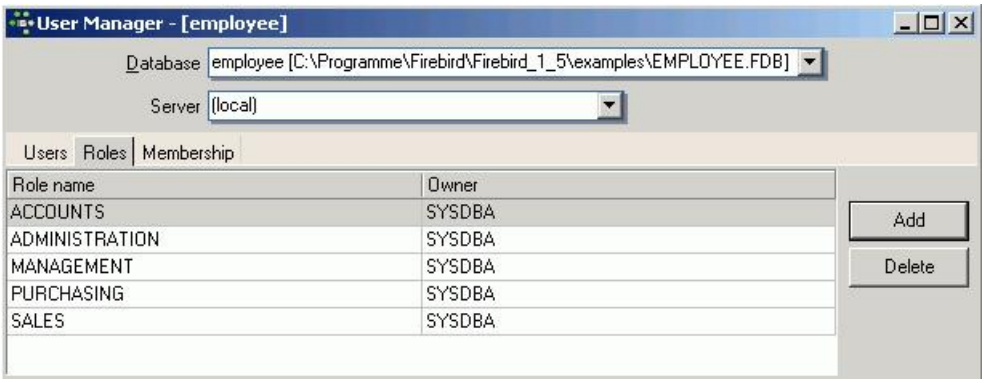
If a user forgets his password, the SYSDBA can enter a new one to replace the old one. Alternatively a UDF can be incorporated into the program, to allow the user to change his password himself, without having to disturb the SYSDBA or reveal the new password to a third person.

An example of such a UDF can be found in the `FreeUDFlib.dll`, which can be downloaded from http://www.ibexpert.com/download/udf/.

Users can be entered and assigned rights directly, although it often makes more sense if the majority of users are assigned user rights using roles. Roles are used to assign groups of people the same rights. When changes need to be made, only the role needs to be altered and each user individually.

Roles page

The *Roles* page can be used to create and delete roles exactly in the same way as with the database object roles. All roles and their owners are displayed for the selected database. Other databases on the same server may be selected to display their full range of existing roles.

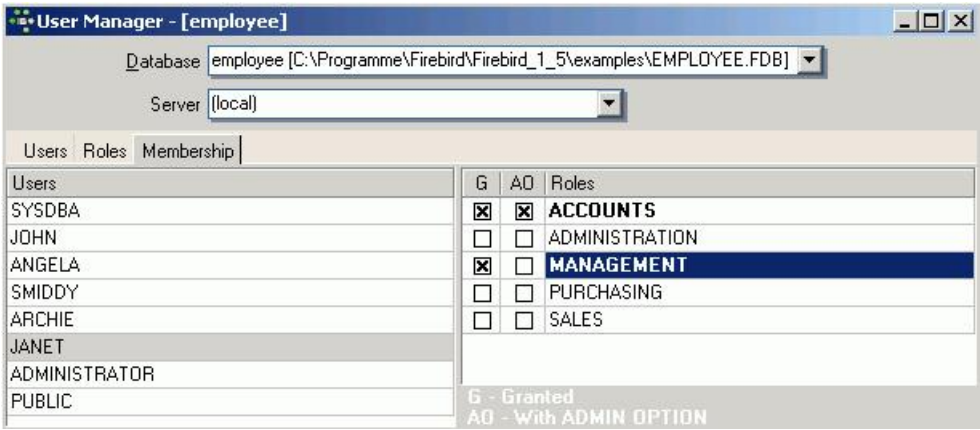


This list contains existing roles. To add or delete roles click buttons at the right of the list. When creating or deleting a role the *Compile* Window appears. Commit the transaction and if it is successful the new role is created or dropped. After the role has been created, users need to be added to the role. Role users and rights can be specified, edited and deleted using IBExpert's Grant Manager.

Roles can only be altered at system table level. They can however be deleted and new roles added using the User Manager.

Membership page

The *Membership* page shows which users have been granted rights to which roles.



The abbreviations *G* and *AO* stand for *Granted* and *With Admin Option*. Users can be assigned roles simply by selecting the user, and checking either the *Grant* boxes or the *Admin Option* boxes. For example, all sales staff could be given the user name `SALES` with the role `SALES`. When logging into the system, both these names need to be entered. Checking the *Admin Option* automatically entitles the user to pass his rights on to other users.

Server security `ISC4.GDB` / `SECURITY.FDB`

When InterBase/Firebird is installed on a server, a database of authorized users is also installed. This is vital for server security, to protect the server from being accessed, manipulated or damaged by unauthorized users.

The database's security database is called `ISC4.GDB`; since Firebird 1.5 `SECURITY.FDB`, the change of suffix being due to Windows XP's eternal copying problems with `.GDB` files. The `SECURITY.FDB` was renamed `SECURITY2.FDB` in Firebird 2.0 (Please refer to Server security `SECURITY2.FDB` below for details of main changes).

The `ISC4.GDB` provides a user page detailing rights for the InterBase/Firebird server. Here all users are entered, that are allowed to use the server. The user password is server-oriented and not database-oriented. It is important to employ users and rights to limit access and control manipulation, and is particularly advantageous, for example, to trace who has done what and when, as user names are included in the log.

Any user listed in the server security database's user list can open a database by providing the appropriate user name and password. If a user name and password is specified when the database is created, this user becomes the database owner. Only the SYSDBA and database owner are allowed to drop the database. If no database owner is specified at the time of database creation, then only the SYSDBA is authorized to drop the database.

If a user creates a table, InterBase/Firebird appoints that user as the table owner, and only the table owner and the SYSDBA are authorized to drop the table.

The SYSDBA and database owner can `GRANT`, `REVOKE` and grant access rights to users in the database; the SYSDBA and table owner can `GRANT`, `REVOKE` and grant access rights for tables. These rules also apply to views and stored procedures.

Simply allowing users into the database is not particularly helpful if they have not been granted access to the objects in this database. Therefore server security is administrated in IBExpert using the User Manager; user rights can then be assigned and controlled using the IBExpert Grant Manager.

Further security features include the following:

1. **Views:** as they can be used to hide many table details from users; the users only have access to those columns and rows that they really need to see.
2. **Referential integrity:** protects the data against orphaned rows and other operations, which could possibly damage the database integrity.
3. **`GRANT` and `REVOKE` statements:** can be used in the IBExpert Grant Manager to specify which users may access which tables and views, and whether they are also allowed to manipulate data.
4. **An object may not be dropped if it is referenced elsewhere in the database**. For example, a table cannot be dropped if it is referenced in a view, check constraint, trigger, stored procedure or other object.

Server security `SECURITY2.FDB`

The new security database is renamed as `security2.fdb`. Inside, the user authentication table, where user names and passwords are stored, is now called `RDB$USERS`. There is no longer a table named `"users"` but a new view over `RDB$USERS` that is named `"USERS"`. Through this view, users can change their passwords.

For instructions on updating previous security databases, refer to the section Dealing with the new security database at the end of this section.

The following is a summary of the major changes, the details of which can be found in the Firebird 2.0.4 Release Notes in the Security in Firebird 2 chapter:

- Better password encryption
- Users can modify their own passwords
- Non-server access to security database is rejected
- Active protection from brute-force attack

- Vulnerabilities have been closed

Classic Server on POSIX

The main reason to restrict direct access to the security database was to protect it from access by old versions of client software. Fortuitously, it also minimizes the exposure of the embedded Classic on POSIX at the same time, since it is quite unlikely that the combination of an old client and the new server would be present on the production box.

Caution: However, the level of Firebird security is still not satisfactory in one serious respect: an important security problem with Firebird still remains unresolved: the transmission of poorly encrypted passwords "in clear" across the network. It is not possible to resolve this problem without breaking old clients.

The immediate problem can be solved easily by using any IP-tunneling software (such as ZeBeDee) to move data to and from a Firebird server, for both 1.5 and 2.0. It remains the recommended way to access your remote Firebird server across the Internet.

Dealing with the new security database

If you try to put a pre-Firebird 2 security database, `security.fdb` or a renamed [[#ServerSecISC4 | `isc4.gdb`], into Firebird's new home directory and then try to connect to the server, you will get the message "Cannot attach to password database". It is not a bug: it is by design. A security database from an earlier Firebird version cannot be used directly in Firebird 2.0 or higher.

In order to be able to use an old security database, it is necessary to run the upgrade script `security_database.sql`, that is in the `../upgrade` sub-directory of your Firebird server installation, or in the Appendix to Firebird 2 Release Notes to these notes: Security Upgrade Script.

Doing the security database upgrade

To do the upgrade, follow these steps:

1. Put your old security database in some place known to you, but not in Firebird's new home directory. Keep a copy available at all times!
2. Start Firebird 2, using its new, native `security2.fdb`.
3. Convert your old security database to ODS11 (i.e. backup and restore it using Firebird 2.0). Without this step, running the `security_database.sql` script will fail!
4. Connect the restored security database as SYSDBA and run the script.
5. Stop the Firebird service.
6. Copy the upgraded database to the Firebird 2 home directory as `security2.fdb`.
7. Restart Firebird.

Now you should be able to connect to the Firebird 2 server using your old logins and passwords.

Nullability of `RDB$PASSWD`

In pre-2.0 versions of Firebird it was possible to have a user with `NULL` password. From v.2.0 onward, the `RDB$PASSWD` field in the security database is constrained as `NOT NULL`.

However, to avoid exceptions during the upgrade process, the field is created as nullable by the upgrade script. If you are really sure you have no empty passwords in the security database, you may modify the script yourself. For example, you may edit the line:

```
RDB$PASSWD RDB$PASSWD,
```

to be

```
RDB$PASSWD RDB$PASSWD NOT NULL,
```

Caution with `LegacyHash`

As long as you configure `LegacyHash = 1` in `firebird.conf`, Firebird's security does not work completely. To set this right, it is necessary to do as follows:

1. Change the SYSDBA password.
2. Have the users change their passwords (in 2.0 each user can change his or her own password).
3. Set `LegacyHash` back to default value of `0`, or comment it out.
4. Stop and restart Firebird for the configuration change to take effect.

Source: Firebird 2.0.4 Release Notes

Change user password per batch

To alter a user's password at command-line level, use the following syntax:

```
gsec -modify SYSDBA -pw password
```

or:

```
gsec -user SYSDBA -password oldpassword -modify SYSDBA -pw newpassword
```

An example for a batch:

```
set isc_user=sysdba
set isc_password=masterke
gsec -add username -pw password
```

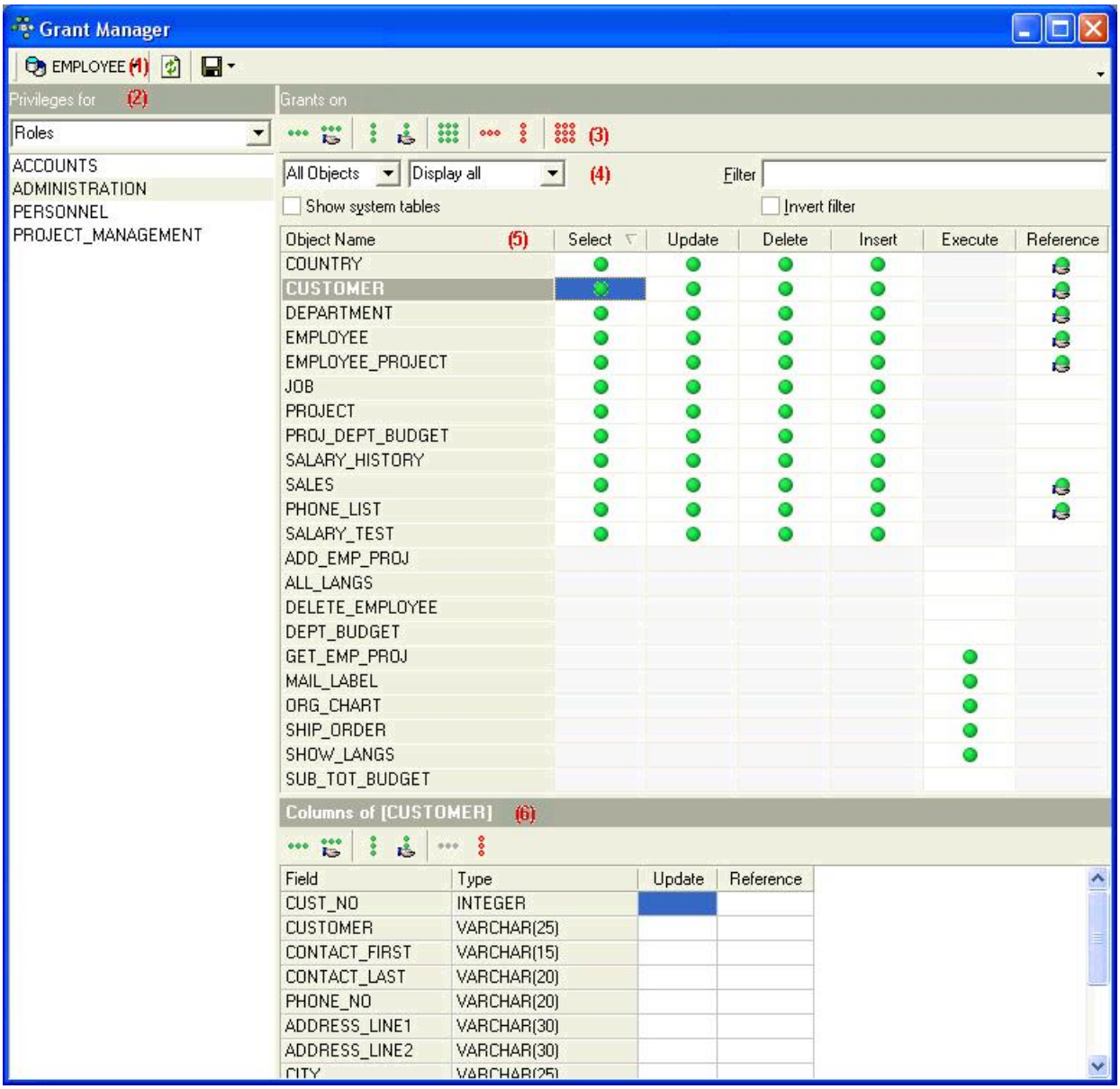
See also:
GSEC
Referential Integrity

Grant Manager

The Grant Manager is used to administrate database security by controlling user permissions for a specific database. It allows you to specify the access rights for users, roles and database objects.

To start the Grant Manager select the IBExpert menu item, Tools / Grant Manager, use the respective icon in the Tools toolbar, or double-click on a role in the DB Explorer. Alternatively use the DB Explorer's right mouse-click menu item *Edit Role* or key combination [Ctrl + O]. This feature is unfortunately not included in the IBExpert Personal Edition.

The *Grant Manager* Editor appears:



(1) Select Database: The toolbar displays the alias name for the current selected connected database. Another database on this server can be selected from the drop-down list at the top of the window.

(2) Privileges for: The pull-down list (default = Users) allows a group for the processing of privileges to be selected. The options include:

- users
- roles
- views
- triggers
- procedures

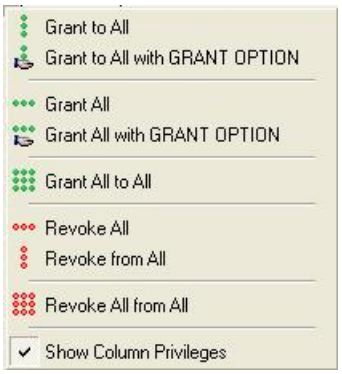
Once a database object has been selected, a full list of such users/objects in this database is displayed in the panel directly below.

(3) Grants toolbar: The Grants toolbar enables the user to quickly assign or revoke rights to one or more objects, or for one or more operations. These can also be found in the right-click pop-up menu (see below).

(4) Filters: It is possible, using the pull-down lists, to specify exactly which grants should be displayed, i.e. for all database objects (default), just the tables, just the views or just the procedures. Furthermore the user can determine whether all of the selected objects should be displayed, or only those with grants, or only those not granted. To the right of these pull-down lists is an empty filter field for user-defined filters. It is also possible to specify whether system tables should be included or the user-defined filter inverted, using the check boxes provided.

(5) The main window displays the object grants in a grid, displaying the granted operations *Select*, *Update*, *Delete*, *Insert*, *Execute* and *Reference*) for the listed objects. A green circle indicates that access for this operation on this database object has been granted; a green circle held by a hand indicates that the `GRANT WITH GRANT AUTHORITY` option has been granted. An empty field indicates logically that either no rights have been granted, or they have been revoked.

The right-click pop-up menu offers the various `GRANT` and `REVOKE` options also displayed in the Grant Manager toolbar.



A further menu option here is *ShowColumn Privileges* (checkbox). This blends the lower window in and out **(6)**, which displays the individual columns for tables and views, allowing *Update* and *Reference* rights to be granted and revoked for individual fields in the selected object.

Rights can be simply granted and revoked by double-clicking (or using the space bar) on the grid fields (in both the upper (object) and lower (column) windows). Alternatively, to assign several rights (i.e. select, update, delete and insert) to a single object or to assign one operative right to all objects displayed, use either the Grant Manager toolbar or the right-click menu.

Please note that *Reference* rights only allow the user to read data sets, if there is a foreign key relationship to other data. And the *Grant All to All* command may only be performed by the database owner or the SYSDBA.

The majority of these operations can also be performed in the *Grants* pages, found in the individual database object editors. These were introduced to remind the developer not to forget the assignment of rights! They allow the developer to check existing rights for the object concerned and, if necessary, subsequently assign rights for a new or existing object.

Rights are however in practice usually administered at the front end. There is, as a rule, only one system user, with which the program can log into the database. For those preferring direct SQL input, please refer to `GRANT` and `REVOKE`.

Granting access to stored procedures

To grant a user the right to execute stored procedures, use the IBExpert Grant Manager `EXECUTE` column:

or the SQL `EXECUTE` statement. For example, to grant Janet and John the right to execute the stored procedure `SP_Delete_Employee`, use the following:

```
GRANT EXECUTE
ON PROCEDURE SP_Delete_Employee
TO Janet, John;
```

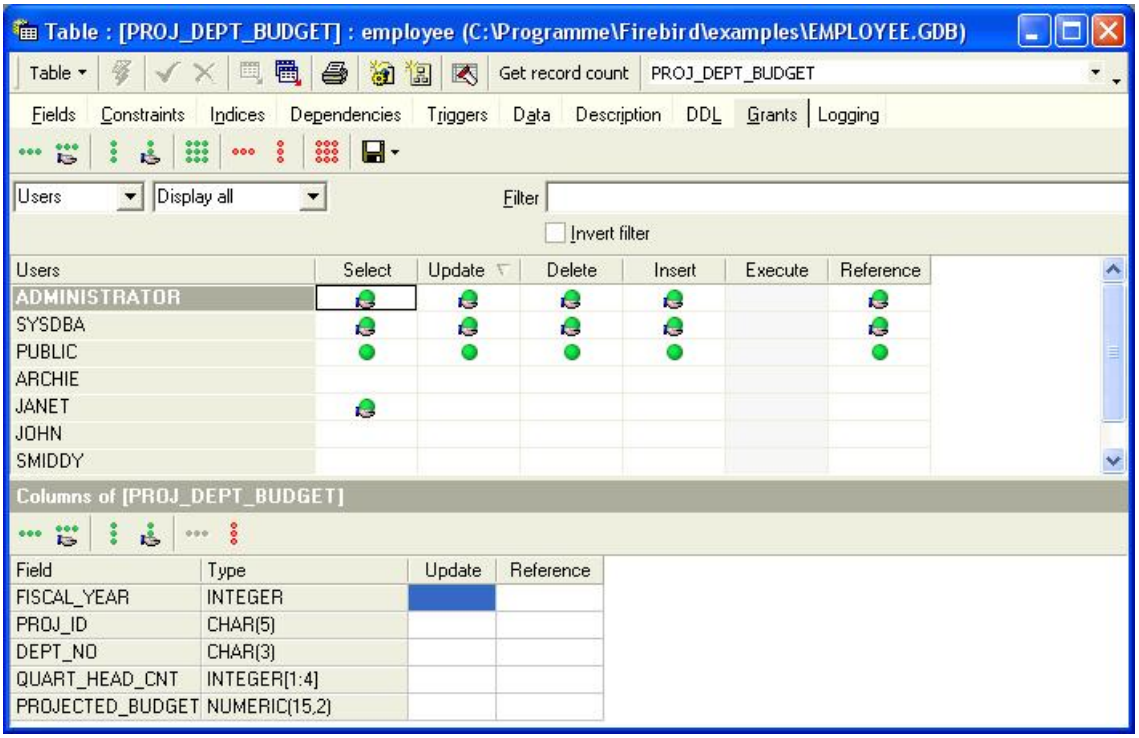
InterBase/Firebird considers stored procedures as virtual users of the database. If a stored procedure modifies a table, the procedure needs the relevant privileges on that table. So the user only needs `EXECUTE` privileges on the procedure and not any separate rights for the table. In this situation, the stored procedure performs the changes on behalf of the user.

If a stored procedure needs the ability to execute another stored procedure, simply select *Procedures* from the *Privileges For* list and *Procedures* from the *Grants On* list, to grant the `EXECUTE` privilege on the desired procedure. Using SQL the `GRANT` statement is necessary, naming the procedure instead of one or more users (`<user_list>`).

Using the `GRANT AUTHORITY` option

A user, that has been granted certain privileges, may also be assigned the authority to grant those privileges in turn to other users. This is known as assigning grant authority. InterBase/Firebird allows by default only the creator of a table and the SYSDBA to grant additional privileges onto other users.

Grant authority can be assigned in the IBExpert or the *Grants* pages in the relevant object editors, using the *Grant All with GRANT OPTION* or the *Grant to All with GRANT OPTION* icons or right-click menu items:



It is also simple to see which grant authorities have already been assigned to which users and roles.

In SQL the `WITH GRANT OPTION` clause may be used in conjunction with a grant of privileges, to assign users the authority to grant their privileges in turn to other users (refer to `GRANT` statement for the full syntax and examples).

See also:
`GSEC`
Server security `ISC4.GDB` / `SECURITY.FDB`
Table Editor / Grants page
`REVOKE ADMIN OPTION FROM`

Secondary Files Manager

The Secondary Files Manager can be found in the IBExpert Tools menu.

First select the database for which the secondary files are to be created, from the pull-down list of connected databases.

Then simply click on the *New File* button (bottom left corner) to specify a secondary file. As a database file is being created here, it is important not to forget to also specify the drive and path, as well as the file name and suffix (usually `.GDB`). Otherwise the file will be created and stored anywhere on the system (usually in the Windows System32 folder). Should this happen, the file drive and path can be viewed when the Secondary Files Manager is restarted.

After specifying the secondary file's name, either the starting page (*File Start*) or length in pages (*File Length*) can be specified by selecting the field, and clicking or using the space bar to activate the counter or allow numerical entry. Specifying both these parameters is unnecessary, and only provides an error source, as the starting pages of two files must of course concur with the number of pages of the first file.

When using the IBExpert Secondary Files Manager, the first secondary file starts at the current position in the primary file, i.e. the primary file is immediately considered to be "full", and all new data and metadata from this point onwards is stored in this first secondary file. This can be viewed in the IBExpert Services menu item, Database Statistics. See below for the specification of the primary file size at the time of database creation. Of course, multiple secondary files may be specified here if wished. It is not necessary to specify the length of the last secondary file; this can therefore become as large as the physical disk space allows.

When all files have been specified satisfactorily, simply click the *Apply* button,

and check before finally committing or rolling back.

There are no performance advantages to be expected by distributing the database across several files, so it is not recommended that secondary files be used, unless the disk storage space and database size absolutely require it.

The secondary files' size, path and name can only be altered when the database is restored, as this is the only option which allows secondary files to be redefined.

For those preferring direct SQL input the syntax is as follows:

```
CREATE DATABASE "database name"
LENGTH <number > PAGES
FILE <secondary file 1> LENGTH <number> PAGES
FILE <secondary file 2> LENGTH <number> PAGES
...
FILE <secondary file N>;
```

The alternative syntax, using `STARTING (AT PAGE)`, is as follows:

```
CREATE DATABASE "database name"
FILE <secondary file 1> STARTING AT PAGE <number>
FILE <secondary file 2> STARTING AT PAGE <number>
...
FILE <secondary file N> STARTING AT PAGE <number>;
```

The `AT` and `PAGE` keywords are optional. InterBase/Firebird recognizes any of the following variations:

```
STARTING AT PAGE 5000
STARTING AT 5000
STARTING 5000
```

Please note that when a database is dropped/deleted, all secondary and shadow files are also deleted. The complete structure and all the data is permanently deleted!

Primary file

A database's primary file is the main database file. If no secondary files are specified, it is the only database file.

When secondary files are used, the length in pages needs to be specified for the primary file, or alternatively the first secondary file needs to be specifies with the `STARTING (AT PAGE)` parameter.

Primary and secondary files can be specified in the IBExpert Tools menu item, Secondary Files Manager.

Secondary files

One or more secondary files may be specified by the database creator, to be used for database storage once the primary file has reached its specified limit. The database can be distributed across as many secondary files as wished.

Usually InterBase/Firebird databases grow dynamically, when database objects, program code or data are added. The only limitations are the physical limits of the hard disk or file system on which the database is stored.

Some file systems such as, for example, HP UNIX have additional limitations which do not enable the partition size to go over two Gigabytes. To avoid such a limitation, the InterBase database can be spanned across multiple file systems. Each file can be assigned a maximum size. Due to the automatic administration in InterBase/Firebird, the primary file is first filled until the maximum page size has been reached. Subsequent information is then packed into the secondary files until their capacity has been reached. As many secondary files can be created as wished.

Since InterBase 6.5/Firebird secondary files are really no longer necessary. In those particular cases, where secondary files may need to be considered, please consult the respective database Release Notes.

There are no performance advantages to be expected by distributing the database across several files, so it is not recommended that secondary files be used, unless the disk storage space and database size absolutely require it.

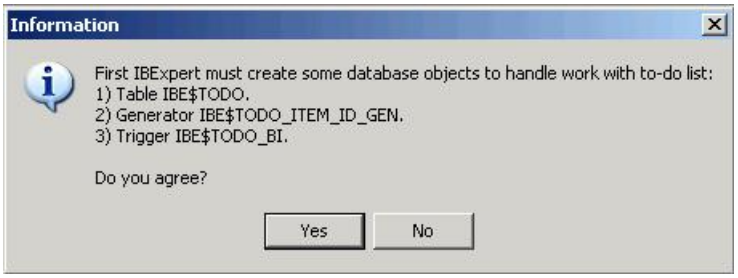
Secondary files can be simply and easily created using the IBExpert Tools menu item, Secondary Files Manager.

Please note that when a database is dropped/deleted, all secondary and shadow files are also deleted. The complete structure and all the data is permanently deleted!

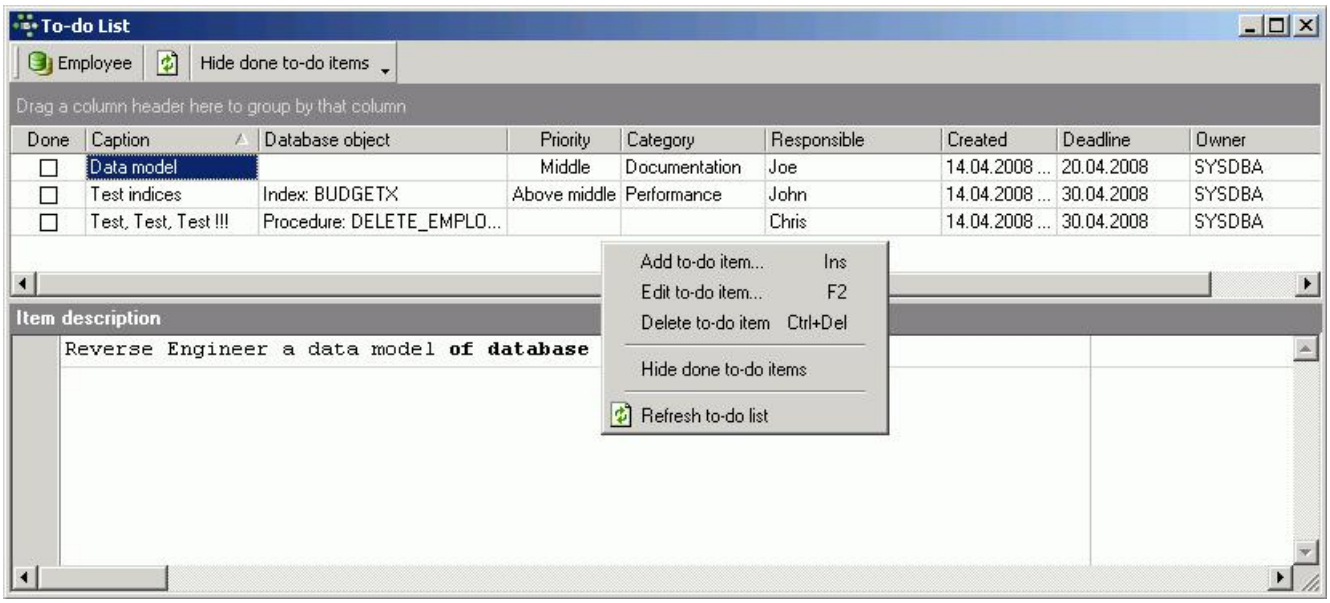
To-do list

This feature was introduced in IBExpert version 2007.12.01 and can be used to organize your database development.

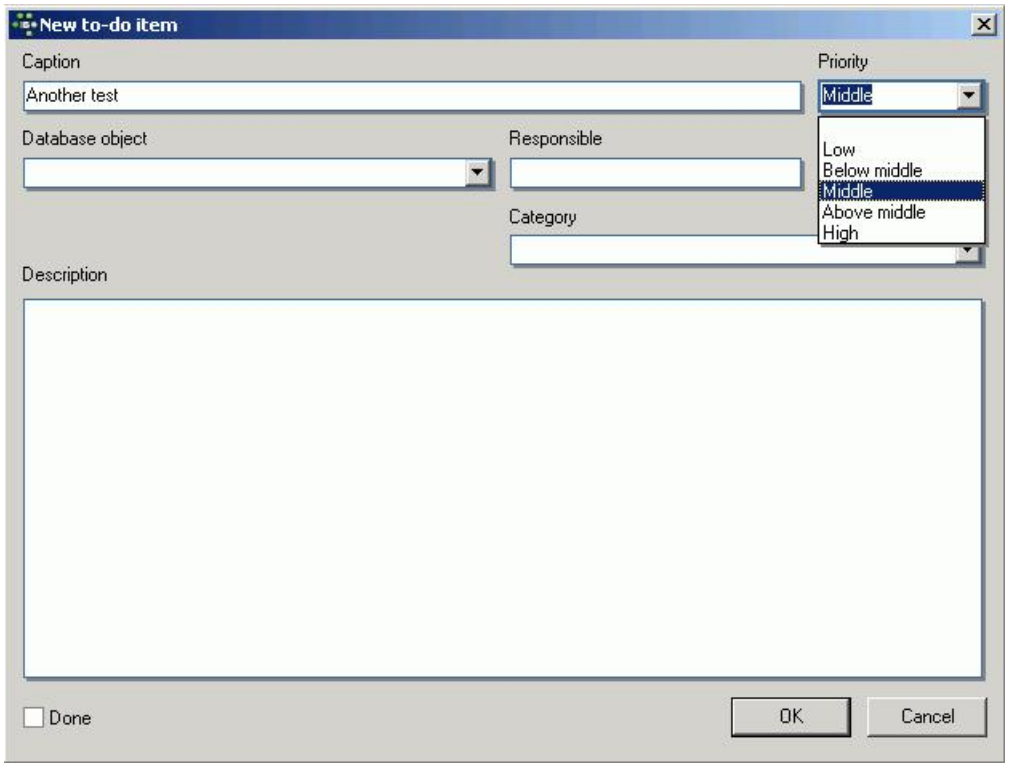
After allowing IBExpert to create the necessary system objects:



you can add to-do items for each object in the database, using the right mouse-click context-sensitive menu or the [Ins] key. This menu also allows you to *Edit a to-do item* ([F2]), *Delete a to-do item* ([Ctrl # Del]), *Hide done to-do items* (or click the icon in the toolbar) and *Refresh to-do list* (also found in the toolbar).



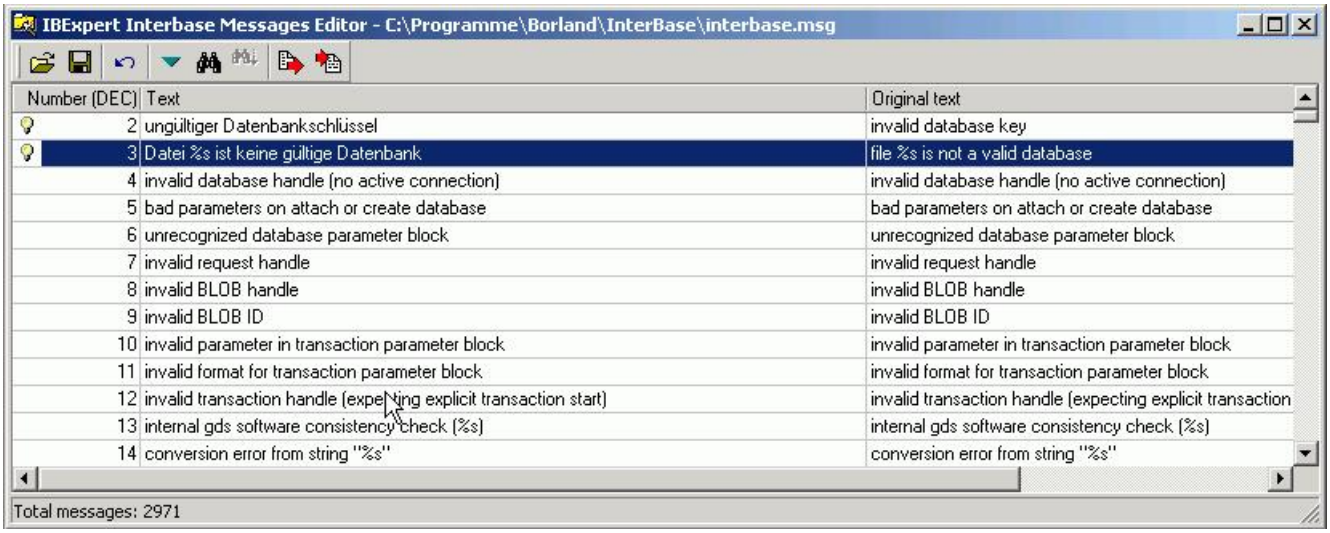
The fields in the *Newto-do item* dialog are not mandatory, but may be completed as wished. Pull-down options lists and a calendar are provided where relevant, and the *Description* field in the lower part of this window allows you to include as much information as you wish or need.



Once a to-do item has been completed, it can be checked as *Done* and, if wished, either hidden from view or deleted.

Localize IB Messages

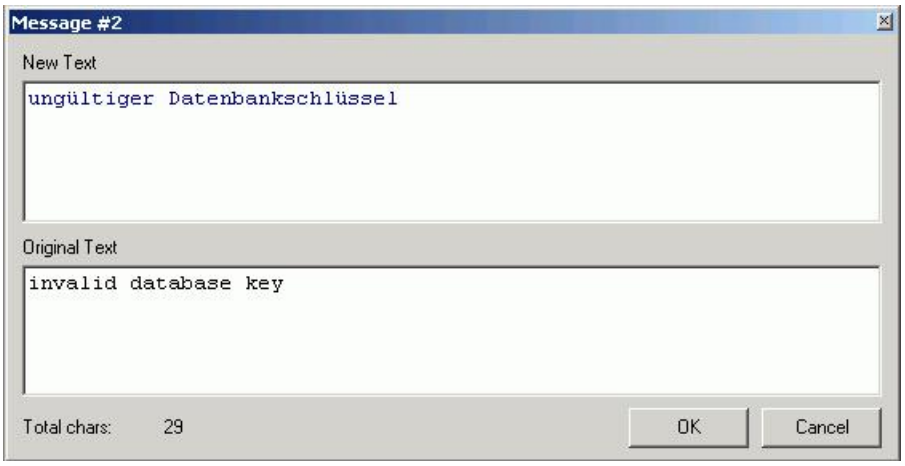
Localize IB Messages can be found in the IBExpert Tools menu. It enables the user to translate InterBase/Firebird messages into another language.



The InterBase/Firebird messages can be loaded by clicking on the *Open File* icon and specifying the drive and path (`Firebird\interbase.msg` or `InterBase\interbase.msg`).

The messages are displayed in tabular form. The first column displays the message number (the total number of messages is displayed in the status bar). The second column shows the editable text; the third column the original English text.

To translate a message, simply double-click to open the *Edit* window, enter the desired translation, confirm to return to the main window, and save (or undo). When saving it is recommended a new file name be specified, for example `interbase_german.msg`, as otherwise the original English text is overwritten by the translation.



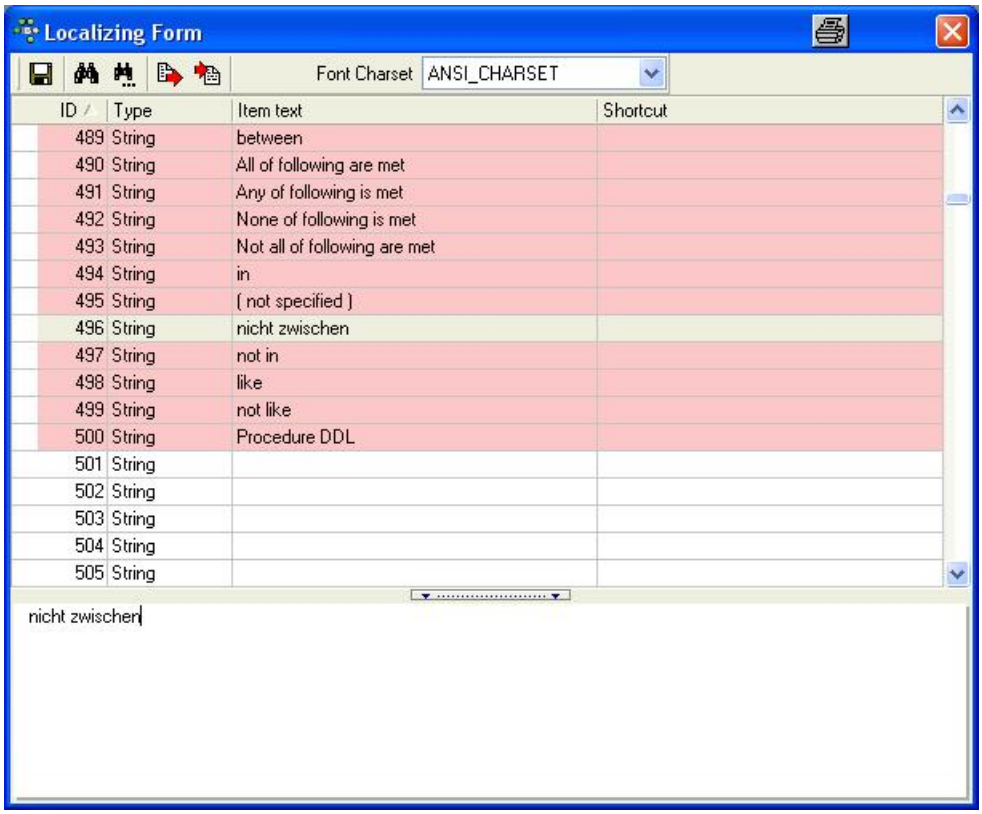
Other options offered in the Localize IB Messages toolbar include:

- **Save to File:** saves all changes to the file named in the title bar.
- **Undo:** allows the message text to be reverted to the original, provided it has not yet been saved to file.
- **Goto Message Number:** spring to specified message number.
- **Find and Search Again:** search options for finding message texts.
- **Export to Text File:** enables the message list to be exported to a text file.
- **Import from Text File:** allows a message list to be loaded from a text file as opposed to loading the `interbase.msg` file).

See also:
IBExpert Edit menu
Toolbar Localize IB Messages

Localize IBExpert

Localize IBExpert can be found in the IBExpert Tools menu. It enables the user to translate InterBase/Firebird messages into another language.



The InterBase/Firebird messages are automatically loaded. An alternative *Font Character Set* may be selected if necessary from the pull-down list offered in the Localize IBExpert toolbar.

The *Localizing Form* displays all IBExpert messages in a tabular form. The first column displays the ID number (there are 2,999 ID records altogether). The second column shows the message type (e.g. string), the third the editable item text; and the fourth column the respective shortcut. Initially pink highlighted records show messages already created and assigned in the original English version. Blank rows (non-highlighted) indicate non-assigned messages.

To translate a message, simply select it, enter the desired translation in the lower editing panel and save. When saving it is recommended an new file name be specified, for example `interbase_german.msg`, as otherwise the original English text is overwritten by the translation.

Other options offered in the Localize IBExpert toolbar include:

- **Save to File:** saves all changes to the file named in the title bar.
- **Find and Search Again:** search options for finding message texts.
- **Export to Text File:** enables the message list to be exported to a text file.
- **Import from Text File:** allows a message list to be loaded from a text file (as opposed to loading the standard IBExpert original English file).

If you have succeeded in translating this file into a language that IBExpert does not yet offer, please contact info@ibexpert.biz. We would love to hear from you!

Find IBExpert Message

This *Search* dialog is useful for finding individual words or word strings in the long lists of IBExpert language translations. It can be called using the *Binocular* icon in the *Localizing Form* toolbar. The dialog offers a number of options:

The *Text to Find* field allows direct input, or the pull-down list may be used to select a text recently searched for.

The *Direction*: *forward* (default) or *backward* may be selected, as well as the area to be searched (from a *selected area* or across the *entire scope*).

Use the *OK* button to spring to the first occurrence of the text specified.

The

icon can be used to search for further occurrences, should any exist, of the specified string.

See also:
IBExpert Edit Menu
Localize IB Messages

Report Manager

Using the menu item Tools / Report Manager or the respective icon in the Tools toolbar, the Report Manager dialog is opened. (This feature is unfortunately not included in the IBExpert Personal Edition.)

A new report can be created on any volume or in the database (double-click on a database entry to create the necessary objects automatically). To edit the report, just use [Ctrl+D] and the editor will open. To create a new report, simply right-click on the `Page1` header and add a new dialog form. On this form you can add a database and one or more query components. Go back to `Page1` and insert some bands and rectangular objects. All data connections can be viewed in the *Object Inspector* or following a double click.

In IBExpert version 2005.09.25 the Report Engine was upgraded to FastReport 3. In this version all metadata reports have been redesigned and printing of unicode strings is now supported. There are many export filters available. And the integration of user reports with IBExpert registered databases has been improved. There is a sample report, `\Reports\Sample1.fr3`, provided which illustrates how to connect database access components within a report with registered databases.

Take a look at `http://www.fast-report.com/` to view some examples and the original components, which can be used in any Delphi/CBuilder project as an extremely powerful, quick and stable replacement for Quickreport and other report tools.

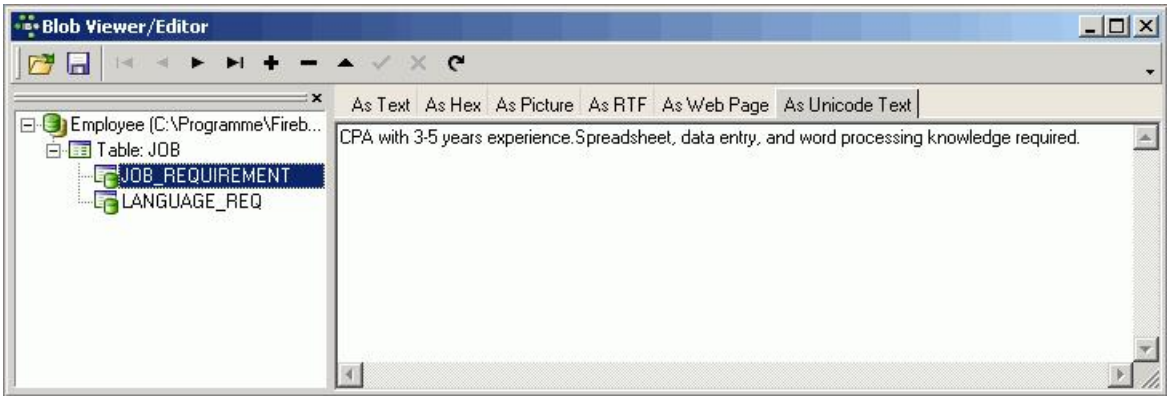
Since IBExpert version 2008.05.08 we have introduced some new IBEBlock commands for executing reports created with the IBExpert Reportmanager in command-line mode, for example with batch files. The monthly sales report, invoices or other reports can be designed in the Report Manager and executed with simple SQL statements. The result can be saved in the database as a pdf or other formats and sent by e-mail. Please refer to ibec_CreateReport and ibec_ExportReport for further information.

We personally have still not found anything that Fast Report cannot do!

See also:
Report Manager toolbar
Tools toolbar

Blob Viewer/Editor

The IBExpert Blob Viewer/Editor can be found in the IBExpert Tools menu. (This feature is unfortunately not included in the IBExpert Personal Edition.)

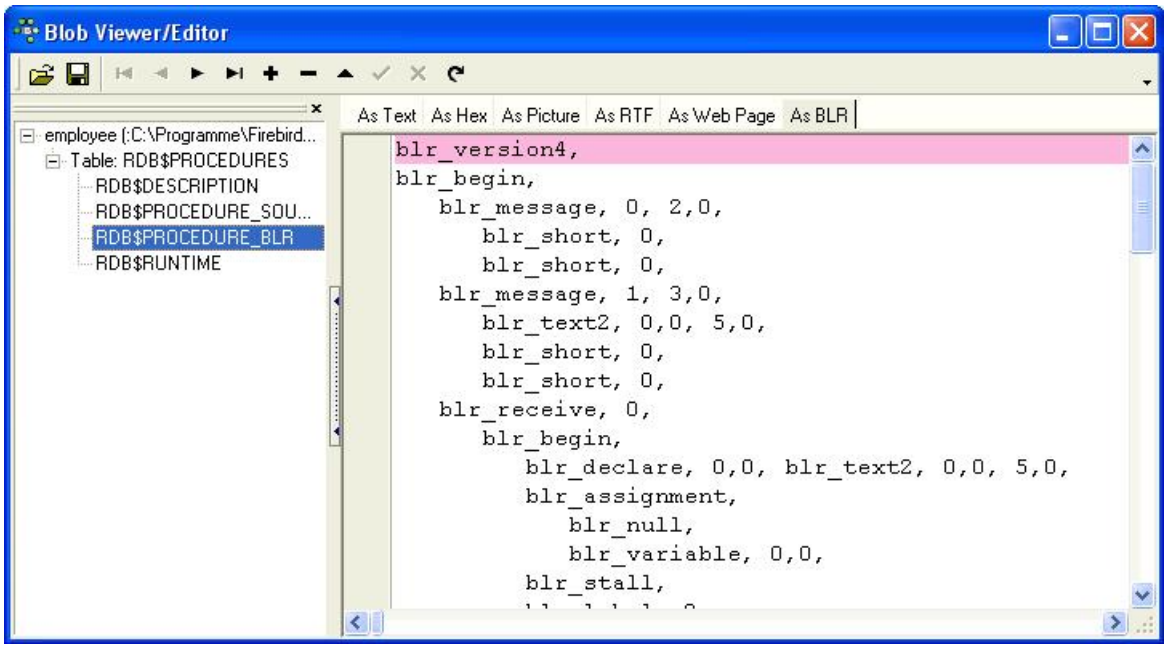


It enables blob fields in an open grid (e.g. the Table Editor / Data page, the SQL Editor / Results page) to be viewed as *Text*, *Hex*, *Picture*, *RTF* or *As Web Page* or *As Unicode Text*.

And since IBExpert version 2005.08.08 array values can also be viewed and edited here (HEX format).

The individual fields in the blob column can be viewed and navigated using the editor's navigational toolbar (please refer to Blob Viewer/Editor toolbar for details).

New to IBExpert version 2003.11.6.1 is the added syntax highlighting for SQL. This is useful if your blobs contain SQL queries. And IBExpert version 2007.09.25 introduced syntax highlighting for Delphi forms (`dfm`). Furthermore, there is now a new *As BLR* page. This allows blobs with subtype 2 data to be displayed:



This shows what is really physically in the database.

Since IBExpert version 2005.03.12 there is also added support for PNG (Portable Network Graphics) images, and IBExpert version 2008.02.19 added support for TIFF images.

See also:
Toools toolbar

Database Designer

The IBExpert Database Designer is a comprehensive tool, which allows database objects to be managed visually. It can be used to represent an existing database visually, or create a new database model, and then create a new database, based upon this model. It is possible to add, edit and drop tables and views, edit table fields, set links between tables, edit and drop procedures, and so on. This feature is unfortunately not included in the IBExpert Personal Edition.

The Database Designer can be started from the IBExpert Tools menu.

The *Designer* Menu offers the following options:

- Reverse Engineer ...
- Generate Script...
- New Diagram
- Load Diagram from File
- Save Diagram
- Export...
- Print
- Manage Subject Areas
- Manage Subject Layers
- Model Options...

There are also a number of toolbars (please refer to Database Designer toolbars for further information).

Should IBExpert not load the toolbars automatically after starting the Database Designer, delete `IBExpert.tb` from the `\Documents and Settings\<user>\Application Data\HK-Software\IBExpert\` directory and restart IBExpert.

Using the *Designer* menu items or icons, an existing diagram can be opened from file, or a new diagram created.

Reverse Engineering will be used here for the sake of demonstration. By simply creating a model of the sample `EMPLOYEE` database using the *Reverse Engineer ...* menu item, it is possible to view and test the many features the Database Designer has to offer.

The magnifying glass icons in the Menu and Palette toolbar can be used to increase or reduce the diagram size. Using the pointer icon (= normal editing mode), tables and views can be selected by clicking on them with the mouse, or dragged'n'dropped as wished; the connecting lines (= links) automatically move as well.

Insert new tables or views by simply clicking on the relevant icon in the *Palette* toolbar, and positioning in the main diagram area. Since IBExpert version 2004.10.30.1 templates can be used (IBExpert menu item Environment Options / Templates) to create foreign and constraint names automatically.

Alternatively, existing objects may be dragged and dropped from the DB Explorer and SQL Assistant into the main editing area. When an object node(s) is dragged from the DB Explorer or SQL Assistant, IBExpert will offer various versions of text to be inserted into the Code Editor. It is also possible to customize the highlighting of variables. Use Options / Editor Options / Color to choose color and font style for variables. Since IBExpert version 2007.05.03 custom colors are saved in and restored from a `grc` file. And since IBExpert version 2007.02.22 objects can also be dragged and dropped from the DB Explorer Project View tree.

IBExpert version 2004.9.12.1 introduced the Model Navigator in the SQL Assistant, enabling you to navigate models quickly. Use the corresponding page in the SQL Assistant (*Model Navigator*). The Database Explorer now offers an additional Diagrams page, displaying all objects in the database model in a tree form. Simply click on any object, and it is automatically marked for editing in the main Database Designer window.

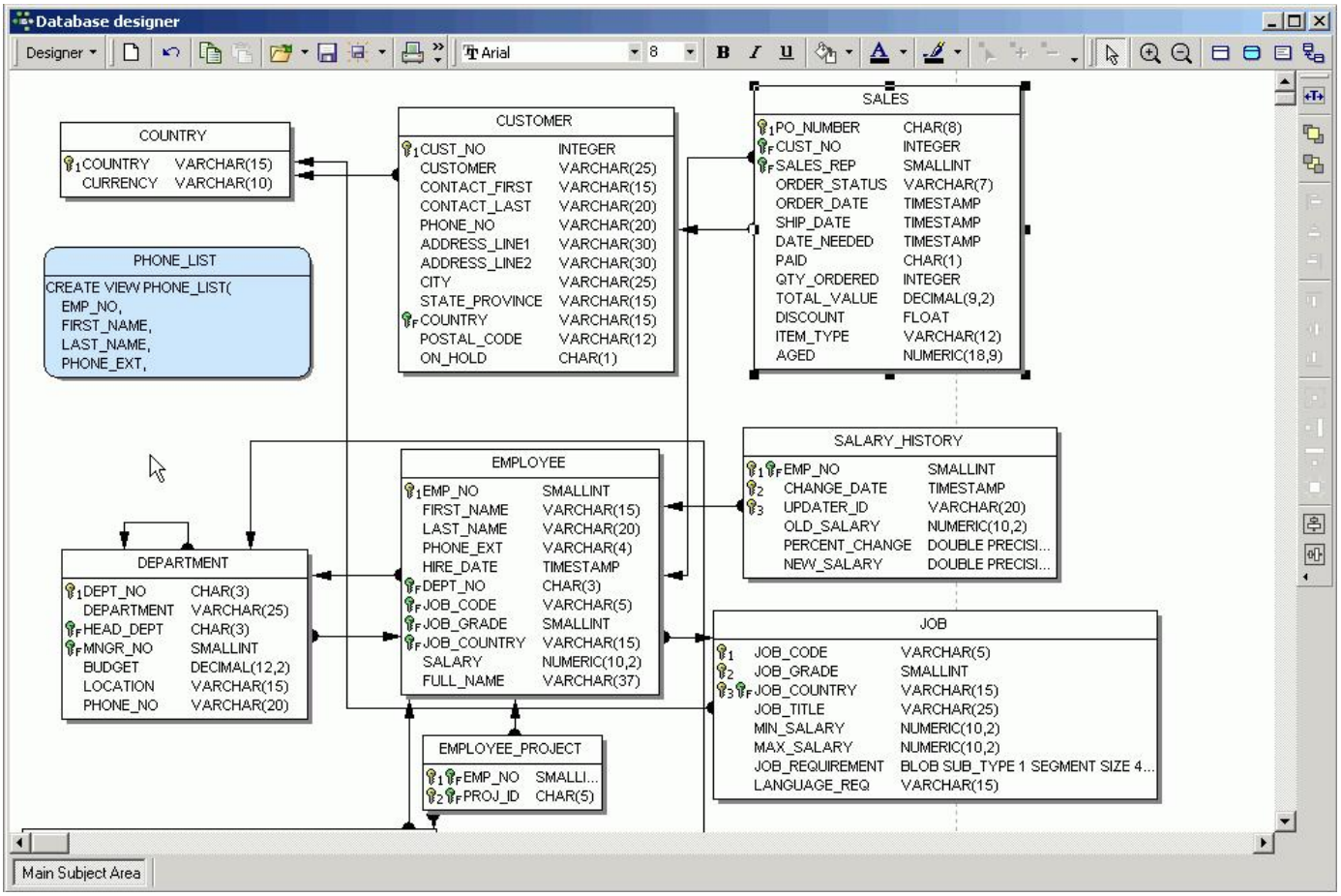
The *Comment box* icon allows comments to be added to the diagram. Insert and position a comment box, double-click to add the comment text in the Model Options window on the Database Designer *Comment Box* page.

Reference lines, i.e. foreign key relationships can be drawn between tables/views using the right-hand icon in the *Menu and Palette* toolbar, and dragging the mouse from one table to the next.

Context-sensitive right-click menus offer a number of options for selected tables, views or links (please refer to Database Designer Right-Click Menus for further information).

Double clicking on any table or view opens the Model Options menu item in the lower window, where information can be viewed, altered or specified.

By double-clicking on the line between two tables, the relationships are shown in detail. The name and automatic tracing of links are options, as already mentioned, included in Model Options.



Database objects may be grouped using the [Shift] key and selecting objects with the mouse, and then using the respective Layout toolbar icons to group or ungroup objects.

Objects can also be aligned (left, center, right, top, middle, bottom), again by holding the [Shift] key and selecting objects with the mouse, and using the respective *Layout* icons. Using these key combinations, it is also possible to select a group of objects, and make them the same size, height or width, size to grid, or center horizontally or vertically.

And since IBExpert version 2005.03.12 there is the added option, using the right-click context-sensitive menus, to lock visual objects, to protect them against casual modification of size and position.

Don't forget, the white pointer icon returns the mouse to the normal editing mode!

It is also possible to Manage Subject Areas and Manage Subject Layers.

When the database model has been designed/altered as wished, a script can be generated (please refer to Generate Script) and executed, to apply these alterations to the database itself.

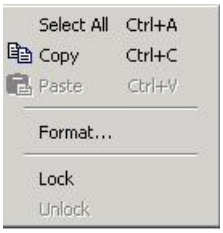
Database Designer right-click menus

The main Database Designer design area offers a selection of context-sensitive right-click menus. When a table is selected, the following options are offered:

These include options to *Select All*, *Copy* and *Paste*; *Columns*, *Indexes*, *Keys*, *Checks*, *Triggers* and *SQL Preview* are those options also offered in the *Model Options* dialog in the lower part of the screen; a check box to specify whether a selected table should be depicted with a shadow or not; and *Format*. This menu item opens a new dialog - for tables however, this only offers the shadow option, also listed as a check option in the menu.

The *Lock / Unlock* option is new to IBExpert version 2005.03.12, and allows visual objects to be locked, to protect them against casual modification of size and position.

When a view is selected, the right-click menu offers the following options:



Again the option to *Select All*, *Copy* and *Paste* is offered, along with the *Format* option. This dialog must be opened and the shadow option checked or unchecked, if the appearance of the view is to be altered.

When a link is selected, the following options are offered:

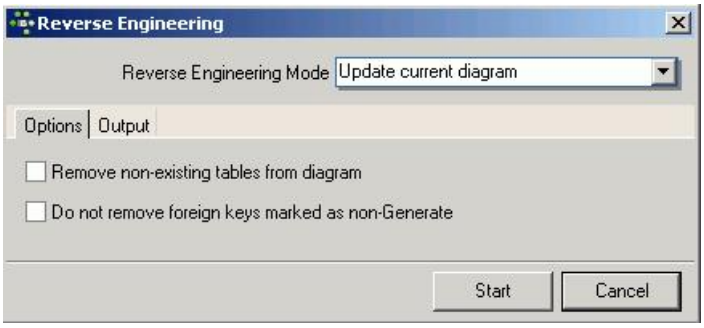
Again there is the option to *Select All*, *Copy* and *Paste*. Furthermore, it is possible to spring to either the *Parent* or *Child* (i.e. primary key table or foreign key table), and again the *Format* option opens a new dialog, where, on the *Links* page, the rounded corners option may be checked or unchecked as wished.

Reverse Engineer

Reverse engineering creates a diagram of an existing database.

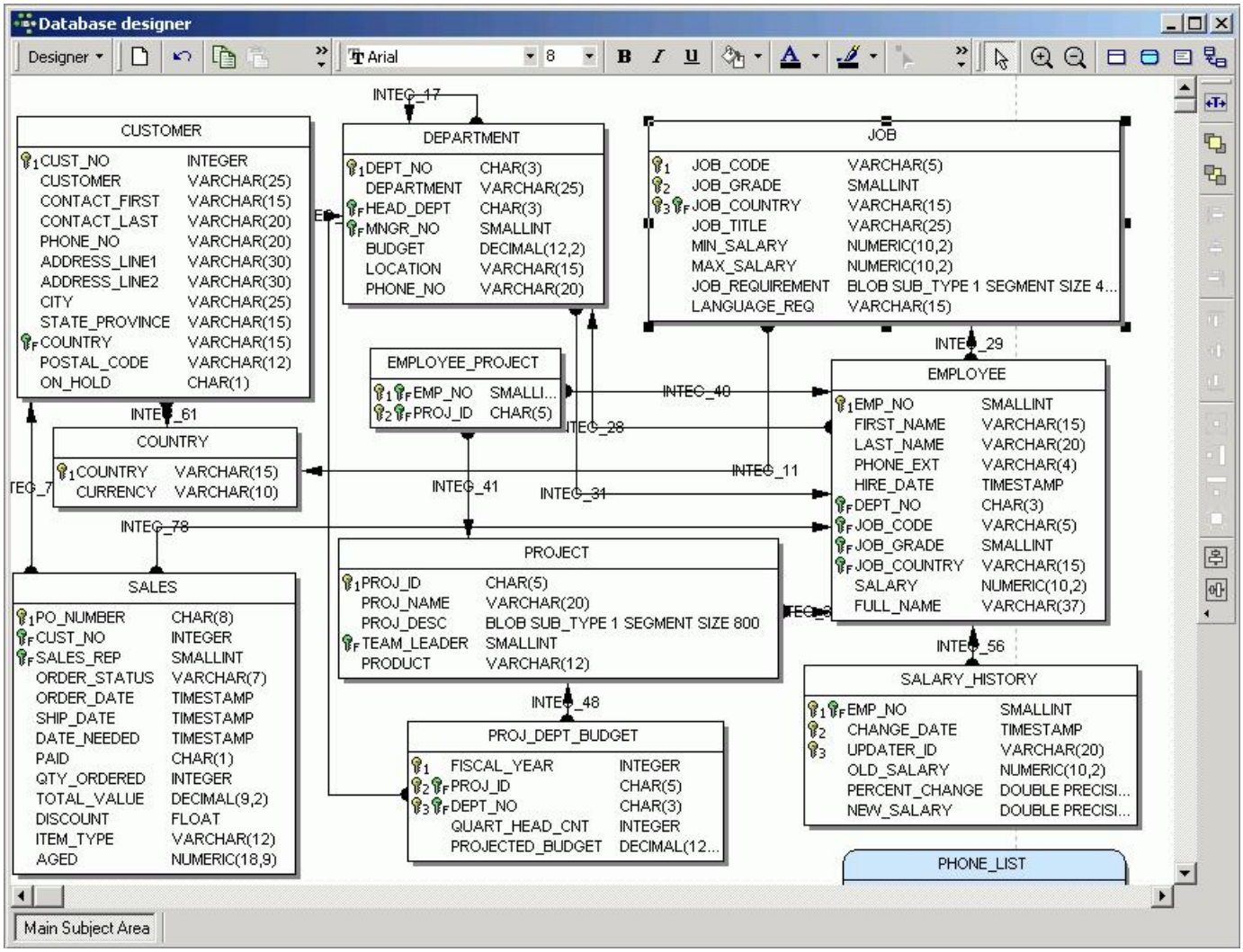
When reverse engineering, select the database to be visually displayed from the list of registered databases.

In the case of the selection of an unconnected database, IBExpert asks whether it should connect. Specify whether a new diagram should be created or an existing one updated, and check the *Clear Diagram* option if necessary:



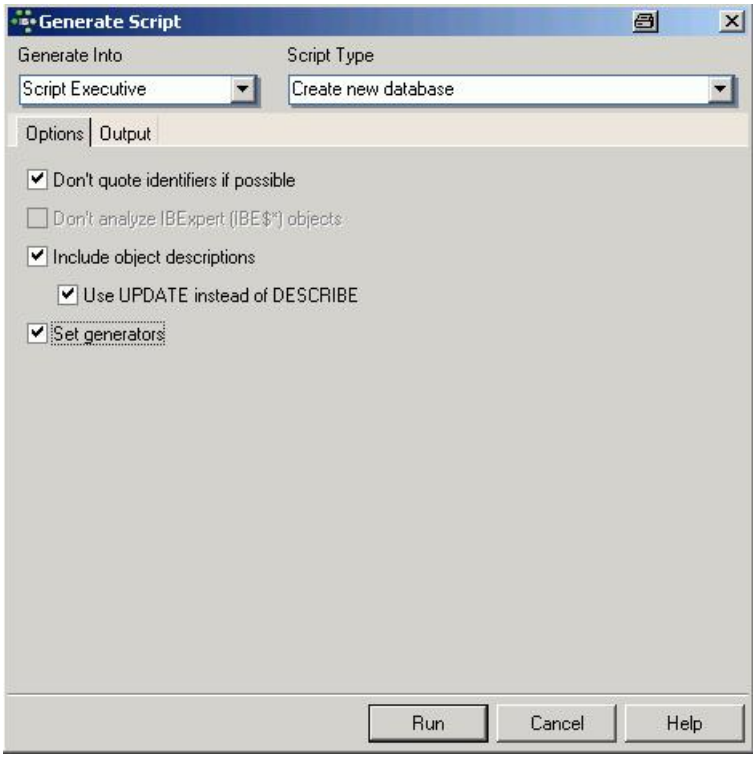
The option *Do not remove foreign keys marked as non-Generate* was added in IBExpert version 2004.12.12.1 and is useful to prevent fake relationships from being deleted.

Start the reverse engineering, and see how quickly IBExpert creates a diagram of the database:



Generate Script

It is also possible to generate a script for the model using the *Generate Script* menu item. This is necessary in order to apply any changes made to the model to the database itself.



The script can be generated into the Script Executive, to a file or to clipboard. The *Script Type* options include:

- Create new database
- Update existing database
- Difference script (for testing only)

Specify the file name if saving to file and check/uncheck the options

- Don't quote identifiers if possible
- Don't analyze IBExpert (IBE$*) objects
- Include object descriptions (and since IBExpert version 2005.04.24 also including the option *Use UPDATE instead of DESCRIBE*
- Set Generators (new to IBExpert version 2005.06.07) as wished, and run.

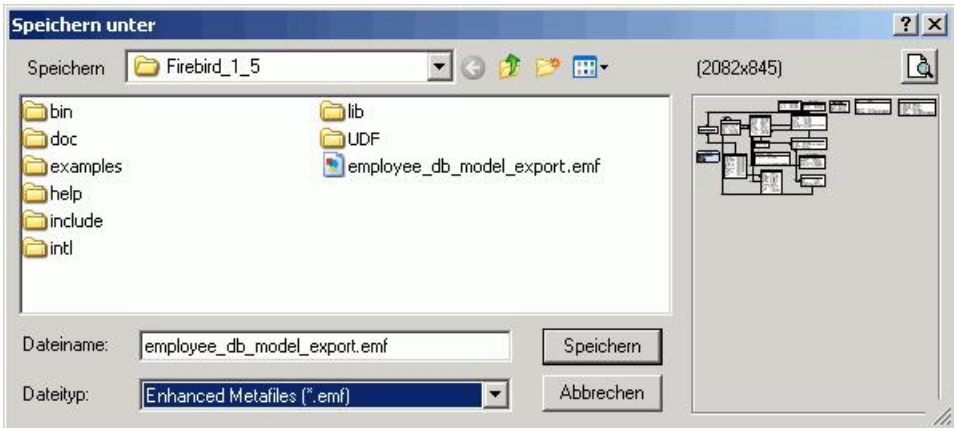
Since version IBExpert 2004.8.5.1 generators and triggers are now processed during generation of the update database script. View dependencies are also now taken into account when the script is generated.

Since IBExpert version 2004.9.12.1 the `SET NAMES`, `SET SQL DIALECT`, `CREATE DATABASE` statements were removed from the resulting `CREATE DATABASE` script. You now need to use the model prescript (*Model Options*) to specify necessary `INIT` statements.

The generation of update scripts was improved in IBExpert version 2004.12.12.1. to include analysis of exceptions and procedures.

Export

The database model can be exported, either as a bitmap (`.bmp`) or an enhanced metafile (`.emf`). This is new to IBExpert version 2.5.0.61. Simply load the model to be exported, click the *Export* menu item, and specify the name and format.

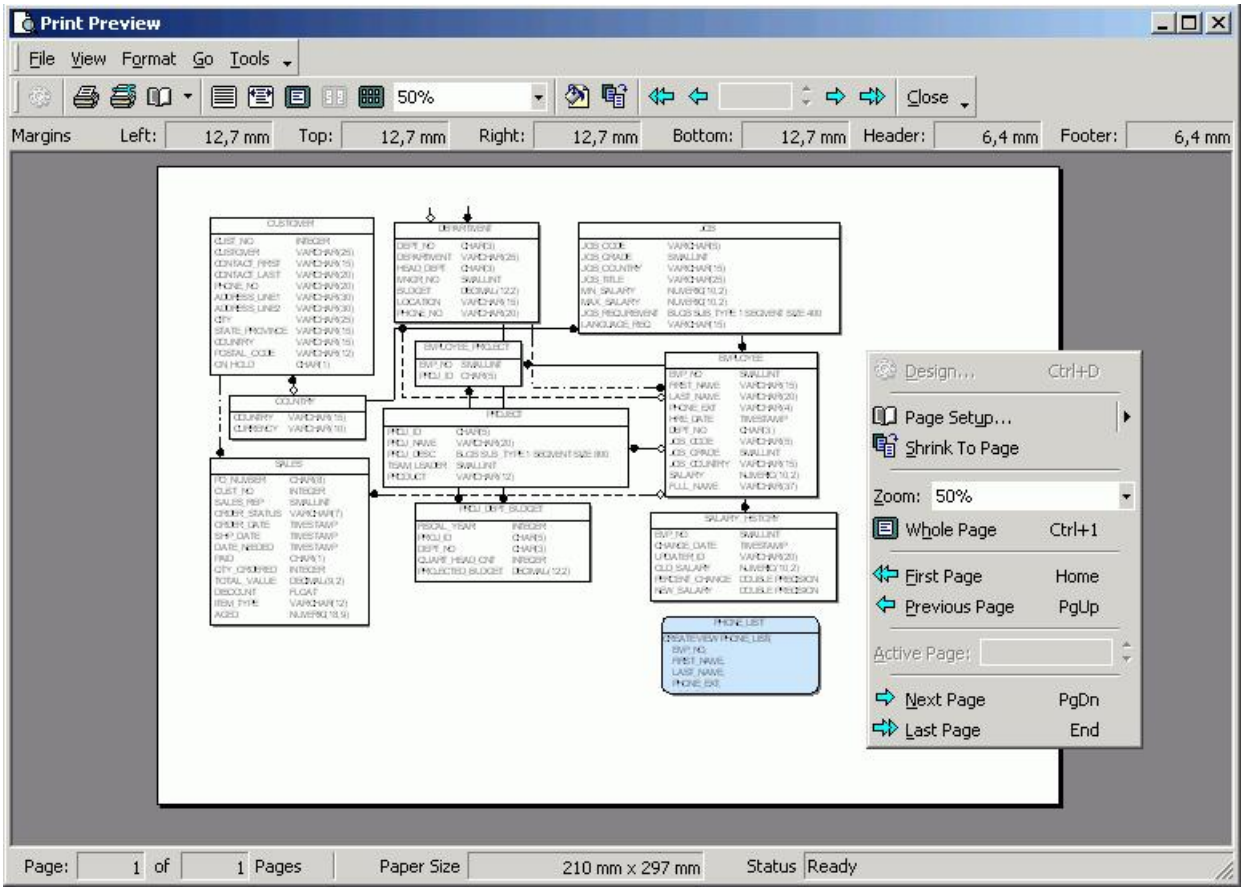


Print

The database model can be printed, using the respective Database Designer menu item or icon. This option firstly produces a print preview, allowing adjustments to be made before printing.

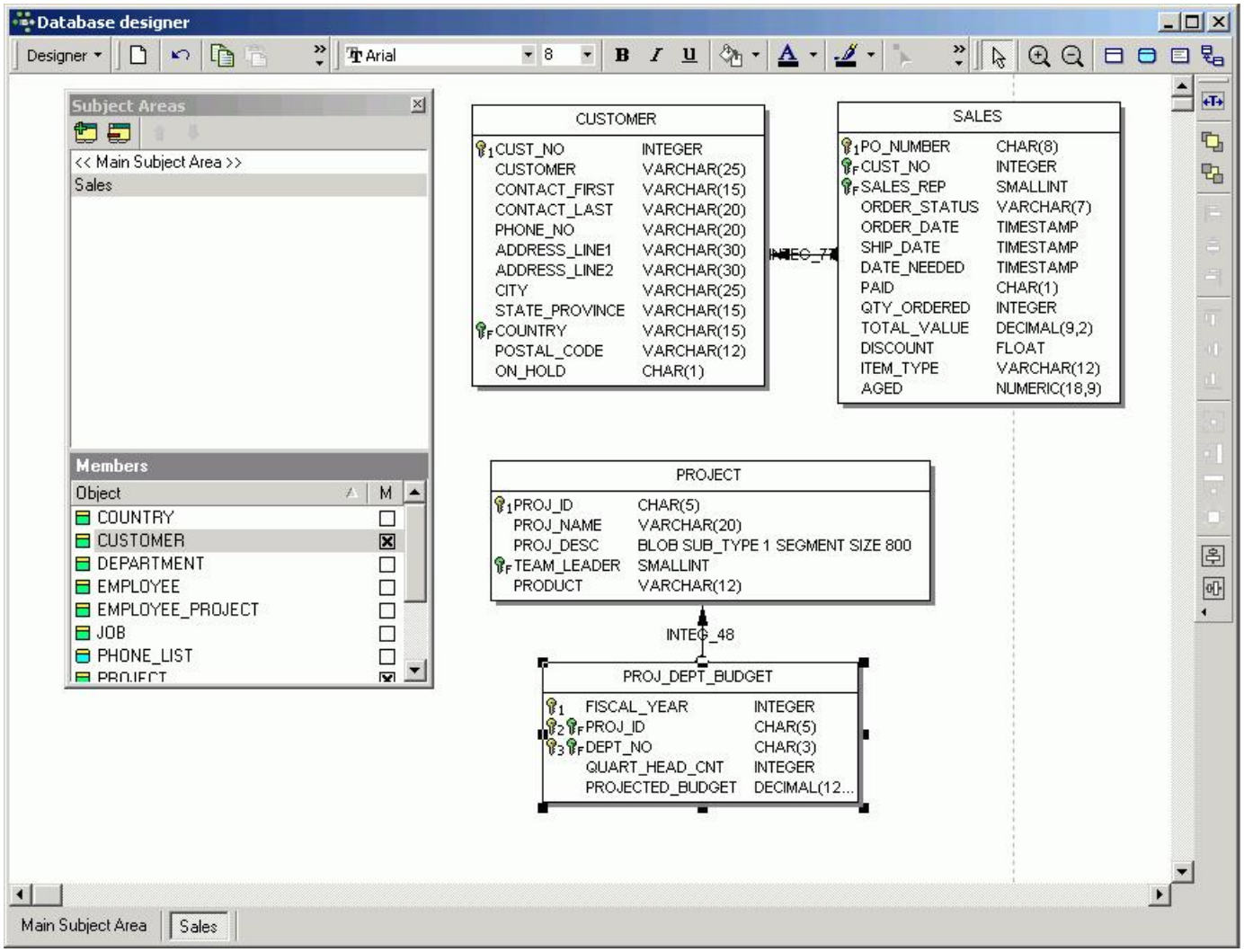
New Features

Since IBExpert version 2.5.0.61 it is possible to store printing options between sessions. Since version 2003.12.18.1 it is now possible to display borders of pages (printable parts) with dashed lines. You can customize the page options (size, headers and footers etc.) using the *Print Preview* form.



Manage Subject Areas

The IBExpert Database Designer menu item Manage Subject Areas is particularly useful, for example, to administrate or visualize certain sub-areas of the database, e.g. Sales or Administration, independently or separately from the rest of the database. Use the *Manage Subject Areas* menu item.



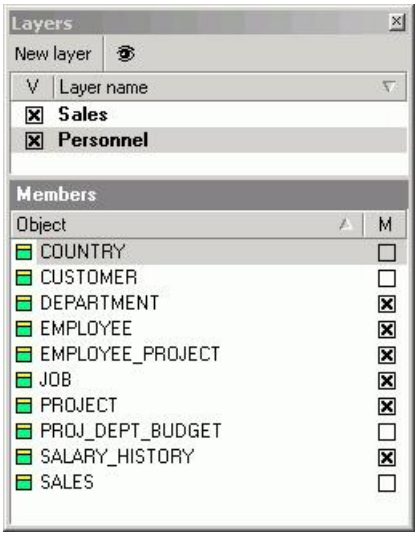
Using the two dialog icons, new subjects can be defined by entering a name and checking those tables to be included; or existing subjects altered or deleted. Since IBExpert version 2004.12.12.1 it is possible to drag 'n' drop objects from the DB Explorer (Diagrams page) to the subject areas to include them as members of this area. It is also possible to drag objects from the list of objects in the Subject Areas Manager.

Several subject areas can be opened and administrated simultaneously; switch from subject to subject by clicking on the window buttons underneath the main editing area.

These subject areas are stored with the main subject area when the diagram is saved to file.

Manage Subject Layers

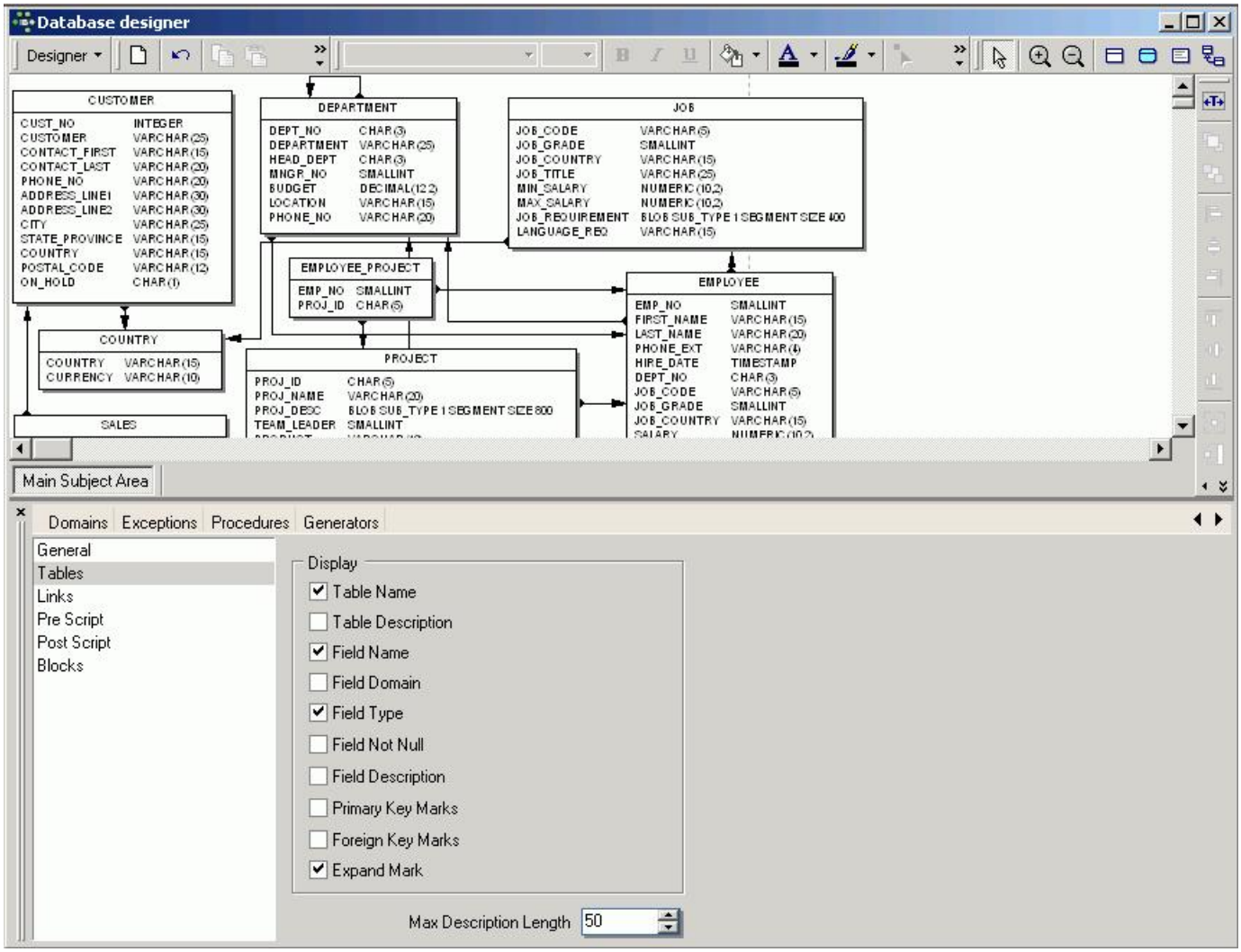
This filter option allow certain specified tables and their relationships to be viewed. Simply click the *NewLayer* icon, name the layer, and check those objects to be included. In order to view everything again, it is necessary to reopen the *Manage Layers* dialog, and click the icon *ShowAll*.



The diagram created may be saved to file or exported using the respective Designer menu item or *Save* icon.

Model Options

The Model Options menu item opens a new window in the lower half of the Database Designer dialog. Here the following visual display and script options may be selected:

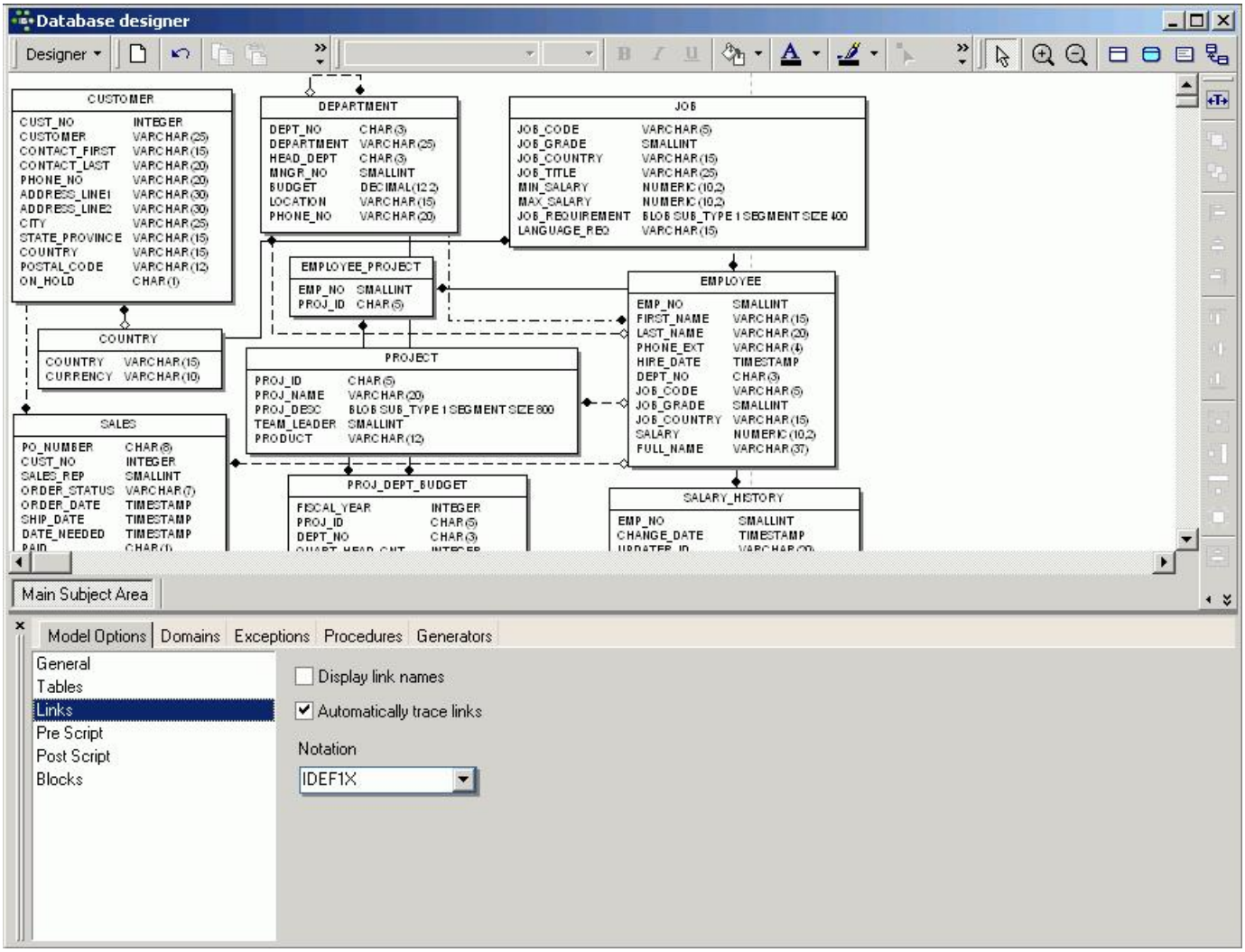


When a table or view is double-clicked in the main editing area, an additional window appears automatically in the model options dialog.

- **General:** Since version 2004.6.17 it is possible to specify the font character set for model objects. Simply click *General* on the left-hand list, and specify the character set using the pull-down list.
- **Table:** Options to display the following: *Table Name* and *Description*, *Field Name*, *Domain* (since IBExpert version 2006.08.12), *Type*, *Not Null* and *Description*, *Primary Key* and *Foreign Key Marks* and *Expand Marks*. It is even possible to specify the maximal description length.
- **Links:** *Display Link Names* (i.e. display FK relationships) and *Automatically Trace Links* (displays the links as horizontal/vertical lines with 90° corners).

By double-clicking on the line between two tables, the relationships are shown in detail. The name and automatic tracing of links, are options already mentioned, included in the Model Options menu item.

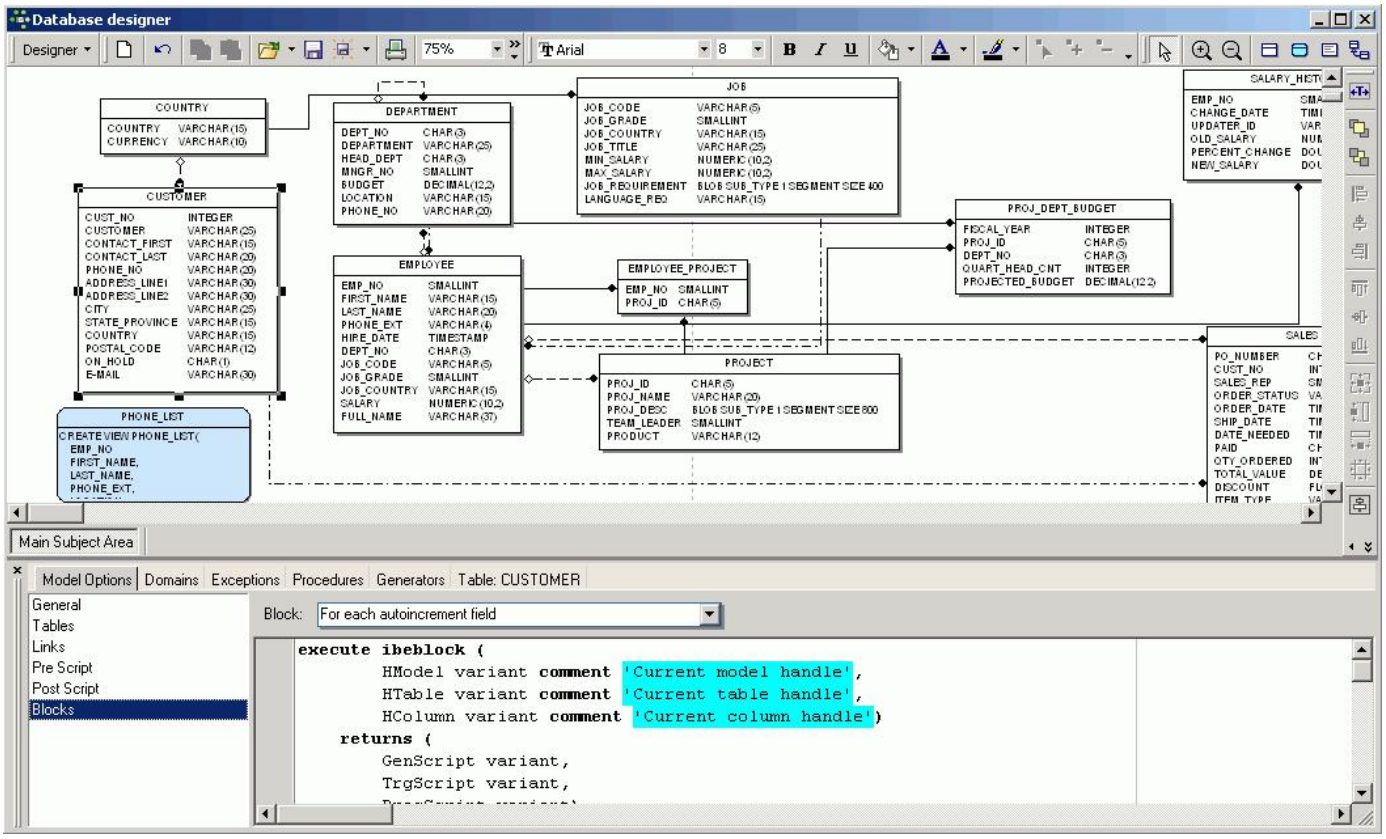
New to IBExpert version 2005.04.24 is the added support for following reference notations: `IDEF1X`, `DM`, `IE`. Simply click the *Notation* drop-down list and select as required. Close the Model Options window, and your model ist notated to the norm specified:



The pre- and postscript options were added in IBExpert version 2003.12.18.1. This offers the possibility to define pre- and postscripts for your database model. The prescript will be inserted into the model script just after the CREATE DATABASE or CONNECT statement. The postscript will be added to the end of the model script. There is also an added option allowing you to define pre- and postscripts for each table separately.

IBExpert version 2007.02.22 introduced added support for autoincrement fields based on the IBEBlock feature. To automatically create generators and triggers for autoincrement fields you have to mark the necessary fields as autoincrement and define for each autoincrement field block (*Model Options / Blocks*):

```
execute ibeblock (
    HModel variant comment 'Current model handle',
    HTable variant comment 'Current table handle',
    HColumn variant comment 'Current column handle')
returns (
    GenScript variant,
    TrgScript variant,
    ProcScript variant)
as
begin
  LF = ibec_CRLF();

  TblName = ibec_dbd_GetObjectProp(HTable, 'NAME');
  FldName = ibec_dbd_GetObjectProp(HColumn, 'NAME');

  GenName = 'GEN_' || TblName || '_' || FldName;
  GenName = ibec_AnsiUpperCase(GenName);

  TrgName = TblName || '_BI';
  TrgName = ibec_AnsiUpperCase(TrgName);

  GenScript = 'CREATE GENERATOR ' || GenName || ';' || LF ||
              'SET GENERATOR ' || GenName || ' TO 0;' || LF;

  TrgScript = 'CREATE TRIGGER ' || TrgName || ' FOR ' || TblName || LF ||
         'ACTIVE BEFORE INSERT POSITION 0' || LF ||
         'AS' || LF ||
         'BEGIN' || LF ||
         '  IF (' || FldName || ' IS NULL) THEN' || LF ||
         '    NEW.' || FldName || ' = GEN_ID(' || GenName || ', 1);' || LF ||
         'END^' || LF || LF;
end
```



The *Model Options* window may be closed by clicking the small black X in the top left-hand corner.

Domains

The *Model Options* also included a *Domains* page with various insert, alter and delete options, similar to the Domain Editor in the DB Explorer.

Exceptions

The *Model Options* also included an *Exceptions* page with various insert, alter and delete options, similar to the Exception Editor in the DB Explorer. The support of exceptions and stored procedures has been included since IBExpert version 2.5.0.6.1.

Procedures

The *Model Options* also included a *Procedures* page, similar to the Procedure Editor in the DB Explorer. The support of stored procedures has been included since IBExpert version 2.5.0.6.1.

It is possible to insert a new procedure or delete a selected procedure, using the relevant icons. Procedures can be selected from the pull-down list to the right of these icons. The code can be altered as wished; the editing page offering all those features included in all IBExpert Edit pages (such as Code Completion, comprehensive right-click menu (SQL Editor Menu) etc).

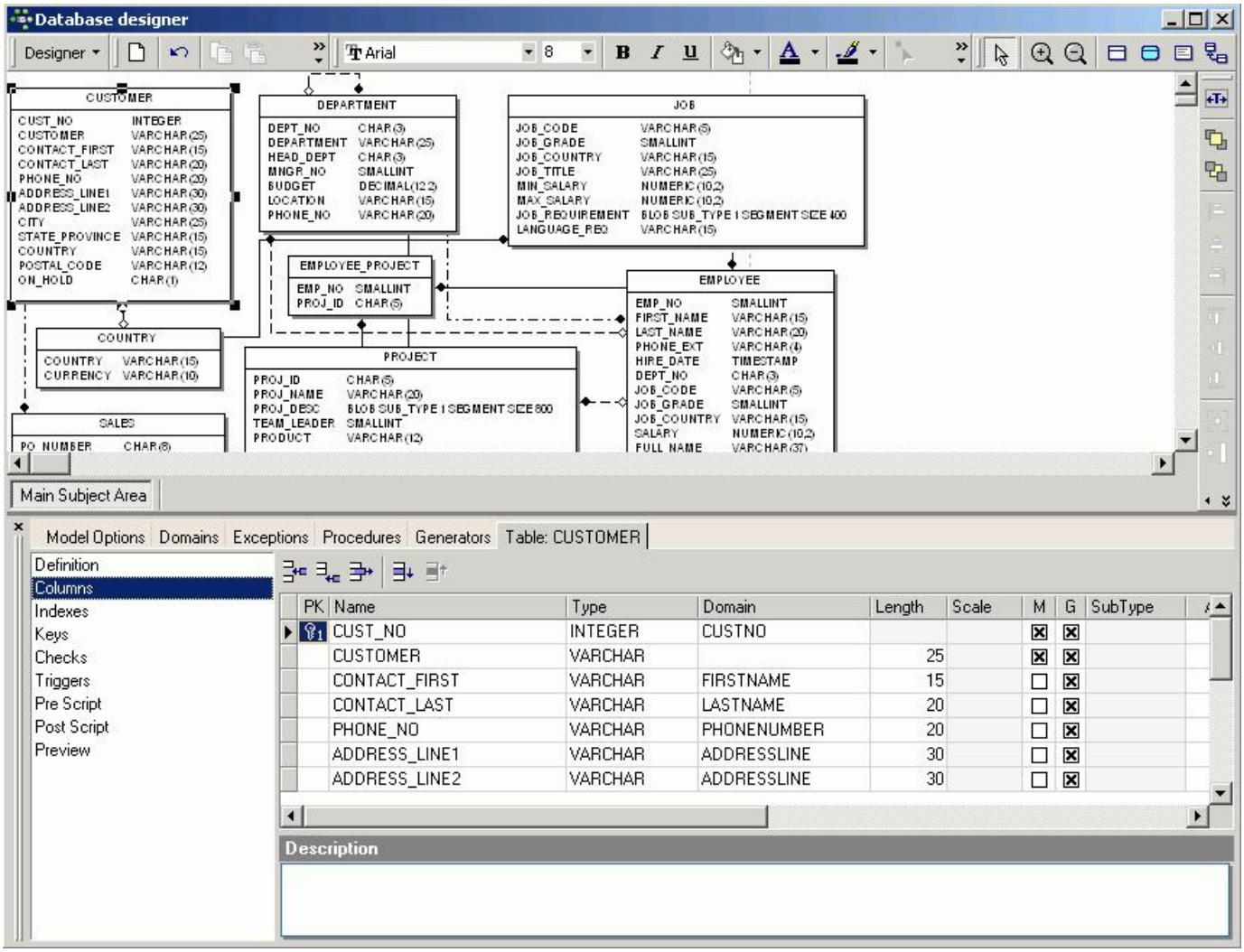
Generators

The *Model Options* also include a *Generators* page with various insert, alter and delete options, similar to the Generator Editor in the DB Explorer. The support of generators has been included since IBExpert version 2004.1.22.1.

Selected Table / Selected View

Table <selected table>: The options allow columns, indices, keys, checks and triggers to be added, amended or deleted. This version of the IBExpert Table Editor can be used to create a new table or view, or alter an existing selected table. For details please refer to Create Table and Table Editor.

View <selected view>: A new view can only be created in the Database Designer using SQL. Alternatively create a new view in the DB Explorer, and update an existing diagram using Reverse Engineer... . For further information regarding view creation in the IBExpert DB Explorer, please refer to New View.



The *Definitions* page displays the table or view name, allows a description to be displayed/entered and the *Generate* check option allows the selected table or view to be updated in the diagram.

The *Selected Table* options: *Columns*, *Indexes*, *Keys*, *Checks*, *Triggers* and *Preview*, and the *Selected View* options, *SQL*, *Triggers* and *Preview*, are based on those pages found in the Table Editor and View Editor in the DB Explorer. There is however a number of abbreviations included in these frames, which are not included in the DB Explorer editors. These have the following meaning:

- G = generate, i.e. include into the result script.
- U = unique (for indices)
- A = active (for triggers)
- M = mandatory (for columns, i.e. `NOT NULL`)

IBExpert version 2006.12.11 introduced the possibility to specify expressions for indices (new column on the *Indexes* page).

Since version 2003.12.18.1 there is also the possibility to define pre- and postscripts for each table separately. The prescript will be inserted into the model script just after the `CREATE DATABASE` or `CONNECT` statement. The postscript will be added to the end of the model script. You can also define pre- and postscripts for each table separately.

Since version 2.5.0.61 IBExpert has increased the flexibility with regard to customizing table layout. It is possible to toggle on/off displaying of *field name*, *field type*, *NOT NULL* flag, *field description* and *foreign key* mark in any combination. It is also possible to display the table description instead of, or together with the table name.

In IBExpert version 2003.11.6.1 the *ViewEditor* and *Note Editor* were redesigned. They are now no longer modal.

Comment Box

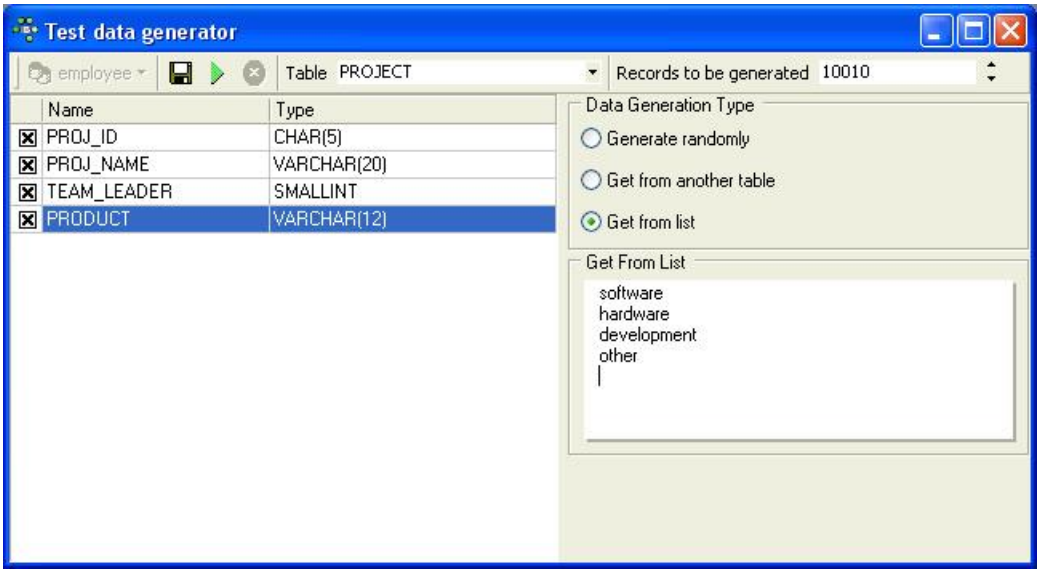
When a Comment Box is inserted into the main diagram, double-clicking upon this box produces a new Comment Box page in the *Model Options* dialog. This can be used to insert, alter or delete a comment text as wished.

In IBExpert version 2003.11.6.1 the *ViewEditor* and *Note Editor* were redesigned. They are now no longer modal.

See also:
Database Design

Test Data Generator

The IBExpert Test Data Generator can be found in the IBExpert Tools menu. (This feature is unfortunately not included in the IBExpert Personal Edition.)



A database connection must already exist. Select the database for which test data is to be generated, if more than one database is connected. To generate data for a specific table, select the table, then select the number of data sets to be generated. Over 100,000 data sets are not a problem for IBExpert here, even when working locally, although it may take a little time. Click on the individual fields and specify the contents on the right. It is possible to specify the following:

Data Generation Type: options here include:

- **Generate randomly:** User-defined constraints include the following:
  - *Integer:* the minimum and maximum value.
  - *Float:* check option *Fixed Float Number*, and user specification of number of digits and level of precision.
  - *String:* the minimum and maximum length; the range of characters within the character set which may be used for the data content.
  - *Date:* the minimum and maximum date, and a check option, whether a time slice should also be included.
- **Get from another table:** Specify table, field and number of records. This is a useful way of generating test data for a foreign key field.
- **Get from a list:** A list can be typed or pasted in the panel.
- **Autoincrement:** This option is of course only offered for integral fields, and enables the developer to specify an initial value, and the interval (step).

Finally execute (green > icon or [F9]), and watch the counter generate the test data!

The data can finally be viewed in the Table Editor on the Data page:

Table : [PROJECT] : employee (:C:\Programme\Firebird\examples\EMPLOY...

Table | 2437 records in table | PROJECT

Fields | Constraints | Indices | Dependencies | Triggers | Data | Description | DDL | Grants | Logging

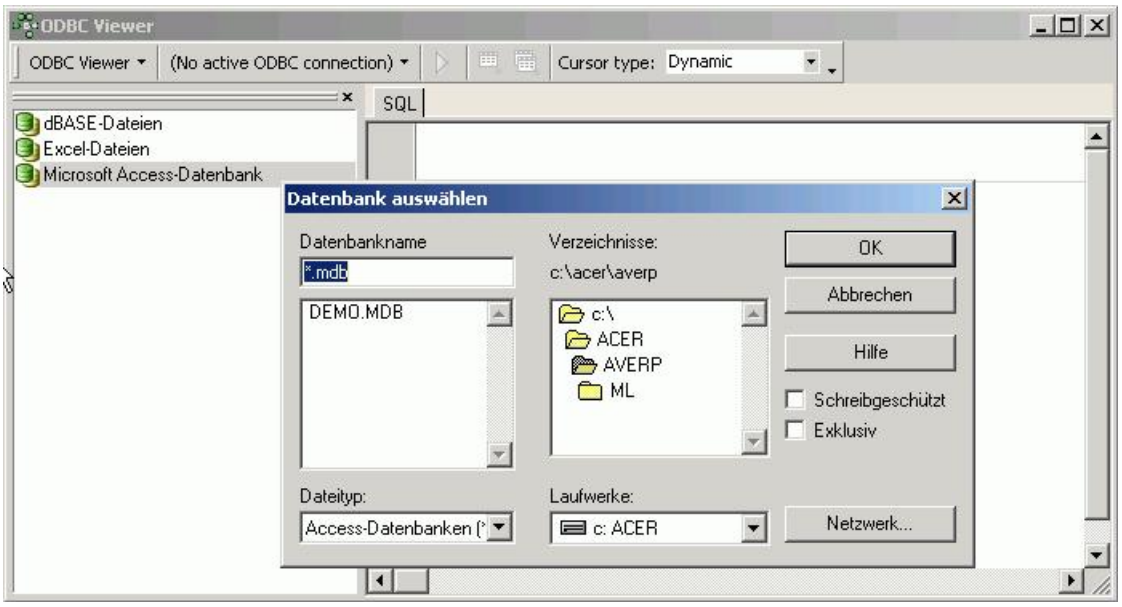
Record: 1 | 2437 records fetched

PROJ_ID	PROJ_NAME	TEAM_LEADER	PRODUCT	PRO
AABVA	FUSSQBNUCUGTHNKV	11	other	
AALWS	ICVDJLLOCBXWRJ	44	other	
AASOD	EXKPAOMKSUSRKGTYE	46	software	
ABLPI	AXYYJFSSXWG	11	software	
ABTYY	UIMFUXVA	65	hardware	
ACJAY	LPMSLMELJMPFTDC	20	hardware	
ACSLC	BDQCYVQQ	34	software	
ACSSA	FXVFYGYXH	29	other	
ACUMY	LGRDCTGJYSSJ	29	other	
ADESW	CCXUGRXTNXVNWCSQK	20	software	
ADFRV	WMSXYNTN	94	software	
ADHOW	QJEUMHMHUAWIVRFEA	85	software	
ADOLJ	BMWIGQM	37	other	
ADYXQ	VIYHPMKTPJVUNIRK	8	software	
AEIXH	MXPGTAJEL	94	hardware	
AENDL	AIEUPSKTV	2	software	
AETSU	NKYRMWHXMKTG	20	software	
AFBRG	YEVFNTCWBDVRPUJTH	94	other	
AFCUD	FSVVVIRPWVOUDN	24	hardware	
AFIND	PTHVUENVGISXXST	29	other	
AFJTU	DQIHPITSLVKP	8	other	
AFLXU	CPIHFIWINYWKQ	20	software	
AFRUW	XMRPARCQTTA	29	hardware	
AFWMI	XRQSAOBMOYFBI	44	hardware	
AGBXN	CRRGNFJF	9	other	
AGCSK	STCHNDWEFUTF	65	software	
AGDKY	OIKPXNILSYTU	65	other	
AGIAP	CJSGKQAOEJDL	37	software	
AGJPS	MKXAIBJD	12	other	
AGSUE	NHIKYMGVJJAEGSUX	44	other	
AHCTX	VGDKAPUQYXOYXBKS	46	hardware	
AHIBV	YQENAUORUYV	83	hardware	
AHISG	TKHRRRQY	2	other	
AHOUY	QQGAURT	28	hardware	
AIAIV	QJTWSTSRMBROJH	46	software	
AIBNX	FNFTDAABBJJB	12	other	

Grid View | Form View | Print Data

ODBC Viewer

New to IBExpert version 2007.09.25 the ODBC Viewer allows you to browse data from any ODBC source available on your PC and also export data from an ODBC source into an SQL script or directly into a Firebird/InterBase database.

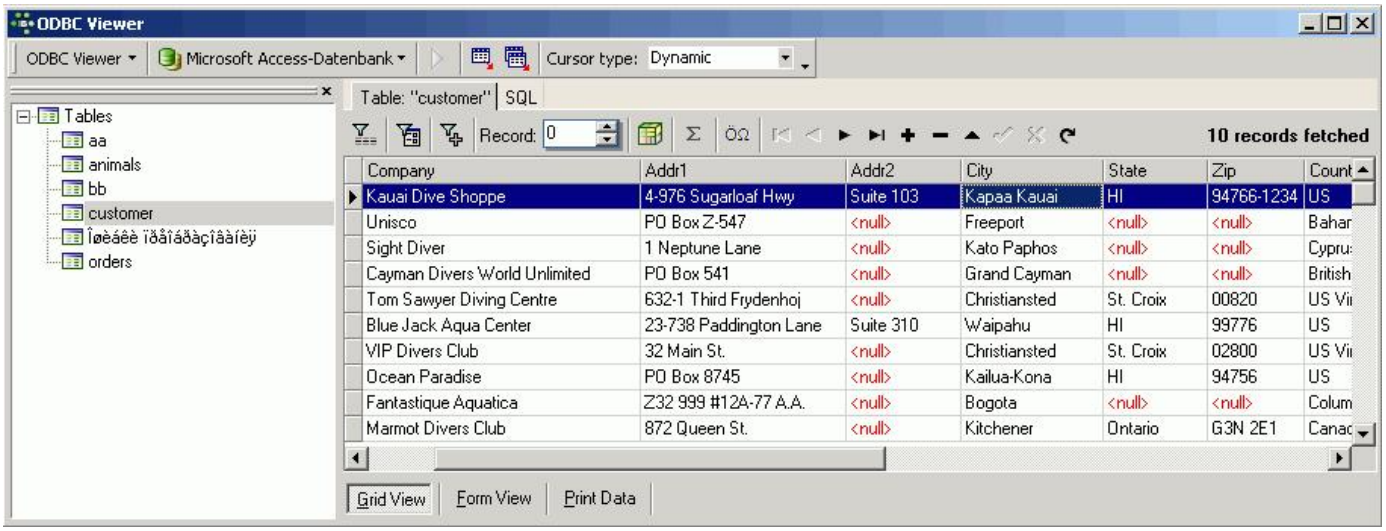


Simply select the database from the selection of formats: dBASE or Excel files, or Microsoft Access databases, to load the database tables.

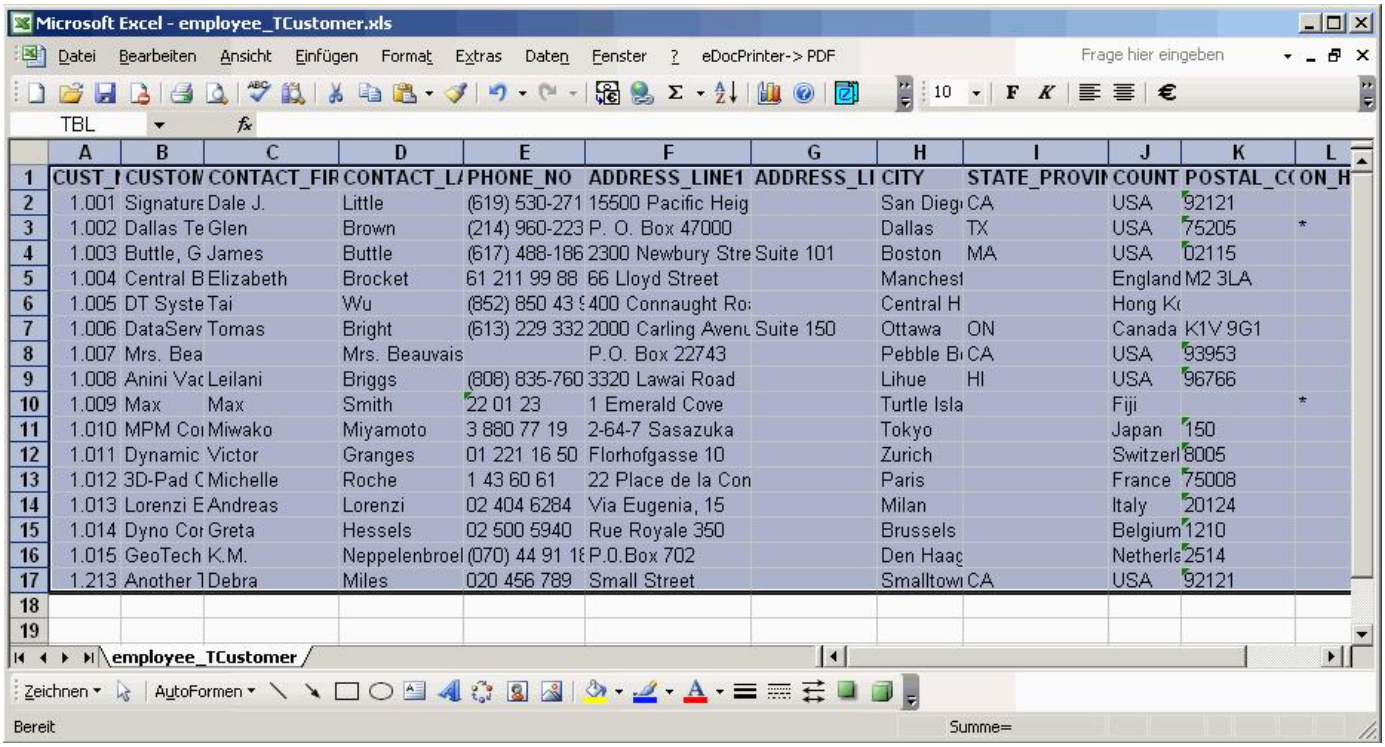
The navigational buttons and icons displayed on the tool bar running across the head of the table data are explained in detail under: Table Editor / Data Grid. The ODBC Viewer's right-click menu is also detailed in this chapter.

The IBExpert Blob Viewer/Editor is automatically opened by double-clicking on any Blob field.

Double-click on a table from the list on the left, to view the data contents. The view type can be easily altered by clicking on the buttons at the bottom left: Grid View, Form View and there is even the possibility to print the data. More information regarding these options can be found under: IBExpert Tools Menu / SQL Editor / Results.

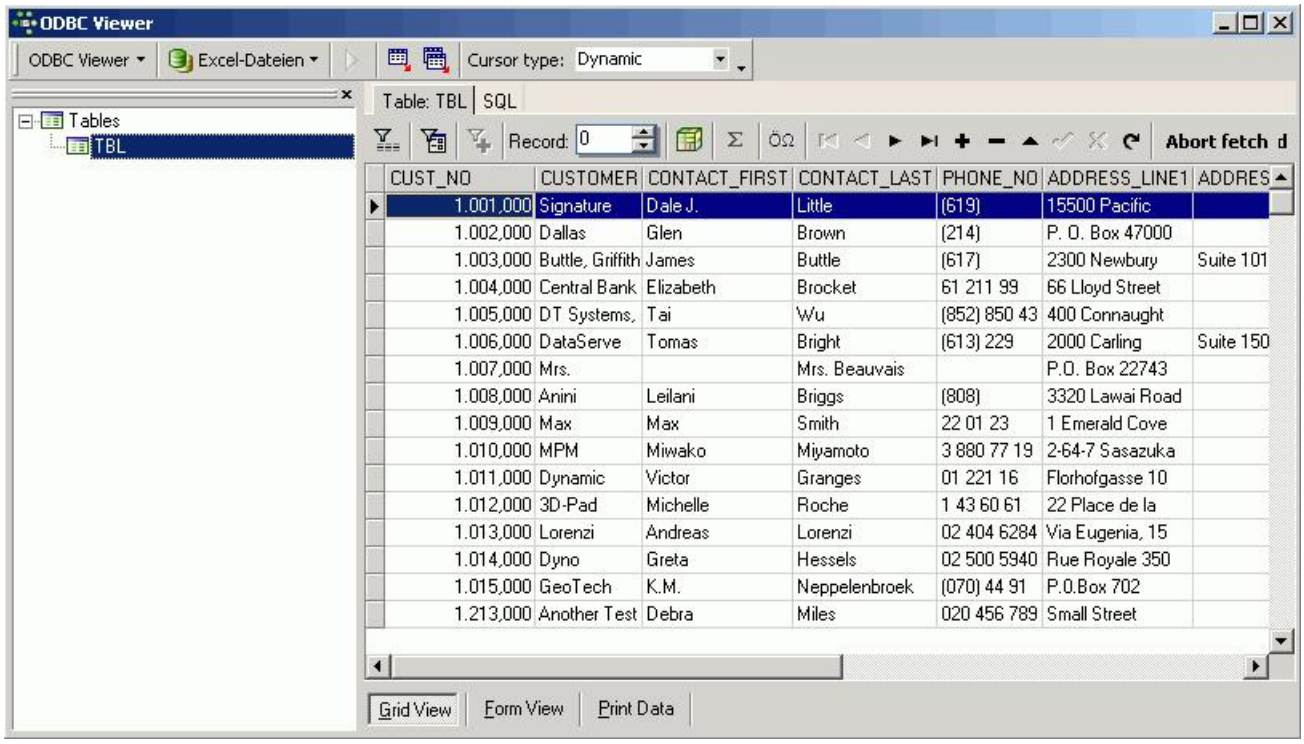


In Excel it is possible to define a specific area (a whole table or just parts of the data contents) and give this marked area a name (in the upper left area):



	A	B	C	D	E	F	G	H	I	J	K	L
1	CUST_I	CUSTOM	CONTACT_FIR	CONTACT_LA	PHONE_NO	ADDRESS_LINE1	ADDRESS_LI	CITY	STATE_PROVIN	COUNT	POSTAL_CO	ON_H
2	1.001	Signature	Dale J.	Little	(619) 530-271	15500 Pacific Heig		San Diego	CA	USA	92121	
3	1.002	Dallas Te	Glen	Brown	(214) 960-223	P. O. Box 47000		Dallas	TX	USA	75205	*
4	1.003	Buttle, G	James	Buttle	(617) 488-186	2300 Newbury Stre	Suite 101	Boston	MA	USA	02115	
5	1.004	Central B	Elizabeth	Brocket	61 211 99 88	66 Lloyd Street		Manchest		England	M2 3LA	
6	1.005	DT Syste	Tai	Wu	(852) 850 43	400 Connaught Ro		Central H		Hong Ko		
7	1.006	DataServ	Tomas	Bright	(613) 229 332	2000 Carling Avenu	Suite 150	Ottawa	ON	Canada	K1V 9G1	
8	1.007	Mrs. Bea		Mrs. Beauvais		P.O. Box 22743		Pebble Bi	CA	USA	93953	
9	1.008	Anini Vac	Leilani	Briggs	(808) 835-760	3320 Lawai Road		Lihue	HI	USA	96766	
10	1.009	Max	Max	Smith	22 01 23	1 Emerald Cove		Turtle Isla		Fiji		*
11	1.010	MPM Co	Miwako	Miyamoto	3 880 77 19	2-64-7 Sasazuka		Tokyo		Japan	150	
12	1.011	Dynamic	Victor	Granges	01 221 16 50	Florhofgasse 10		Zurich		Switzerl	8005	
13	1.012	3D-Pad C	Michelle	Roche	1 43 60 61	22 Place de la Con		Paris		France	75008	
14	1.013	Lorenzi E	Andreas	Lorenzi	02 404 6284	Via Eugenia, 15		Milan		Italy	20124	
15	1.014	Dyno Co	Greta	Hessels	02 500 5940	Rue Royale 350		Brussels		Belgium	1210	
16	1.015	GeoTech	K.M.	Neppelenbroel	(070) 44 91 1	P.O.Box 702		Den Haag		Netherla	2514	
17	1.213	Another T	Debra	Miles	020 456 789	Small Street		Smalltown	CA	USA	92121	
18												
19												

This defined data can then be used as a table in the ODBC Viewer (our example has been defined in Excel as TBL):

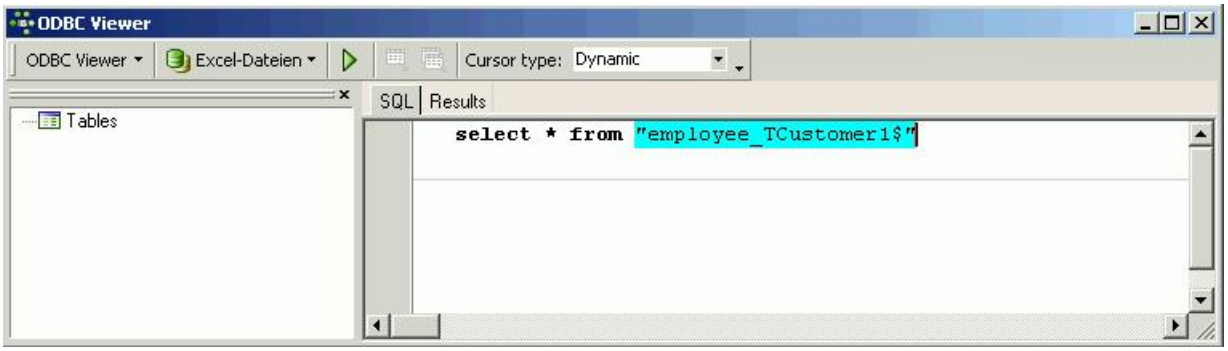


CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_LAST	PHONE_NO	ADDRESS_LINE1	ADDRES
1.001,000	Signature	Dale J.	Little	(619)	15500 Pacific	
1.002,000	Dallas	Glen	Brown	(214)	P. O. Box 47000	
1.003,000	Buttle, Griffith	James	Buttle	(617)	2300 Newbury	Suite 101
1.004,000	Central Bank	Elizabeth	Brocket	61 211 99	66 Lloyd Street	
1.005,000	DT Systems,	Tai	Wu	(852) 850 43	400 Connaught	
1.006,000	DataServe	Tomas	Bright	(613) 229	2000 Carling	Suite 150
1.007,000	Mrs.		Mrs. Beauvais		P.O. Box 22743	
1.008,000	Anini	Leilani	Briggs	(808)	3320 Lawai Road	
1.009,000	Max	Max	Smith	22 01 23	1 Emerald Cove	
1.010,000	MPM	Miwako	Miyamoto	3 880 77 19	2-64-7 Sasazuka	
1.011,000	Dynamic	Victor	Granges	01 221 16	Florhofgasse 10	
1.012,000	3D-Pad	Michelle	Roche	1 43 60 61	22 Place de la	
1.013,000	Lorenzi	Andreas	Lorenzi	02 404 6284	Via Eugenia, 15	
1.014,000	Dyno	Greta	Hessels	02 500 5940	Rue Royale 350	
1.015,000	GeoTech	K.M.	Neppelenbroek	(070) 44 91	P.O.Box 702	
1.213,000	Another Test	Debra	Miles	020 456 789	Small Street	

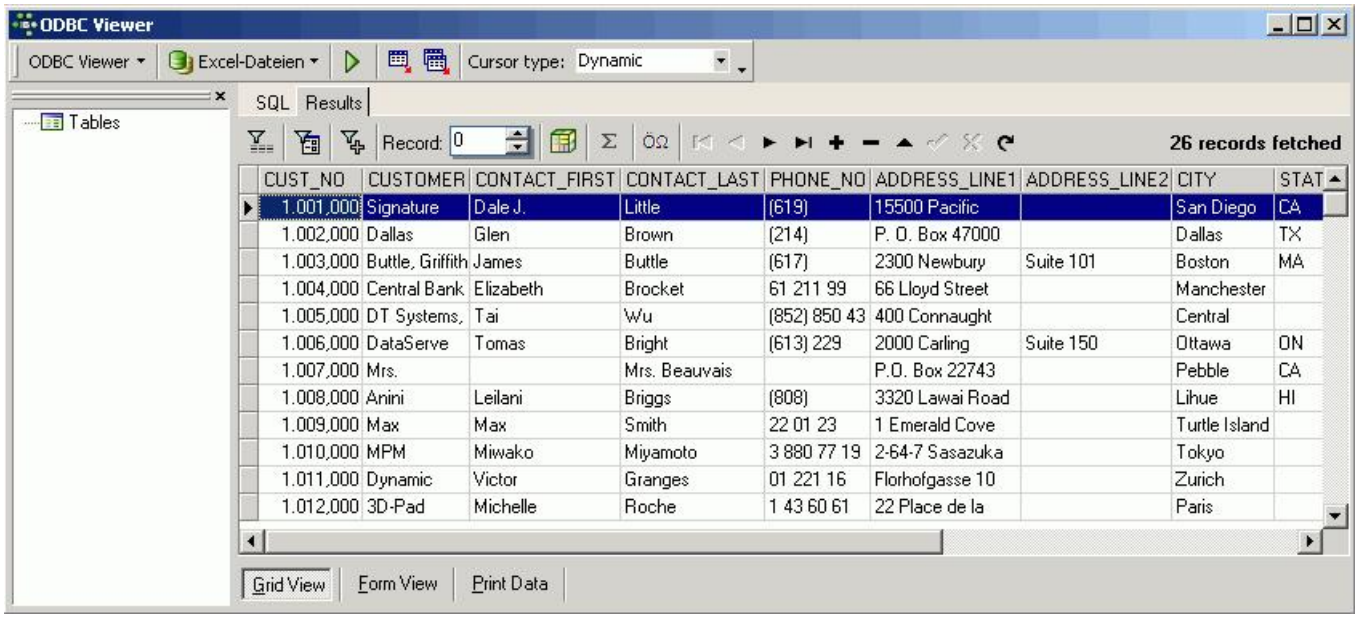
Alternatively an Excel file which is connected via ODBC can be viewed by typing the query:

```
select * from "sheet1$"
```

where `sheet1$` is the name of the spread sheet (visible on the tab at the bottom of the sheet). To view our example above:



The first line is used always used for the column names.



CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_LAST	PHONE_NO	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STAT
1.001,000	Signature	Dale J.	Little	(619)	15500 Pacific		San Diego	CA
1.002,000	Dallas	Glen	Brown	(214)	P. O. Box 47000		Dallas	TX
1.003,000	Buttle, Griffith	James	Buttle	(617)	2300 Newbury	Suite 101	Boston	MA
1.004,000	Central Bank	Elizabeth	Brocket	61 211 99	66 Lloyd Street		Manchester	
1.005,000	DT Systems,	Tai	Wu	(852) 850 43	400 Connaught		Central	
1.006,000	DataServe	Tomas	Bright	(613) 229	2000 Carling	Suite 150	Ottawa	ON
1.007,000	Mrs.		Mrs. Beauvais		P.O. Box 22743		Pebble	CA
1.008,000	Anini	Leilani	Briggs	(808)	3320 Lawai Road		Lihue	HI
1.009,000	Max	Max	Smith	22 01 23	1 Emerald Cove		Turtle Island	
1.010,000	MPM	Miwako	Miyamoto	3 880 77 19	2-64-7 Sasazuka		Tokyo	
1.011,000	Dynamic	Victor	Granges	01 221 16	Florhofgasse 10		Zurich	
1.012,000	3D-Pad	Michelle	Roche	1 43 60 61	22 Place de la		Paris	

IBExpert command-line tools

Please note that from IBExpert version 2005.06.07 IBEExtract and IBECompare will no longer be supported as their functionality is now available via IBEScript.exe and EXECUTE IBEBLOCK.

For those of you working with older versions of IBExpert, the following command-line tools are available:

- IBECompare
- IBEExtract
- IBEScript

These cover the majority of the options offered by the InterBase command-line utilities and much more.

To be allowed to distribute any of the IBExpert Modules (ibexpert.exe, ibescript.exe, ibescript.dll, ibeextract.exe and ibecompare.exe) together with your application, you need:

- IBExpert Site License, if the distribution is located only on computers in your own company.
- IBExpert VAR License, if the distribution is located on any computer outside your company.
- IBExpert Junior VAR a "slim" VAR version for those not requiring all IBExpert command-line modules.

If you are already an IBExpert customer, you can upgrade to a Site or VAR License and directly buy the 24 month Extension Product. See http://ibexpert.net/ibe/pmwiki.php?n=Main.Upgrade for details.

Some functions of the new IBExpert Modules do not work on non-licensed computers, so you can only use them where your IBExpert license is valid.

Customers with a Site License are allowed to use them on every computer in their company just by copying the License file to the path, where the module (such as ibescript.exe) should run.

VAR License customers may also integrate these modules and the License file in their software installation.

IBECompare

Please note that from IBExpert version 2005.06.07 IBECompare will no longer be supported as its functionality is now available IBEScript.exe and EXECUTE IBEBLOCK.

For those of you working with older versions of IBExpert, IBECompare is a command-line tool to compare databases, scripts and table data. It is new to IBExpert version 2004.04.01.1. The current version (04/2005) is 2005.04.24.1.

IBECompare.exe can be found in the IBExpert root directory, and needs to be started from DOS:

```
c:\Program Files\HK-Software\IBExpert 2004>ibecompare
```

IBECompare offers the following options:

- **-C<config_file>** = config file
- **-O<output_file>** = output file (Result.sql if not specified)
- **-V<verbose_file>** = verbose file
- **-D** = compare database metadata and script
- **-T** = compare table data
- **-S** = silent mode
- **-s** = create a config file sample (config_sample.ini)

WARNING: All options are case-sensitive!

Example

```
IBECompare -D -Cconfig.ini -OC:\Scripts\result.sql -Vlog.txt
```

In both cases (i.e. options -D or -T) IBECompare produces an SQL script file. It is necessary to specify an input settings file using the -C option.

You can obtain the template of this file starting IBECompare with the -s option (IBECompare -s). In this case IBECompare will create a config_sample.ini file within the current directory, which is simple and quick to modify.

It is also possible to create a settings file using *Save* configuration button in the IBExpert Tools menu / Database Comparer.

The following is an example of an .ini file, for comparing table data:

```
[MasterDB]
ConnectString=LOCALHOST:C:\MyData\Master.gdb
```

```
Username=SYSDBA
Password=masterkey
Charset=WIN_1251
ClientLib=gds32.dll
; Next item will be used while comparing tables
TableName=CUSTOMER

; Instead of MasterDB section you can use MasterScript section:
;[MasterScript]
; ScriptFile=D:\MyScripts\MyData.dql

[TargetDB]
ConnectString=MYSERVER:D:\Data\customer.gdb
Username=SYSDBA
Password=masterkey
Charset=WIN_1251
ClientLib=gds32.dll
; Next item will be used while comparing tables
TableName="Customer"

; Instead of TargetDB section you can use TargetScript section:
;[TargetScript]
;ScriptFile=D:\MyScripts\MyData.dql

[CompareObjects]
Domains=1
Tables=1
Views=1
Triggers=1
Procedures=1
Generators=1
Exceptions=1
Functions=1
Roles=1
Indices=1
Grants=1
Descriptions=1
PrimaryKeys=1
ForeignKeys=1
Uniques=1
Checks=1

[Options]
; Next items will be used while comparing tables
ProcessINSERTs=1
ProcessUPDATEs=1
ProcessDELETEs=1
```

Should the script generated by `IBECompare` include a

```
SET BLOBFILE 'xxx.lob';
```

command, it is necessary to execute the script using `IBEScript` or the IBExpert Script Executive.

`SET BLOBFILE` is a special extension of script language that allows insert or update blob values via script.

See also:
Script Executive
Database Comparer
Table Data Comparer

IBEExtract

Please note that from IBExpert version 2005.06.07 `IBEExtract` will no longer be supported as its functionality is now available `IBEScript.exe` and `EXECUTE IBEBLOCK`. Please refer to `ibec_ExtractMetadata` if you are using a version post 2005.06.07.

For those of you working with older versions of IBExpert, `IBEEXtract.exe` can be found in the IBExpert root directory, and needs to be started from DOS. The current version (04/2005) is 2005.04.24.1.

Syntax

IBEExtract database [options]

- **-U<user_name>** = user name (*SYSDBA* if not specified).
- **-P<password>** = password (*masterkey* if not specified).
- **-C<character_set>** = character set.
- **-O<output_file>** = output file (`Result.sql`" if not specified).
- **-F<output_folder>** = output folder (for *Separate Files* mode; current directory, if not specified).
- **-G** = set generator values.
- **-D** = extract data.
- **-B** = extract blobs (please refer to blob fields for further information about blobs).
- **-S** = silent mode.
- **-V<verbose_file>** = verbose file.
- **-M<config_file>** = use config file.

- **-T** = generate CREATE DATABASE statement.
- **-N** = generate CONNECT statement.
- **-W** = include password into CREATE DATABASE or CONNECT statement.
- **-R** = extract object descriptions.
- **-A<integer_value>** = commit after <integer_value> records.
- **-Y** = extract computed fields separately.
- **-X** = extract privileges.
- **-L** = extract privileges only for selected objects.
- **-d** = date format (native InterBase/Firebird date format <DD-MMM-YYYY>, if not specified).
- **-f** = extract into separate files (new to IBExpert version 2004.9.12.1/IBEExtract version 2.02).
- **-s** = extract into separate files.
- **-r** = use REINSERT instead of repeated INSERTS.
- **-l** = client library file (gds32.dll, if not specified).
- **-z** = maximum size of resulting files in megabytes (new to IBExpert version 2004.9.12.1/IBEExtract version 2.02).
- **-u** = Use UPDATE instead of DESCRIBE option (new to IBExpert and IBEExtract versions 2005.04.24)

WARNING! All options are case-sensitive!

Example 1

```
IBEExtract localhost:c:\mydata\mydatabase.gdb -OC:\scripts\result.sql -USYSDBA -Pmasterkey -CWIN1251
```

Example 2

```
IBEExtract "C:\IB Data\my.gdb" -O"My Script.sql" -V"Extract Log.txt"
```

Since IBExpert version 2003.11.6.1, the problem with extracting exceptions has been solved.

All options listed here can also be found in IBExpert under Tools / Extract Metadata.

See also:
Extract Metadata ibec_ExtractMetadata

IBEScript

IBEScript.exe can be found in the IBExpert root directory, and needs to be started from DOS. The current version (04/2005) is 2005.04.24.1.

Syntax

```
IBEScript script_filename [options]
```

- **-S** = silent mode
- **-V<verbose_file>** = verbose output file. If <verbose_file> exists, IBEScript will overwrite it.
- **-v<verbose_file>** = verbose output file. If <verbose_file> exists, IBEScript will append message to this file.
- **-E** = display only error messages
- **-N** = continue after error.
- **-T** = write timestamp into log.
- **-D** = connections string (use it if your script does not contain CONNECT or CREATE DATABASE statements).
- **-P** = connection password (use only with -D option).
- **-R** = connection role (use only with -D option) (new to IBExpert version 2005.08.08)
- **-U** = connection user name (use only with -D option).
- **-C** = character set (use only with -D option).
- **-L<1|2|3>** = SQL dialect (use only with -D option; 1 if not specified)
- **-i** = idle priority (new to IBExpert version 2004.9.12.1 / IBEScript version 2.02).

WARNING! All options are case-sensitive!

Since IBExpert version 2003.11.6.1 there is the added possibility to encrypt/decrypt scripts and to execute encrypted scripts. There are two possible ways to encrypt:

1. Encrypting without the password. In this case there is no possibility to decrypt an encrypted script but it is possible to execute this script with IBEScript.
2. Encrypting with the password. In this case it possible to decrypt the script and execute it with IBExpert if the correct password is specified.

The following options control the encrypting and decrypting:

- **-e** = encrypts a script file and create a file with the extension .esql if the output file is not specified (no execution will be performed).
- **-d** = decrypts an encrypted script file if it was encrypted with password (no execution will be performed).
- **-p<password>** = encrypt/decrypt password.
- **-o<file_name>** = output file name for encrypted and decrypted scripts.

Again: all options are case-sensitive!

Please note that IBExpert cannot work with scripts larger than 2 GB. Should the script exceed 2 GB, you will need to split it into two or more smaller ones.

Example 1

```
IBEScript "C:\MyScripts\CreateDB.sql"
```

Example 2

```
IBEScript C:\MyScripts\CreateDB.sql -S -UScriptLog.txt
```

Support for EXECUTE IBEBLOCK was implemented in IBEScript version 2.02 (released with IBExpert version 2004.9.12.1). This is unfortunately not available in the free version of IBEScript.

Support was added for the COMMENT ON statement (Firebird 2) in IBExpert version 2005.09.25.

See also:
IBEBlock
Script Executive

IBEScript.dll

New to IBExpert version 2004.12.12.1: IBEScript.dll (for registered customers only).

For registered customers we've included the IBEScript.dll in the installation archive. You can use it in your applications to execute scripts from file or from a string buffer. There is a small demo application illustrating its use in the IBEScriptDll folder. Please also refer to the IBEScriptDll Readme.txt.

To be allowed to distribute any of the IBExpert modules (ibexpert.exe, ibescript.exe, ibescript.dll, ibeextract.exe and ibecompare.exe) together with your application, please refer to the beginning of this chapter.

IBEScriptDll Readme.txt

1. IBEScript.dll exports the following functions:

- **ExecScriptFile:** executes script from file.
- **ExecScriptText:** executes script from string buffer.
- **CONNECT:** connects to the database if there is no CONNECT statement in the script.

2. Examples of the use of ExecScriptFile and ExecScriptText: see demo application in the IBEScriptDll folder.

3. Example using the CONNECT function:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Hndl : THandle;
  ESP : TExecuteScriptProc;
  CP : TConnectDBProc;
  s : string;
  Res : integer;
begin
  ErrCount := 0;
  StmtCount := 0;
  mLog.Lines.Clear;
  s := mScript.Text;
  if Trim(s) = '' then
  begin
    ShowMessage('Nothing to do!');
    Exit;
  end;
  try
    Hndl := LoadLibrary(PChar('IBEScript.dll'));
    if (Hndl > HINSTANCE_ERROR) then
    begin
      ESP := GetProcAddress(Hndl, 'ExecScriptText');
      CP := GetProcAddress(Hndl, 'Connect');
      if (@ESP <> nil) and (@CP <> nil) then
      begin
        Pages.ActivePage := tsOutput;
        Res := CP(PChar('db_name=localhost:c:\empty.fdb; password=masterkey; user_name=SYSDBA;'

+
                        'lc_ctype=win1251; sql_role_name=ADMIN; sql_dialect=3;' +
                        'clientlib="c:\program files\firebird\bin\fbclient.dll"'), @CEH);
        if Res = 0 then
          ESP(PChar(s), @HandleError, @BeforeExec, @AfterExec);
      end;
    end;
  finally
    if Hndl > HINSTANCE_ERROR then
      FreeLibrary(Hndl);
  end;
end;
```

See also:
InterBase and Firebird command-line utilities

InterBase and Firebird command-line utilities

Several command-line tools are provided with InterBase/Firebird. They perform the same range of functions as the Server Manager and run on both UNIX and Windows platforms. Like the Server Manager, they can access servers on any platform that InterBase supports. The command-line tools include the following:

- fbguard.exe
- fbserver.exe
- fb_inet_server.exe
- NBAK
- NBACKUP
- GBAK
- GFIX
- GSEC
- GSTAT
- IBLOCKPR (Windows) GDS_LOCK_PRINT (Unix)
- IBMGR
- ISQL - Interactive SQL

The majority of the options provided by these command-line tools are also offered by IBExpert. Please refer to IBECompare, IBEExtract and IBEScript for further information.

fbguard.exe

The FBGuardian monitors the server process. Should the server go down for whatever reason the Guardian automatically restarts it. Please refer to FBGuardian in the *Download and Install Firebird* chapter for further information.

fbserver.exe

This is the Firebird SuperServer binary.

fb_inet_server.exe

This is the Firebird Classic binary.

On-line incremental backup

New to Firebird 2.0: the implementation of new, fast, on-line, page-level incremental backup facilities. The backup engine comprises two parts:

- NBAK, the engine support module and
- NBACKUP, the tool that does the actual backups.

NBAK

The functional responsibilities of NBAK are:

1. to redirect writes to difference files when asked (ALTER DATABASE BEGIN BACKUP statement),
2. to produce a GUID for the database snapshot and write it into the database header before the ALTER DATABASE BEGIN BACKUP statement returns,
3. to merge differences into the database when asked (ALTER DATABASE END BACKUP statement),
4. to mark pages written by the engine with the current SCN [page scan] counter value for the database,
5. to increment SCN on each change of backup state.

The backup state cycle is:

```
nbak_state_normal -> nbak_state_stalled -> nbak_state_merge -> nbak_state_normal
```

- In normal state writes go directly to the main database files.

- In stalled state writes go to the difference file only and the main files are read-only.
- In merge state new pages are not allocated from difference files. Writes go to the main database files.

Reads of mapped pages compare both page versions and return the version which is fresher, because we don't know if it is merged or not.

Note: This merge state logic has one quirky part. Both Microsoft and Linux define the contents of file growth as "undefined" i.e., garbage, and both zero-initialize them.

This is why we don't read mapped pages beyond the original end of the main database file and keep them current in difference file until the end of a merge. This is almost half of `NBak` fetch and write logic, tested by using modified PIO on existing files containing garbage.

NBACKUP

The functional responsibilities of `NBackup` are:

1. to provide a convenient way to issue `ALTER DATABASE BEGIN/END BACKUP`,
2. to fix up the database after filesystem copy (physically change `nbak_state_diff` to `nbak_state_normal` in the database header),
3. to create and restore incremental backups.

Incremental backups are multi-level. That means if you do a Level 2 backup every day and a Level 3 backup every hour, each Level 3 backup contains all pages changed from the beginning of the day till the hour when the Level 3 backup is made.

Backing up

Creating incremental backups has the following algorithm:

1. Issue `ALTER DATABASE BEGIN BACKUP` to redirect writes to the difference file.
2. Look up the `SCN` and GUID of the most recent backup at the previous level.
3. Stream database pages having `SCN` larger than was found at step 2 to the backup file.
4. Write the GUID of the previous-level backup to the header, to enable the consistency of the backup chain to be checked during restore.
5. Issue `ALTER DATABASE END BACKUP`.
6. Add a record of this backup operation to `RDB$BACKUP_HISTORY`. Record current level, `SCN`, snapshot GUID and some miscellaneous stuff for user consumption.

Restoring

Restore is simple: we reconstruct the physical database image for the chain of backup files, checking that the `backup_guid` of each file matches `prev_guid` of the next one, then fix it up (change its state in header to `nbak_state_normal`).

Usage

```
nbackup <options>
```

Valid Options

- **`-L <database>`:** Lock database for filesystem copy
- **`-N <database>`:** Unlock previously locked database
- **`-F <database>`:** Fixup database after filesystem copy
- **`-B <level> <database> [<filename>]`:** Create incremental backup
- **`-R <database> [<file0> [<file1>...]]`:** Restore incremental backup
- **`-U <user>`:** User name
- **`-P <password>`:** Password

Note:

1. `<database>` may specify a database alias.
2. incremental backups of multi-file databases are not supported yet.
3. `"stdout"` may be used as a value of `<filename>` for the `-B` option.

A user manual for `NBak`/`NBackup` has been prepared. It can be downloaded from the documentation area at the Firebird website: http://www.firebirdsql.org/pdfmanual/ - the file name is `Firebird-nbackup.pdf`.

Source: Firebird 2.0.4 Release Notes: Command-line utilities

GBAK and GSPLIT

(`GBAK.EXE` and `GSPLIT.EXE`)

`GBAK` is an InterBase/Firebird command-line utility, which can be used to back up and restore databases. `GSPLIT` backs up and restores multiple file databases. Please refer to `GBAK` - Firebird backup and restore for further information.

The parameters and options offered by `GBAK` can be found in the IBExpert Backup Database and Restore Database menus.

Many thanks to Stefan Heymann (http://www.destructor.de) for the following overview of options and examples.

`GBAK` is Firebird's/InterBase's command-line tool for online backup and [Restore Database | restore]] of a complete database.

General Syntax

```
gbak <options> -user <username> -password <password> <source> <destination>
```

Backup

For backups, `<source>` is the database you want to back up, `<destination>` is the file name of the backup file. The usual extension is `.fbk` for Firebird and `.gbk` for InterBase.

Only the SYSDBA or the database owner can perform a backup. For multi-file databases, specify only the name of the first file as the database name.

Restore

For restores, `<source>` is the backup file and `<destination>` is the name of the database that is to be built up from the backup file. You will have to specify the `-c` option for restore. Please note that if you run the `GBAK` restore in verbose mode, it can take an awful long time.

For new and altered Firebird 2 parameters, please refer to: Firebird 2.0.4. Release Notes: `gbak` Backup/Porting/Restore Utility.

Options

(Parts in square brackets are optional)

`-b[ackup_database]`	Back up. This switch is optional.	Backup only
`-bu[ffers]`	Set cache size for restored database.	Restore only
`-c[reate_database`	Restore (mandatory).	Restore only
`-co[nvert]`	Converts external tables to internal tables.	Backup only
`-e[xpand]`	Creates an uncompressed backup.	Backup only
`-fa[ctor] n`	Blocking factor for tape device.	Backup only
`-g[arbage collect]`	Does not perform garbage collection (sweeping) during backup.	Backup only
`-i[nactive]`	All indices will be restored as `INACTIVE`.	Restore only
`-ig[nore]`	Ignores checksum errors while backing up.	Backup only
`-k[ill]`	Does not create shadows that are defined in the backup.	Restore only
`-l[imbo]`	Ignores Limbo transactions while backing up.	Backup only
`-m[etadata]`	Only backs up metadata (schema). No table data will be stored.	Backup only
`-mo[de] read_write`	Restores to a read/write database (This is the default).	Restore only
`-mo[de] read_only`	Restores to a read-only database.	Restore only
`-n[o_validity]`	Does not restore validity constraints. So you can restore data that does not meet these constraints and could not be restored otherwise.	Restore only
`-nt`	Non-transportable format (use only when you know you will restore on same platform and database version).	Backup only
`-o[ne_at_a_time]`	Restores one table at a time. You can use this to partially restore databases with corrupt table data.	Restore only
`-ol[d_descriptions]`	Old-style format.	Backup only
`-p[age_size] <size>`	Sets page size of new database. `<size>` can be one of 1024, 2048, 4096, 8192. Default is 1024.	Restore only
`-pa[ssword] <password>`	Database password.	
`-r[eplace_database]` *	Restores over an existing database. This can only be performed by the SYSDBA or the owner of the database hat is overwritten. Do NOT restore over a database that is in use!	Restore only
`-role <role>`	Connect as role.	
`-se[rvice] <hostname>:service_mgr`	*Backup:* creates the backup file on the database server, using the Service Manager. *Restore:* creates the database from a backup file on the server, using the Service Manager.	
`-t[ransportable]`	Creates a transportable backup (transportable between platforms and server versions).	Backup only
`-u[ser] <username>`	Database user name.	

-use_[all_space]	Normally, on restore, database pages will be filled to about 80 %. With the use_all_space option, database pages will be filled to 100 %. (Useful for read-only databases which will see no more modifications).	Restore only
-v[erbose]**	Verbose output of what GBAK is doing.	
-y <filename>	Redirect all output messages to <filename>. *NOTE:* the file must not exist before running GBAK!	
-y suppress_output	Quiet mode.	
-z	Show GBAK version and server version.	

***New to Firebird 2.0: Change to gbak -R semantics**

An important change has been done to prevent accidental database overwrites as the result of users mistakenly treating -R as an abbreviation for restore. gbak -R was formerly a shortcut for -REPLACE_DATABASE. Now the -R switch no longer restores a database by overwriting an existing one, but instead reports an error. If you actually want the former behaviour, you have two alternatives:

- Specify the full syntax gbak -REPLACE_DATABASE. There is a new shortcut for the -REPLACE_DATABASE switch: gbak -REP

or

- Use the new command -R[ECREATE_DATABASE] OVERWRITE. The -R shortcut now represents the -R[ECREATE_DATABASE] switch and the OVERWRITE keyword must be present in either the full or the abbreviated form.

Warning: If you use the full syntax, you are expected to know what this restore mode actually means and have some recovery strategy available if the backup subsequently turns out to be unrestorable.

**** New to Firebird 2.0: gbak -V and the counter parameter**

During Firebird 1 development, an optional numeric <counter> argument was added to the -V[erbose] switch of gbak for both backup and restore. It was intended to allow you to specify a number and get a running count of rows processed as the row counter passed each interval of that number of rows. It caused undesirable side-effects and was removed before Firebird 1.0 was ever released. So, although it never happened, it was documented as "implemented" in the release notes and other places.

GBAK Examples

A "normal" backup:

```
gbak -v -t -user SYSDBA -password "masterkey" dbserver:/db/warehouse.fdb c:\backups\warehouse.fbk
```

Backup with output to a logfile:

```
gbak -v -t -user SYSDBA -password masterkey -y c:\backups\warehouse.log dbserver:/db/warehouse.fdb  c:\backups\warehouse.fbk
```

A "normal" restore:

```
gbak -c -v -user SYSDBA -password masterkey c:\backups\warehouse.fbk dbserver:/db/warehouse2.fdb
```

Restore to an already existing database:

```
gbak -c -r -v -user SYSDBA -password masterkey c:\backups\warehouse.fbk dbserver:/db/warehouse.fdb
```

Create a read-only database:

```
gbak -c -v -mode read_only -use_all_space -user SYSDBA -password masterkey c:\backups\warehouse.fbk c:\files\warehousedb.fdb
```

Multi-file backups

Syntax for backup:

```
gbak [options] <database> <target file 1> <size 1> <target file 2> <size 2> ... <target file n>
```

NOTE: Do not specify a size for the last file. It will always be filled to take up what is left over, no matter how large. Size can be given in bytes (8192), kilobytes (1024k), megabytes (5m), or gigabytes (2g)

Syntax for restore:

```
gbak -c [options] <source file 1> <source file 2> ... <source file n> <database>
```

Restoring to a multi-file database

```
gbak -c [options] <source file> <db file 1> <size 1> <db file 2> <size 2> ... <db file n>
```

NOTE: do not specify a size for the last database file. It can always grow unlimited to take up the rest. Size can be given in bytes (8192), kilobytes (1024k), megabytes (5m), or gigabytes (2g) Restoring from a multi-file backup to a multi-file database:

```
gbak -c [options] <source file 1> <source file 2> ... <source file n> <db file 1> <size 1> <db file 2>  <size 2> ... <db file n>
```

See also:
Why is a database backup and restore important?
Firebird 2.0.4. Release Notes: gbak Backup/Porting/Restore Utility

GFIX

(GFIX.EXE)

GFIX is an InterBase/Firebird command-line utility, offering a number of options to validate and repair databases. These options are included in the IBExpert menu items Services / Database Validation and Database Properties.

The following articles are published here with the kind permission of Stefan Heymann (http://www.destructor.de/).

General Syntax

```
gfix [options] -user <username> -password <password> <database> [options]
```

Should your database ever suffer from corruption, we recommend taking the following procedure:

- Copy your database file somewhere safe: `employee.gdb database.gdb`
- Validate database: `gfix -v -full database.gdb`
- On error try to mend: `gfix -mend -full -ignore database.gdb`
- Check again: `gfix -v -full database.gdb`
- On error try backup without garbage collection: `gbak -backup -v -ignore -garbage database.gdb database.gbk`
- Finally try a restore: `gbak -create -v database.gbk database.gdb`

GBAK - Firebird backup and restore

Further information and examples can be found under the following subjects:

- Database shutdown using GFIX
- Database repair and sweeping using GFIX
- GFIX - miscellaneous parameters
- Using GFIX
- New to Firebird 2

Database shutdown using GFIX

by Stefan Heymann.

Database Shutdown

When a database has been shut down, only SYSDBA and the database owner are able to connect to the database in order to perform administrative tasks.

Options

-at[tach] <seconds>	Used with the -shut option. Waits <seconds> seconds for all current connections to end. If after <seconds> seconds there are still connections open, the shutdown will be cancelled.
-f[orce] <seconds>	Used with the -shut option. Waits <seconds> seconds for all connections and transactions to end. After this time, all connections and transactions are cancelled and the database is shut down. Use with caution.
-o[nline]	If a -shut operation is pending, it is cancelled. Otherwise, takes a database back online.
-sh[ut]	Shut down database. Must be used together with -attach, -force or -tran.
-tr[an] <seconds>	Used with the -shut option. Waits <seconds> seconds for all running transactions to end. If after <seconds> seconds there are still running transactions, the shutdown will be cancelled.

Examples

Shut down database, wait 60 seconds until all connections are closed:

```
gfix -user SYSDBA -password "masterkey" dbserver:/db/mydb.fdb -shut -attach 60
```

Note that GFIX will terminate with an error if there are still connections open after 60 seconds.

Shut down database, force shutdown after 60 seconds:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -shut -force 60
```

Shut down database, force shutdown NOW:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -shut -force 0
```

Put database online again:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -online
```

Database repair and sweeping using GFIX

by Stefan Heymann.

Options

-f[ull]	Use with the -v option. Examines all records and pages and releases unassigned record fragments.
-h[ousekeeping] 0	Switch off automatic sweeping.
-h[ousekeeping] <n>	Set sweep interval to <n> transactions (default is 20000).
-i[gnore]	Ignores checksum errors during a validate or sweep.
-m[end]	Marks corrupt records as unavailable so they are skipped on a subsequent backup.
-n[o_update]	Use with the -v option. Examines all records and pages and reports errors but does not repair them.
-s[weep]	Forces an immediate sweep.
-v[alidate]	Check database for validity. At the same time, errors are reported and repaired.

Examples

Validate database:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -v -f
```

Sweep database now:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -s
```

Set sweep interval to 50000 transactions:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -h 50000
```

Switch off automatic sweeping:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -h 0
```

See also:
Repairing a corrupt database
Database sweep/Sweep interval
Firebird for the Database Expert: Episode 4 - OAT, OIT and Sweep

GFIX - miscellaneous parameters

by Stefan Heymann.

Options

-b[uffers] <pages>	Default cache buffers for the database will be set to <pages> pages.
-c[ommit] <id>	Commits limbo transaction specified by the given <id>.
-c[ommit] all	Commits all limbo transactions.
-k[ill]	Drops shadows and unavailable shadows. Syntax is gfix -k (no database name).
-l[ist]	Display IDs of all Limbo transactions and what would happen to each transaction if you would use -t on it.
-mo[de] read_write	Set mode of database to read/write (default). Requires exclusive access to database (shutdown).
-mo[de] read_only	Set mode of database to read-only. Requires exclusive access to database (shutdown).
-pa[ssword] <password>	Database password.
-p[rompt]	Use with -l. Prompts for action.
-r[ollback] <id>	Rolls back limbo transaction specified by the given <id>.
-r[ollback] all	Rolls back all limbo transactions.
-s[ql_dialect] 1	Sets SQL dialect 1 for the database.
-s[ql_dialect] 3	Sets SQL dialect 3 for the database.
-t[wo_phase] <id>	Performs automated two-phase recovery for limbo transaction with the given <id>.
-t[wo_phase] all	Performs automated two-phase recovery for all limbo transactions.
-user <name>	Database username.
-w[rite] sync	Enables Forced Writes.
-w[rite] async	Disables Forced Writes.
-z	Show GFIX and server version.

Examples

Set database to read-only:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -shut -attach 60g
```

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -shut -force 0

gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -mode read_only

gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -online
```

Set database to SQL dialect 3:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -sql_dialect 3
```

Enable forced writes:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -write sync
```

Disable forced writes:

```
gfix -user SYSDBA -password masterkey dbserver:/db/mydb.fdb -write async
```

See also:
Database Corruption

New `GFIX -shut[down]` options in Firebird 2

The options for `gfix -shut[down]` have been extended to include two extra states or modes to govern the shutdown.

Syntax

```
gfix <command> [<state>] [<options>]
<command> ::= {-shut | -online}
<state> ::= {normal | multi | single | full}
<options> ::= {-force <timeout> | -tran | -attach}
```

- **`normal` state:** online database.
- **`multi` state:** multi-user shutdown mode (the legacy one, unlimited attachments of SYSDBA/owner are allowed).
- **`single` state:** single-user shutdown (only one attachment is allowed, used by the restore process).
- **`full` state:** full/exclusive shutdown (no attachments are allowed).

Note: `Multi` is the default state for `-shut`, `normal` is the default state for `-online`.

The modes can be switched sequentially:

```
normal <-> multi <-> single <-> full
```

Examples

```
gfix -shut single -force 0
gfix -shut full -force 0
gfix -online single
gfix -online
```

You cannot use `-shut` to bring a database one level more "online" and you cannot use `-online`

to make a database more protected (an error will be thrown).

These are prohibited:

```
gfix -shut single -force 0
gfix -shut multi -force 0
gfix -online
gfix -online full
gfix -shut -force 0
gfix -online single
```

Source: Firebird 2.0.4 Release Notes: `gfix` server utility

GSEC

(`GSEC.EXE`)

`GSEC` is an InterBase/Firebird command-line utility, which manages server security. It can be used to add, modify, and delete authorized users on the server. `GSEC` commands apply to the database server and not to individual databases, as with the majority of other command-line utilities.

All options offered by `GSEC` can be found in the IBExpert User Manager and Grant Manager.

Many thanks to Stefan Heymann (http://www.destructor.de) for the following overview of commands, options and examples.

All database users are stored in the security database named `security.fdb` (since Firebird 2 this file is now called `security2.fdb`) in the Firebird directory. There is at least one user, the system database administrator, SYSDBA.

After installation, the SYSDBA password is *masterkey*. (Exception: Firebird 1.5 for Linux). Only the first 8 characters of a password are significant. The password should not contain space characters.

Invoking GSEC

`GSEC` can only be run by the SYSDBA.

To use `GSEC` for the local machine, use:

```
gsec -user sysdba -password <password> [options]
```

To use `GSEC` for a remote machine, use:

```
gsec -user sysdba -password <password> -database <databasename>
```

where `<databasename>` is the database name of the remote `security.fdb` database.

You can use `GSEC` as an interactive command line tool or give all commands on one command line.

Commands

`di[splay]`	Displays all users.
`di[splay] <username>`	Displays all information for the given user.
`a[dd] <username> -pw <password> [options]`	Add a new user.
`mo[dify] <username> [options]`	Modify user.
`de[lete] <username>`	Delete user.
`h[elp]`	Display help.
`?`	Display help.
`q[uit]`	Quit interactive mode.
`z`	Display GSEC version number.

If you don't want to invoke the interactive mode, you can enter all commands directly in the command line. To do that, precede the commands with a dash.

Options

`-pa[ssword] <password>`	Password of the user who is performing the change.
`-user <username>`	User name of the user who is performing the change.
`-pw <password>`	Password of target user (or new password).
`-fname <first name>`	Target user's first name.
`-mname <middle name>`	Target user's middle name.
`-lname <last name>`	Target user's last name.

Examples

Add user Elvis Presley as user `ELVIS`, password is "Aaron":

```
gsec -user SYSDBA -password masterkey
GSEC> add elvis -pw Aaron -fname Elvis -lname Presley
GSEC> quit
```

Change password of user `ELVIS` to `chuck`":

```
gsec -user SYSDBA -password masterkey
GSEC> modify elvis -pw chuck
GSEC> quit
```

Change password of SYSDBA on remote Linux server `harry` to `hamburg`:

```
gsec -user SYSDBA -password masterkey -database harry:/opt/firebird/security.fdb -modify sysdba -pw hamburg
```

Change password of SYSDBA on remote Windows server `sally` to `hannover`:

```
gsec -user SYSDBA -password masterkey -database sally:"C:\Program Files\Firebird\security.fdb" -modify sysdba -pw hannover
```

Change password of SYSDBA on remote server `jake` on TCP port 3051 to `london`:

```
gsec -user SYSDBA -password masterkey -database "jake/3051:/opt/firebird/security.fdb" -modify sysdba -pw london
```

Delete user `Joe` on local server:

```
gsec -user SYSDBA -password masterkey -delete joe
```

Notes: On InterBase systems, the security database is named `isc4.gdb`. There will be a warning when a new password is longer than 8 characters.

See also:
Security in Firebird 2
User Manager
Grant Manager

GSTAT

(`GSTAT.EXE`)

`GSTAT` is an InterBase/Firebird command-line utility, which can be used to display database statistics related to transaction inventory, data distribution within a database, and index efficiency.

All information offered by this tool can be found in the IBExpert Services menu item, Database Statistics.

IBLOCKPR (Windows) and GDS_LOCK_PRINT (Unix)

`IBLOCKPR.EXE` on Windows and `gds_lock_print` on UNIX.

These utilities display statistics for the InterBase Lock Manager.

IBMGR

(`IBMGR.EXE`)

`IBMGR` is a windows-based server management program, and includes the functionalities found in `GSEC`, `GBAK` and `GFIX`.

ISQL - Interactive SQL

`ISQL` is a command-line utility program which can be used to run SQL queries on the database. `ISQL` supports data definitions and data manipulation commands as well as SQL scripts with multiple SQL commands within one script. It can be used to create and modify the database's metadata, insertion, alteration and deletion of data, data queries and the display of results (all this can be done in the IBExpert SQL Editor), adding and removal of user database rights (see the IBExpert User Manager and Grant Manager) and execution of other database administrative functions. It is very similar to `DSQL`, with some omissions, such as cursors, and a few additions, for example, `SET` and `SHOW`.

`ISQL` commands end with `;`. Each command must be explicitly committed using the commit statement.

For new features and switches introduced in Firebird 2 please refer to Firebird 2.0.4. Release Notes: ISQL query utility

See also:
Firebird 2 SQL Reference Guide
Firebird administration

IBExpert Services menu

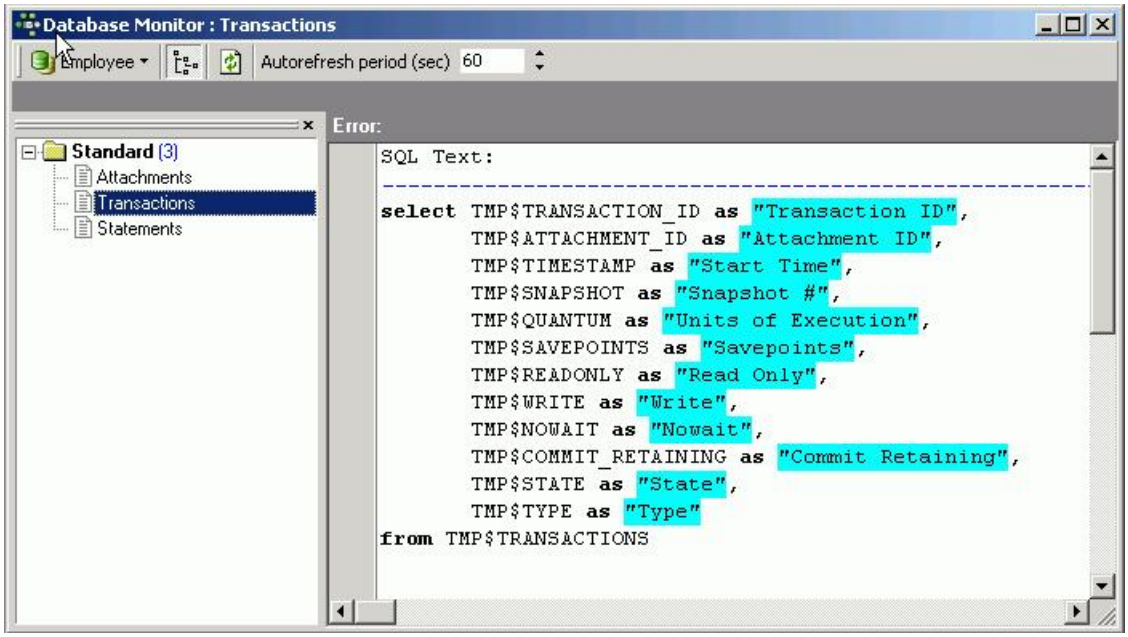
The IBExpert Services menu offers the following range of services:

- Backup Database
- Restore Database
- Server Properties/Log
- Server Activation Certificates
- Database Validation
- Database Statistics
- Database Properties
- Database Shutdown
- Database Online
- Communication Diagnostics
- HK-Software Services Control Center

Database monitoring

Database monitoring

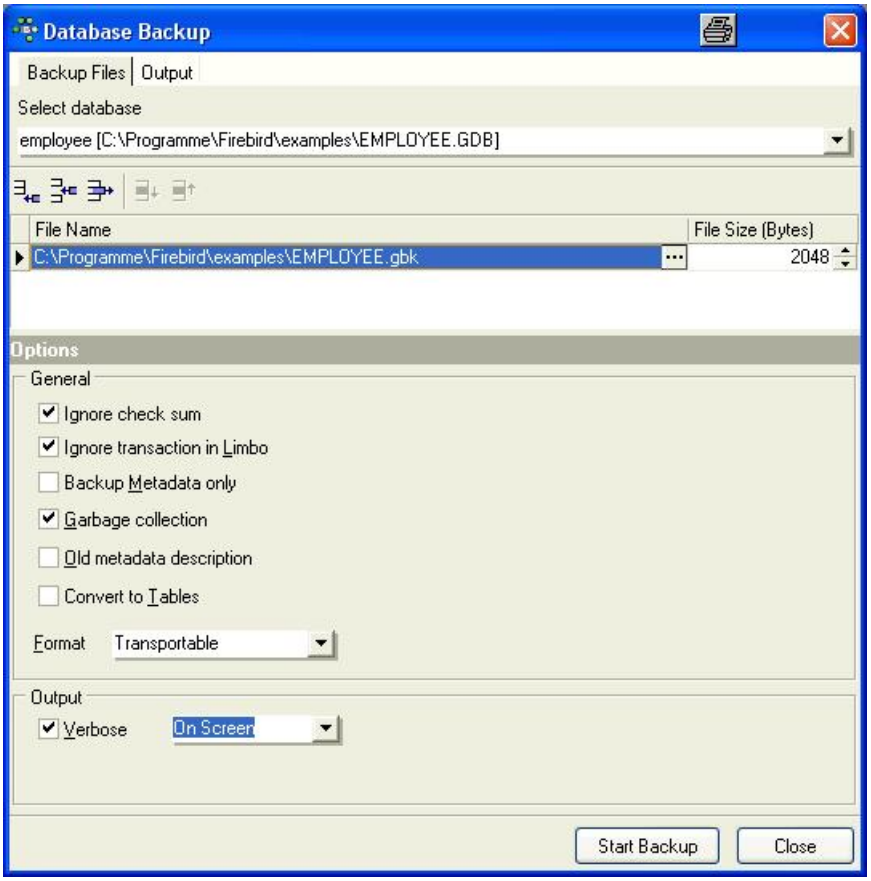
... coming soon.


Database Monitor : Transactions
Employee
Autorefresh period (sec) 60
Standard (3)
Attachments
Transactions
Statements
Error:
SQL Text:
select TMP$TRANSACTION_ID as "Transaction ID",
TMP$ATTACHMENT_ID as "Attachment ID",
TMP$TIMESTAMP as "Start Time",
TMP$SNAPSHOT as "Snapshot #",
TMP$QUANTUM as "Units of Execution",
TMP$SAVEPOINTS as "Savepoints",
TMP$READONLY as "Read Only",
TMP$WRITE as "Write",
TMP$NOWAIT as "Nowait",
TMP$COMMIT_RETAINING as "Commit Retaining",
TMP$STATE as "State",
TMP$TYPE as "Type"
from TMP$TRANSACTIONS

Backup Database

The IBExpert Services menu item Backup Database allows you to create a backup or copy of the database, saving it to file. This database copy may be kept simply for security reasons, or restored for the reasons detailed in Why is a database backup and restore important?.

A database backup may be performed without having to disconnect the database; users may continue their work as InterBase/Firebird uses its multigenerational architecture to take a snapshot of the database at a moment in time the backup is requested. All information generated by committed transactions and present at this moment, is backed up.



First select the database to be backed up from the pull-down list of registered databases. Then select either an existing backup file name, or add a new backup file using the *Insert File* icon (or [Ins] key).

The [...] button to the right of this row allows you to find an existing file or specify the drive, path and backup file name for a new file. Please note that IBExpert will only create a file name on the server, and not locally (as with GBAK), because IBExpert uses the Services API. A local backup can only be performed using GBAK. The suffixes .GBK and .FBK are traditionally respectively used for InterBase and Firebird backup files. A file size only needs to be specified when working with secondary files?. All files in a multifile database are backed up (i.e. both secondary files and shadow files). InterBase/Firebird understands the links that exist with secondary database files and with shadows. Whereas the operating system backup works on a file-by-file basis, InterBase/Firebird always backs up all files in a database.

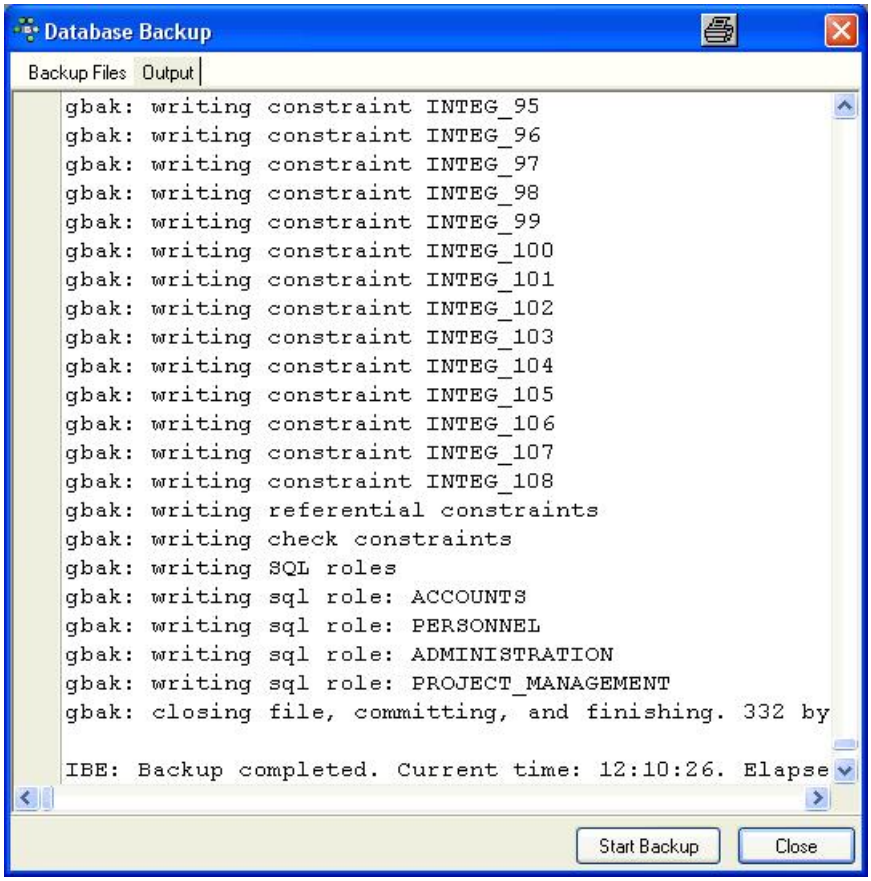
Backup Options

- **Ignore check sum:** If this option is checked, check sum errors in the database header pages, where the database connection properties are stored, are ignored in the backup. As InterBase and Firebird normally abort the backup when check sum errors are discovered, this is a way to force a backup when there are problems. Note that UNIX versions do not use check sums.
- **Ignore transactions in Limbo:** If this option is checked, transactions in limbo, i.e. transactions, that can't be defined as executed or aborted, are ignored in the backup. Only those most recent, committed transactions are backed up. It allows a database to be backed up before recovering corrupted transactions. Generally in limbo transactions should be recovered before a backup is performed.
- **Backup Metadata only:** If this option is checked, only the database's definition (i.e. the metadata, which provides an empty copy of the database) is saved. (If a database copy with certain data content is required, then use the IBExpert Script Executive.)
- **Garbage collection:** If this option is checked, garbage collection is executed during the backup. By disabling this option, the backup can be speeded up considerably. (Refer to garbage collection for further information.)
- **Old metadata description:** If this option is checked, old metadata descriptions are included into the backup database. This is included for compatibility reasons for older InterBase versions.
- **Convert to Tables:** This option converts the database data to tables in the backup. This concerns external files. It is possible in InterBase/Firebird to create a table as an external file - this option converts them to internal database tables.
- **Format**: Select the data format for the backup database file. *Transportable* is the recommended default option, as it allows a restore into different InterBase/Firebird Versions if wished, i.e. it saves the data and metadata to a generic format, as opposed to the option *Non-Transportable*. (Please note that when backing up and restoring, for example, from InterBase 4 to Firebird 1.5, stored procedures are restored as blobs, so that they may not initially work.)

- **Verbose:** Checking *Verbose* provides a detailed protocol of the current database backup process, by writing step-by-step status information to the output log. Select the option *On Screen* or *Into File* (not forgetting to select or specify a file name for this protocol) before starting the backup. This option is useful if the backup is failing and the reason needs to be analyzed.

Then start the backup. If the protocol option *On Screen* was selected, the backup is logged on the *Output* page.

Using the IBExpert menu item Database / Database Registration Info, default backup file names, paths and drives may be specified if wished, along with default backup and restore options. This information may be specified when initially registering a database in IBExpert (see Register Database) or at a later date (see Database Registration Info).



In normal circumstances, the backup should run smoothly without any of the above options having to be checked. If however, corrupt or damaged data is suspected or problems have been encountered, alter the *Format* to *Non-Transportable* and check the options *Ignore Check Sum* and *Ignore Transactions in Limbo*. Although this will not provide the usual database compression, it does provide a complete copy of the database, which is important before starting to repair it.

It is also possible to validate the database using Services / Database Validation or `GFIX`, before retrying.

See also:
Repairing a corrupt database
Restoring a backup to a running database

Why is a database backup and restore important?

Performing regular backups protects from hardware failures and data corruption, which cannot be fixed by the InterBase/Firebird maintenance tools. It is important to use the InterBase/Firebird backup and restore facilities even though most networks include a facility for data backup and restore across the network, because:

- Operating system backups require exclusive access to the database. The InterBase/Firebird backup runs parallel with concurrent database accesses by other users. InterBase/Firebird uses its multigenerational architecture to take a snapshot of the database at a moment in time for the backup. All information generated by committed transactions and present at this moment is backed up.
- All files in a multifile database are backed up. InterBase/Firebird comprehends the links between the different database files and shadows. The operating system backup processes files one after the other and saves them to the specified file or medium, so that all the various files are backed up in different versions and they cannot work together correctly anymore when restored. The InterBase/Firebird backup backs up all database files automatically.
- The different versions of InterBase/Firebird use different database file formats, so that it is impossible to copy a file directly from one operating system environment to the required format of another operating system environment. The InterBase/Firebird backup utility allows a transportable backup format, so that this file can be restored on any desired InterBase/Firebird platform. *Please note:* When backing up and restoring, for example, from InterBase 4 to Firebird 1.5, stored procedures are restored as blobs, so that they may not initially work.
- The InterBase/Firebird backup discards outdated data sets and index files, resulting in a smaller backup (please refer to garbage collection for more information).
- Empty pages are also automatically removed during a backup and restore, which reduces the total database size. The transaction number in the TIP is reset to zero (the total number of transactions that can be recorded in a TIP is approximately 1.3 billion!). The cache works with considerably more

efficiency following a backup and restore as the pages are reordered. *Please note:* In Firebird 1.5 the memory manager allows new data sets to automatically be stored in old pages, without first having to backup and restore.

- During an InterBase/Firebird backup the integrity and references for all database objects, e.g. domains, tables, indices, views, triggers, stored procedures, generators, exceptions, and permissions, are checked.
- Executing a backup and restore is the only way to subsequently alter fundamental parameters in the database structure, such as the page size and distribution across secondary files. It is therefore recommended to not only backup but also restore the database regularly (e.g. once a month).

Garbage collection

When performing a garbage collection, InterBase/Firebird does nothing other than remove outdated data sets and index files, which results in a smaller database. Outdated data sets are stored by InterBase/Firebird for the following reason: InterBase/Firebird are multigenerational databases. When a data set is altered, this alteration is stored in the database as a new copy. The old values remain in the database as a back version, which is the rollback protocol. If the transaction is rolled back after the update, the old value is ready to resume its function as the valid value. If the transaction is however committed, and not rolled back, this back version becomes superfluous. In databases with a lot of update operations this can result in a lot of garbage.

When garbage is collected in InterBase/Firebird, not only the out-of-date update values are deleted, but all outdated and deleted data set versions, based on the Transaction Inventory Page (TIP).

A garbage collection is only performed during a database sweep, database backup or when a `SELECT` query is made on a table (and not by insert, alter or delete). Whenever InterBase touches a row, such as during a `SELECT` operation, the versioning engine? sweeps out any versions of the row where the transaction number is older than the Oldest Interesting Transaction (OIT). This helps to keep the version history small and manageable and also keeps performance reasonable.

The sweep interval (i.e. at what interval (in number of transactions) a database sweep should be automatically conducted) for the garbage collection may be specified under the IBExpert Services menu item Database Properties.

The garbage collection may be performed during 24 hour operation online without any problems (i.e. the server does not need to be shut down). Performance may however be slower during the database sweep which may not be desirable. If the sweep interval is specified at zero (0) (see Database Properties), the garbage collection is not performed automatically at all. It could then be carried out, for example, at night as a sweep or backup using `GFIX` and the `at` Windows command or the Linux `chron` command.

New to Firebird 2.0: Superserver garbage collection changes

Formerly, Superserver performed only background garbage collection. By contrast, Classic performs "cooperative" GC, where multiple connections share the performance hit of GC. Superserver's default behaviour for GC is now to combine cooperative and background modes. The new default behaviour generally guarantees better overall performance as the garbage collection is performed online, curtailing the growth of version chains under high load.

It means that some queries may be slower to start to return data if the volume of old record versions in the affected tables is especially high. ODS10 and lower databases, having ineffective garbage collection on indices, will be particularly prone to this problem. The `GCPolicy` parameter in `firebird.conf` allows the former behaviour to be reinstated if you have databases exhibiting this problem.

See also:
Backup/Restore
Database Properties
Restore Database
InterBase and Firebird command-line utilities: `GBAK`
Firebird 2.0.4 Release Notes: Backup tools
Firebird 2.0.4 Release Notes: Reworking of garbage collection
Recovering a corrupt database
Firebird for the Database Expert: Episode 4 - OAT, OIT and sweep
Garbage Collectors
Firebird administration

Restore Database

The IBExpert Services menu item Restore Database allows you to restore the database from a backed up file.

A database restore is required in the following situations:

- Following approximately 1.3 billion transactions in order to reset the transaction space.
- Following 255 metadata changes on a single table; otherwise no further metadata changes are possible. Please refer to IBExpert Screen*253 changes of table left*? for details.
- When changing the Firebird version you need to backup the old version and restore to the new version number.
- A sweep is also automatically performed during a backup, so long as it has not been disabled.

Before restoring a backup file into a database, it is important to first disconnect the database! - Otherwise you could end up with a corrupt database should users try to log in and perform data operations during the restore.

The *Files* page allows the following specifications:

Restore into: Select to restore into the existing database, or create a new database. When restoring into the existing database, select it from the list of registered databases; if restoring to a new database, then set the database file name not forgetting the drive and path.

Specify the backup file name which is to be restored. The [...] button to the right of this row allows you to find an existing file name, drive, and path. The suffixes `.GBK` and `.FBK` are traditionally respectively used for InterBase and Firebird backup files.

The following restore options may be checked/unchecked as wished:

- **Deactivate indexes:** If this option is checked, database indices are deactivated while restoring. This option is used to improve restore performance. If this option is not checked, InterBase/Firebird updates indices after all tables have been populated with the restored rows. This option may also be necessary if the database contains data with a unique index, but there are values in the table that are not actually unique. It can also be used when the field length in one or more tables is to be altered retrospectively; or when an index is simply not working due to some undiscovered inconsistencies.

- **Don't recreate shadow files:** If this option is checked, shadow files are not recreated while restoring.

- **Restoring without Shadow:** deletes the shadow definition. To restore it, it is necessary to recreate the shadow using the `CREATE SHADOW` statement (please refer to Creating a shadow below for further information). This option is sometimes required if the destination database does not support shadows, if you are migrating from an earlier version of InterBase where shadows were not supported, or if the machine where the shadow resides is not available.

- **Don't enforce validity conditions:** When this option is checked, database validity conditions such as constraints on fields or tables are not restored. This option is useful if the validity constraints were changed after data had already been entered into the database. When a database is restored, InterBase/Firebird compares each row with the metadata; an error message is received if incompatible data is found. Once the offending data has been corrected, the constraints can be added back.

- **Commit after each table:** If this option is checked, IB Manager commits work after restoring each table. This allows all those tables to be restored and committed where there is no corrupted data. It restores metadata and data for each table in turn as a single transaction and then commits the transaction. This option is useful if corrupt data is suspected in the backup file, or if the backup is not running to completion. Normally, InterBase/Firebird restores all metadata and then restores the data. Should you encounter problems when restoring your database, deactivate this option and retry.

- **Replace existing database:** If this option is checked the restored database replaces the existing one. Leaving this option unchecked provides a measure of protection from accidentally overwriting a database file.

- **Use All Space:** This option should be checked when restoring the database onto a CD, as all (i.e. 100%) space is then used, as opposed to the usual 80% for databases which are subject to alterations and stored on hard drives.

- **Metadata Only:** This option produces an empty copy of the database. It may also be used to restore the framework of a corrupt database, to allow analysis and repair work.

- **Client Library:** This is new in version 2003.11.6.1 and is an added possibility to specify a client library which will be used while restoring. This option allows the user to specify whether he requires the InterBase or the Firebird client library for each IBExpert connection. The default client library is `gds32.dll`.

- **Page size:** Database page size in bytes. This is the only option allowing the page size for an existing database to be altered.

- **Verbose:** Check *Verbose* to receive a detailed protocol of the current database backup process, by writing step-by-step status information to the output log. The options *On Screen* or *Into File* (not forgetting to select or specify a file name for this protocol) need to be specified before starting the backup. This option is useful if the restore is failing, and the reason needs to be analyzed.

The restore can then be started. If the protocol option *On Screen* was selected, the backup is logged on the *Output* page.

Under normal circumstances, none of the above restore options should need to be specified. If inconsistencies between the metadata and the data itself are suspected, check the *Commit After Each Table*, *Deactivate Indexes*, and *Don't Enforce Validity Conditions* options.

Please note that InterBase/Firebird does not backup indices. It only backs up the index definition. When the database is restored InterBase/Firebird uses this definition to regenerate the indices.

Using the Database Registration dialog, default backup file names, paths and drives may be specified if wished, along with default backup and restore options. This information may be specified when initially registering a database in IBExpert (see Register Database) or at a later date (see Database Registration Info).

Empty pages are automatically removed during a backup and restore, which reduces the total database size.

The transaction number in the TIP is reset to zero. The cache works with considerably more efficiency following a backup and restore as the pages are reordered. It is therefore recommended not only to backup but also to restore the database regularly (e.g. once a month).

In Firebird 1.5 the new memory manager allows new data sets to automatically be stored in old pages, without first having to backup and restore.

Database Shadow Files

Shadow files are an exact live copy of the original active database, allowing you to maintain live duplicates of your production database, which can be brought into production in the event of a hardware failure. These shadows are administrated in real time by the InterBase/Firebird server. They are used for security reasons: should the original database be damaged or incapacitated by hardware problems, the shadow can immediately take over as the primary database. It is therefore important that shadow files do not run on the same server or at least on the same drive as the primary database files. Shadow files are not normally used on Windows platforms, as the shadow file has to be on the same computer as the active database. These do work however on LINUX/UNIX.

InterBase allows up to 65,536 (216) database files, including shadow files. However the operating system used may have a lower limit on the number of simultaneous open files that the IBServer/FBServer can have. In some cases, the OS provides a means to raise this limit (refer to your OS documentation for the default open files limit, and the means to raise it).

Shadow files, as with the main database and secondary files, may not reside on networked or remote file systems (i.e. mapped drives on Windows and NFS files on UNIX).

The number of existing shadow files in a database may be ascertained using the IBExpert Services menu item Database Statistics, or using GSTAT (the shadow count is included in the database header page information).

Shadowing offers a number of advantages:

- It provides valuable protection of the database, in addition to the regular backups which should be maintained, and in addition to InterBase/Firebird's multigenerational architecture.
- If the original database is damaged, the shadow can be activated immediately, with little lost time.
- Shadowing runs automatically with little or no maintenance.
- You have full control over the shadow's configuration, including its use of hard disk space and distribution across other available devices.
- Creating a shadow does not require exclusive access to the database.
- Shadow files use the same amount of disk space as the database. As opposed to log files, which can grow well beyond the size of the database.
- Shadowing does not use a separate process. The database process handles writing to the shadow.

But there are also some limitations:

- Shadowing only helps to recover from certain types of problems. If a user error or InterBase/Firebird problem causes the database to be damaged beyond recovery, then the shadow is identically damaged. But if the database is accidentally deleted by the user, or a hardware problem on the primary server occurs, the shadow remains intact and can be used immediately.
- Shadowing is not replication. It is one-way writing, duplicating every write operation on the master database. Client applications cannot access the shadow file directly.
- The shadow cannot be used to rollback the database to a specific point in time. When the shadow is used to recover the database, everything up to the point where the original problem occurred is retrieved.
- Shadowing adds a small performance penalty to database operations. Every action on the database which modifies metadata or the data itself is mirrored in the shadow.
- Shadowing does not replace a careful security system within the operating system, but is one aspect or enhancement of the whole.
- Shadowing also works only for operations that go through the InterBase/Firebird database services manager (GDS), which processes all SQL and database requests.
- Shadowing can occur only to a local disk. Shadowing to a NFS file system or mapped drive is not possible.

- Shadowing to tape or other media is also not possible.

Tasks for Shadowing

The main tasks in setting up and maintaining shadows are as follows:

Creating a shadow

(Source: InterBase® 7.1 Operations Guide)

Shadowing begins with the creation of a shadow, using the `CREATE SHADOW` statement. This statement has the following syntax:

```
CREATE SHADOW shadow_number
[AUTO | MANUAL] [CONDITIONAL] shadow_filename
```

The shadow number identifies a shadow set that collects the primary shadow file and any secondary files together. The most important function of the shadow number is to identify the shadow if you decide to drop it (please refer to Deleting a shadow).

This can be performed without affecting users at all, as it does not require exclusive access. Before creating the shadow, the following should be considered:

1. **Shadow location:** a shadow should be created on a different disk from the main database, as shadowing is intended as a recovery mechanism in case of disk failure. Therefore storing the main database and the shadow on the same disk defeats the whole purpose of shadowing!
2. **Distributing the shadow:** a shadow can be created as a single-file (shadow file) or as multiple files (shadow set). To improve space allocation and disk I/O, each file in a shadow set may be placed on a different disk.
3. **User access:** if a shadow becomes unavailable, user access to the database can be denied until shadowing is resumed, or access can be allowed (i.e. work can continue as normal) although any changes made during this period will obviously not be shadowed. Please refer to auto mode and manual mode for further information.
4. **Automatic shadow creation:** To ensure that a new shadow is automatically created, create a conditional shadow (details below).

Please note: If the IBExpert Services menu item Restore Database dialog option, *Don't Recreate ShadowFiles* is checked, shadow files are not recreated while restoring. This deletes the shadow definition; and to restore it, it is necessary to recreate the shadow using the `CREATE SHADOW` statement. This option is sometimes required if the destination database does not support shadows, if you are migrating from an earlier version of InterBase where shadows are not supported, or if the machine where the shadow resides is not available.

The following sections deal with the creation of shadows with various options:

- Creating Single-file or Multifile Shadows
- Auto Mode and Manual Mode
- Conditional Shadows

These options are not mutually exclusive, e.g. it is possible to create a single-file conditional shadow with the option manual mode.

Creating single-file or multifile shadows

(Source: InterBase® 7.1 Operations Guide)

To create a single-file shadow for the sample database `employee.gdb`, enter the following in the IBExpert SQL Editor:

```
CREATE SHADOW 1 '/usr/interbase/examples/employee.shd';
```

The name of the shadow file is `employee.shd`, and it is identified by the number `1`. It is possible to verify that the shadow has been created by using the `isql` command:

```
SHOW DATABASE;
   Database: employee.gdb
   Shadow 1: '/usr/interbase/examples/employee.shd' auto
   PAGE_SIZE 4096
   Number of DB pages allocated = 392
   Sweep interval = 20000
```

The page size of the shadow is the same as that of the database. A large database may be shadowed to a multifile shadow if wished, spreading the shadow files over several disks. Each file in the shadow set needs to be specified by name and size. This can be specified in two ways, the same as with multifile databases:

- Specify the page on which each secondary file starts
- Specify the length in database pages of each file.

You can specify both but this is redundant. If the information specified is inconsistent, InterBase/Firebird uses the length value in preference to the starting page value. In general, it is best to use either length values or starting page number to ensure consistency or legibility.

If the files are specified using the `LENGTH` keyword, do not specify the length of the final file, as InterBase/Firebird sizes the final file dynamically, as needed. Please refer to secondary files for further information.

The following example creates a shadow set consisting of three files. The primary file, `EMPLOYEE.SHD` is 10,000 database pages in length; the second file is 20,000 pages long, and the final file is left open, to expand as needed.

```
CREATE SHADOW 1 'employee.shd' LENGTH 10000
   FILE 'emp2.shd' LENGTH 20000
   FILE 'emp3.shd';
```

The second alternative is to specify the starting page of the files:

```
CREATE SHADOW 1 'employee.shd'
   FILE 'emp1.shd' STARTING AT 10000
   FILE 'emp2.shd' STARTING AT 30000;
```

Using the `SHOW DATABASE` command, the file names, page lengths or starting pages can be verified:

```
SHOW DATABASE;
   Database: employee.gdb
   Shadow 1: '/usr/interbase/examples/employee.shd' auto length 10000
   file /usr/interbase/examples/emp1.shd length 2000 starting 10000
   file /usr/interbase/examples/emp2.shd length 2000 starting 30000
```

```
PAGE_SIZE 4096
Number of DB pages allocated = 392
Sweep interval = 20000
```

The page length for secondary files in the main database does not need to correspond to the page length for the secondary shadow files. As the database grows and its first shadow file becomes full, updates to the database automatically overflow into the next shadow file.

Auto mode and manual mode

(Source: InterBase® 7.1 Operations Guide)

A shadow database may become unavailable for the same reasons a database becomes unavailable (e.g. disk failure, network failure, or accidental deletion) . If a shadow has been created in auto mode and suddenly becomes unavailable, database operations continue automatically without shadowing. If the shadow was created in manual mode, further access to the database is denied until the database administrator gives explicit instructions, as to how work is to be continued.

The benefits of auto mode and manual mode may be compared below:

Mode	Advantage	Disadvantage
Auto	Database operation is uninterrupted.	Creates a temporary period when the database is not shadowed. The database administrator might be unaware that the database is operating without a shadow.
Manual	Prevents the database from running unintentionally without a shadow.	Database operation is halted until the problem is fixed. Needs intervention of the database administrator.

Auto mode

The `AUTO` keyword can be used to create a shadow in auto mode:

```
CREATE SHADOW 1 AUTO 'employee.shd';
```

Auto mode is the default, so this does not necessarily need to be specified explicitly.

In auto mode, database operation is uninterrupted even though there is no shadow. To resume shadowing, it might be necessary to create a new shadow. If the original shadow was created as a conditional shadow, a new shadow is automatically created. Please refer to conditional shadows for further information.

Manual mode

The `MANUAL` keyword can be used to create a shadow in manual mode:

```
CREATE SHADOW 1 MANUAL 'employee.shd';
```

Manual mode is useful when continuous shadowing is more important than continuous operation of the database. When a manual-mode shadow becomes unavailable, further operations on the database are prevented.

To allow work on the database to be resumed, the database owner or SYSDBA must enter the following command:

```
gfix -kill database
```

This command deletes metadata references to the unavailable shadow corresponding to the database. After deleting the references, a new shadow can be created if shadowing needs to be resumed.

Shadow information is kept in the metadata of the primary database file. If this file becomes unavailable for some reason, then the pointers to the shadow are also broken. In this situation, the database administrator can use the `-active` option in the `GFIX` utility to convert the original shadow into a new primary database.

Conditional shadows

(Source: InterBase® 7.1 Operations Guide)

A shadow may be defined so that if it replaces a database, the server creates a new shadow file, and thus allows shadowing to continue uninterrupted. This is termed a conditional shadow, and is specified using the `CONDITIONAL` keyword:

```
CREATE SHADOW 3 CONDITIONAL 'atlas.shd';
```

Creating a conditional file automatically creates a new shadow in either of two situations:

- The database or one of its shadow files becomes unavailable.
- The shadow takes over for the database due to hardware failure.

Activating a shadow

(Source: InterBase® 7.1 Operations Guide)

Should the main database become unavailable for whatever reason, the shadow can be activated, i.e. it takes over the main database and all users now access the shadow as the main database. This activation may be defined to occur automatically or through the intervention of the database administrator.

Shadow information is kept in the metadata of the primary database file. If this file becomes unavailable for some reason, then the pointers to the shadow are also broken. To activate the shadow it is necessary to log in as SYSDBA or the database owner, and use GFIX with the `-activate` option, to convert the original shadow into a new primary database.

Important! The first step is to make sure the shadow is not active, i.e. if the main database has active transactions the shadow is active. Also check that the main database is unavailable. If a shadow is activated while the main database is still available, the shadow can be corrupted by existing attachments to the main database.

To activate a shadow, specify the path name of its primary file. For example, if database `employee.gdb` has a shadow named `employee.shd`, enter:

```
gfix -a[ctivate] shadow_name
```

The shadow name is the explicit path and name of the shadow's primary file.

Examples

For a Windows NT server:

```
gfix -a F:\SHADOW\ORDENT\ORDERS.SHD
```

For any UNIX server:

```
gfix -a /usr/shadow/ordent/orders.shd
```

After a shadow is activated its name should be changed to the name of the original database. Then a new shadow can be created if shadowing needs to continue providing another disk drive is available.

Deleting a shadow

(Source: InterBase® 7.1 Operations Guide)

If a shadow is no longer needed, it can be stopped by simply deleting it. To stop shadowing, use the shadow number as an argument with the DROP SHADOW statement. For example:

```
DROP SHADOW 1
```

If you need to look up the shadow number, use the `isql` command `SHOW DATABASE`.

Important! `DROP SHADOW` deletes all shadow references from a database's metadata as well as the physical files on disk. Once the files have been removed from the disk, there is no way to recover them. However, as a shadow is merely a copy of an existing database, a new shadow will be identical to the dropped shadow.

Adding files to a shadow/modifying a shadow

(Source: InterBase® 7.1 Operations Guide)

Shadow databases may consist of multiple files. As the shadow grows in size, files may need to be added to cope with the increase in space requirements.

To modify a shadow database or add a shadow file, first use the DROP SHADOW statement to delete the existing shadow, then use the CREATE SHADOW statement to create a multifile shadow.

Example

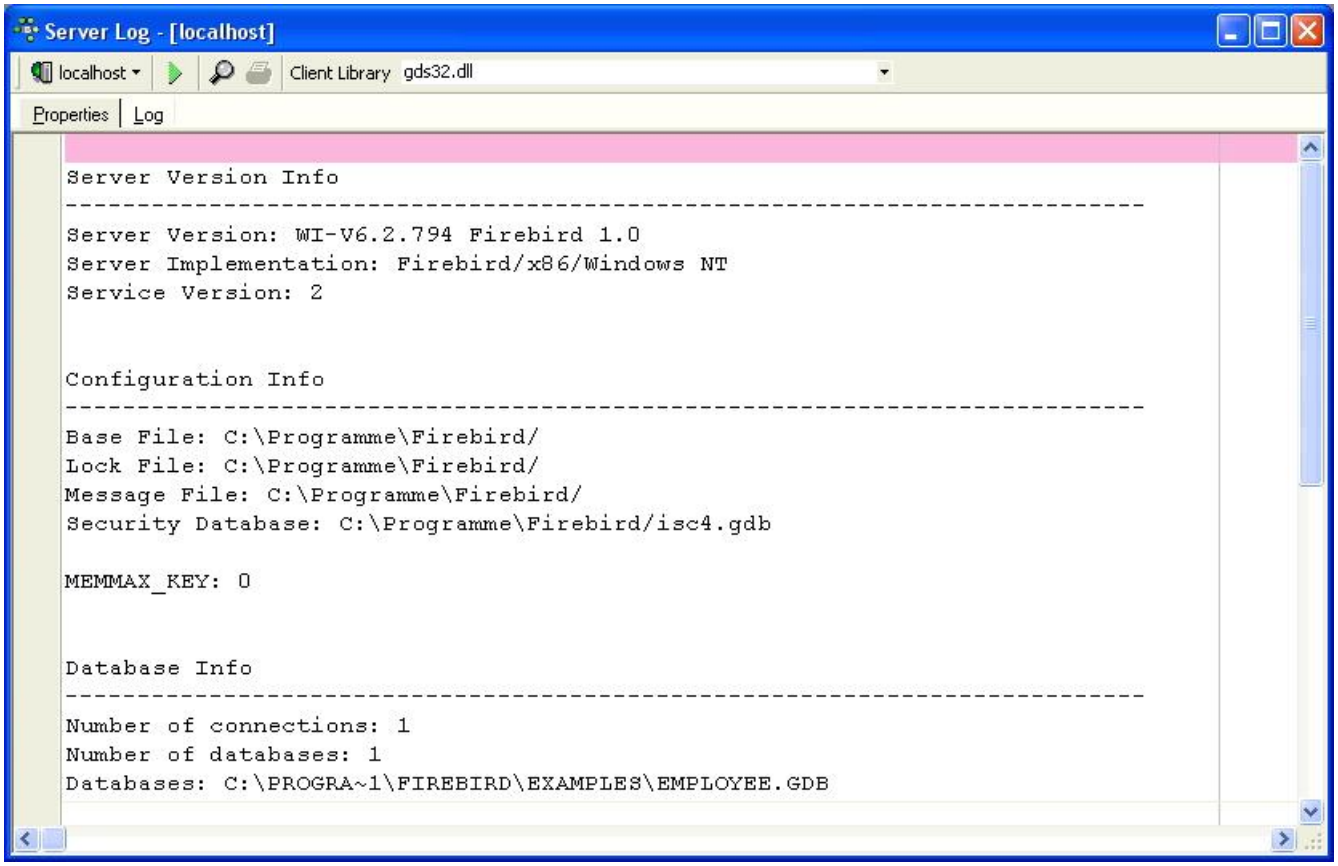
```
DROP SHADOW 2
CREATE SHADOW 3 AUTO CONDITIONAL
'F:\SHADOW\ORDENT\ORDERS.SHD' LENGTH 10000
FILE 'F:\SHADOW\OIRDENT\ORDERS2.SHD'
```

The page length allocated for secondary shadow files need not correspond to the page length of the database's secondary files. As the database grows and its first shadow file becomes full, updates to the database automatically overflow into the next shadow file.

See also:
Allowing users to login during a restore
Backup Database
Backup/Restore
GBAK
Repairing a corrupt database
Why is a database backup and restore important?
Firebird administration

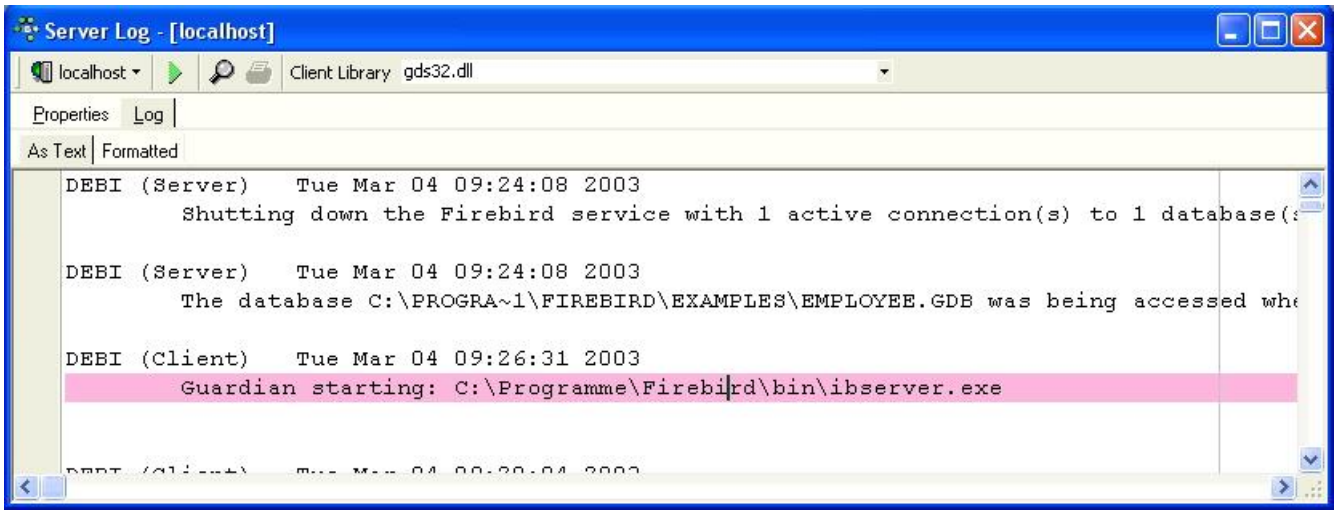
Server Properties / Log

The *Server Properties* page displays the following information:

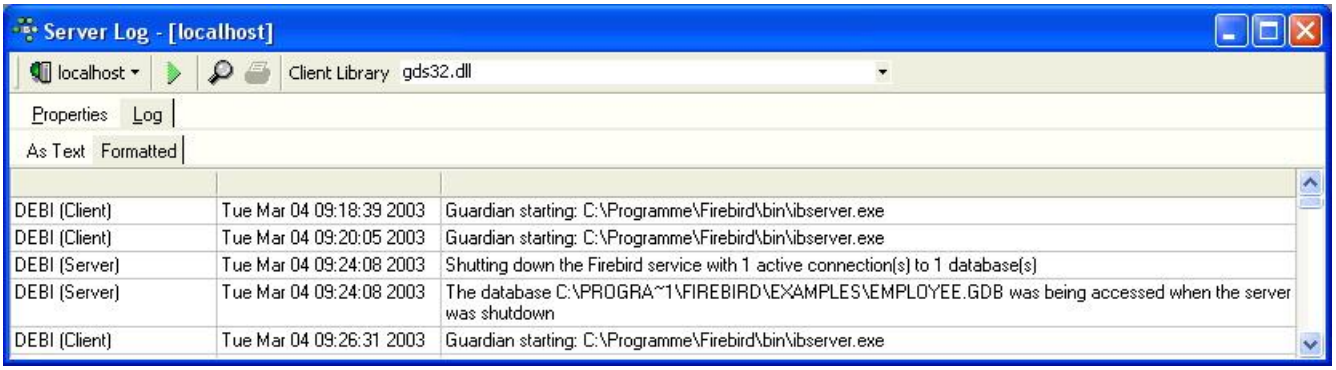


It includes server version information, configuration information and database information, particularly interesting, when working with remote and/or multiple connections.

The log can be started using the *Retrieve* (green arrow) icon. The log page displays information either as text:



or in a grid form:



The log may even be printed - the print preview can be opened using the magnifying glass icon.

See also:
Server Properties/Log toolbar

Server Activation Certificates

This option is purely for Borland InterBase v 6.5. It allows new InterBase users to be registered or existing users to be removed directly in IBExpert, using the Borland InterBase certificate keys and IDs, without having to use IBConsole.

Database Validation

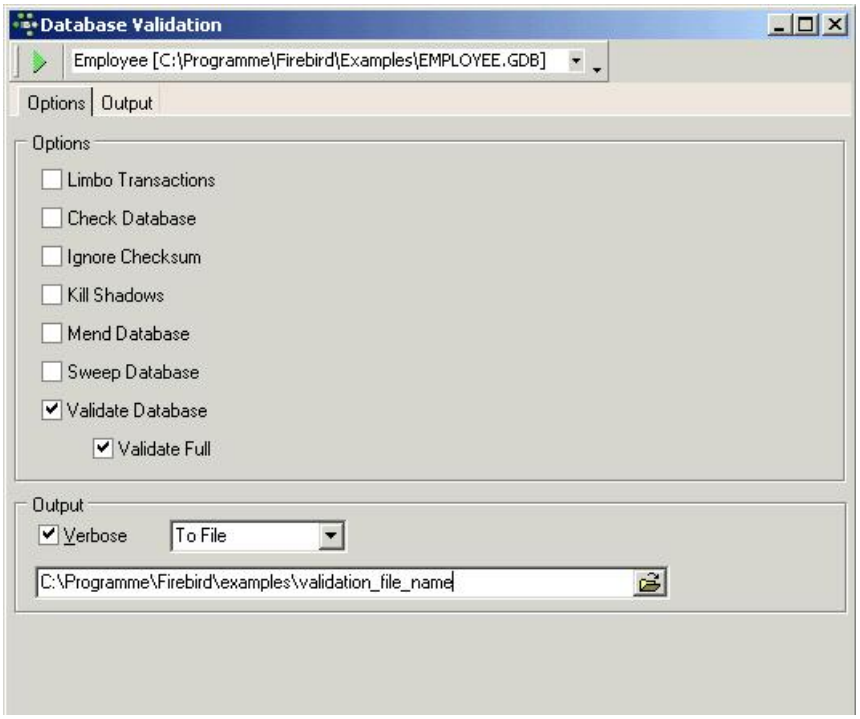
Database validation involves checking the database file to ensure that the various data structures retain their integrity and internal consistency. The validation process checks for three different types of problems:

- **Corrupt data structures:** for example, if a database row spans more than one page and the pointer that links the first page to the second is damaged or missing, there is a corrupt data structure. InterBase/Firebird is able to correct this situation, but the damaged row might be lost.
- **Misallocated data pages:** for example, a page can be used for transaction inventory, header information, data, blob pointers, or indices. If a page has been flagged as one type, but actually stores data of a another type, InterBase/Firebird detects the problem. However InterBase/Firebird cannot recover from this type of problem, so it will probably be necessary to restore from a backup.
- **Orphaned data pages**, which are automatically returned to the free space pool. By default, InterBase/Firebird does not completely fill data pages with records, to allow space for new records to be quickly inserted. As records are added and deleted, some pages are likely to end up with no active records on them. Older InterBase/Firebird versions do not automatically reallocate these pages to the free space pool.

The IBExpert Database Validation menu item offers those options also available in the InterBase/Firebird `GFIX`.

It is advisable to backup the database before validating. If possible it should also be shut down, so that the backup can be restored if necessary without any loss of transactions which may have been performed since the backup.

The Database Validation menu item can be found in the IBExpert Services menu. It enables the database to be validated and verifies the integrity of data structures.



First select the registered database to be validated. The following options are none other than the `GFIX` parameters and may be specified as wished:

- **Limbo Transactions:** If this option is checked, the database is checked for transactions in limbo, i.e. transactions, that can't be defined as executed or aborted. Please refer to transactions in limbo for further information.
- **Check Database:** This option validates the database, but doesn't repair it.
- **Ignore Checksums:** This option ignores all checksum errors. A checksum is a page-by-page analysis of data to verify its integrity. A bad checksum means that a database page has been randomly overwritten (for example, due to a system crash).
- **Kill Shadows:** This option kills all unavailable shadow files.
- **Mend Database:** This prepares a corrupt database for backup and repairs any database corruption if possible.
- **Sweep Database:** This option can be checked to perform a database sweep (see database sweep for more information about sweeps).
- **Validate Database:** (default value). This option validates the database structure.
- **Validate Full:** This validates record fragments. Note: This feature is not available in InterBase versions older than the version 6.
- **Output:** Check Verbose to receive an extended report about the current database validation process. Select whether this report should be displayed on screen or saved to file (not forgetting of course to specify drive, path and file name).

Then start the database validation using the green arrow icon or [F9].

Output

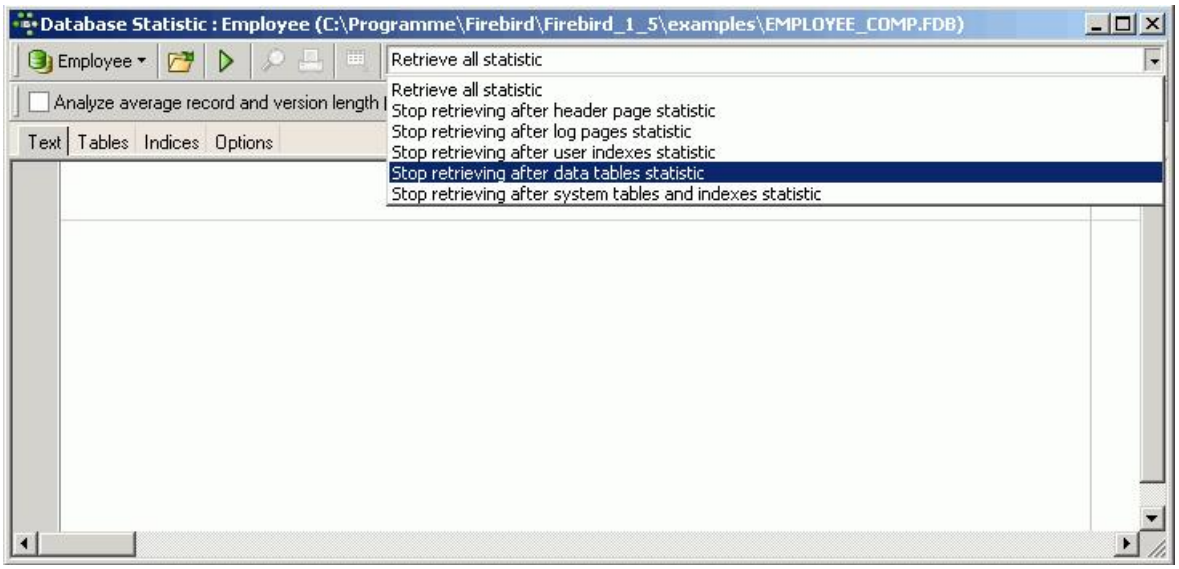
If no corruption is detected, a message is displayed informing that no database validation errors were detected. If corruption is detected that can be repaired, a report is displayed showing the number and types of errors found. Note that sometimes, irreparable database corruption is found, such as damage to the database header or space allocation tables.

Please refer to Database Corruption for further information concerning the recovery of corrupt databases.

Database Statistics

Database Statistics are an invaluable insight to what is actually happening on the server. Firebird statistics should be evaluated regularly and kept, because when things do go wrong, it's immensely helpful to be able to see what they looked like when things were running smoothly. Poor or degrading database performance is practically always to do with poor programming and/or poor transaction handling. The IBExpert Database Statistics retrieves and displays important database statistical information, which can be exported to numerous file formats or printed. This menu item can be found in the IBExpert Services menu.

First select a registered database from the pull-down list on the toolbar, or alternatively open an existing statistics file to view and analyze.



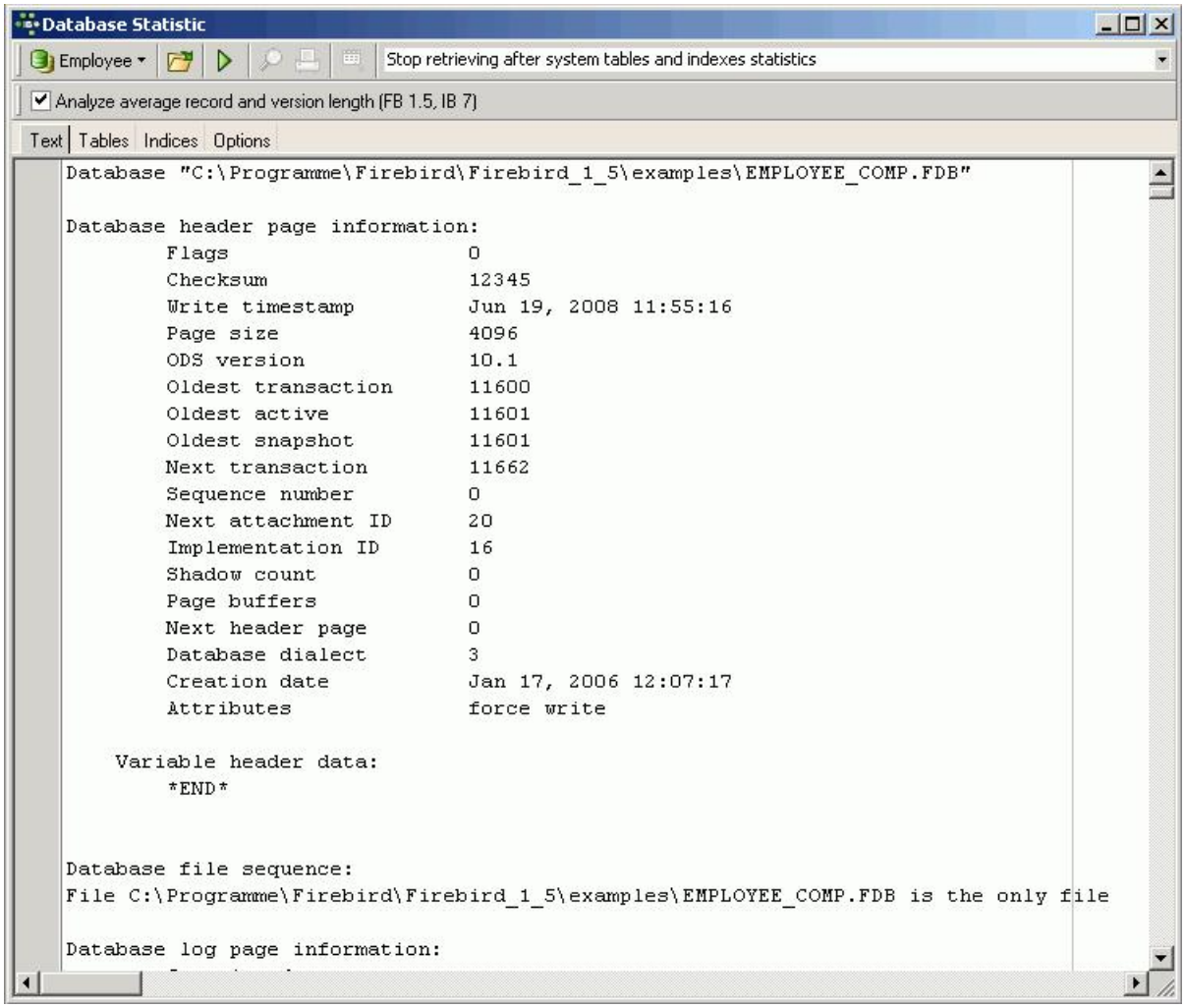
If wished, alter the default value *Retrieve all Statistics*, by selecting one of the following options:

- Stop retrieving after header page statistics
- Stop retrieving after log page statistics
- Stop retrieving after user indexes statistics
- Stop retrieving after data tables statistics
- Stop retrieving after system tables and indexes statistics

Since IBExpert version 2004.8.5 there is the added check option to analyze average record and version length (Firebird 1.5, InterBase 7) which can be found below the toolbar.

Then simply click the *Retrieve Statistics* icon (green arrow) or press [F9] to start the retrieval process.

The database's statistical summary is displayed both as text:

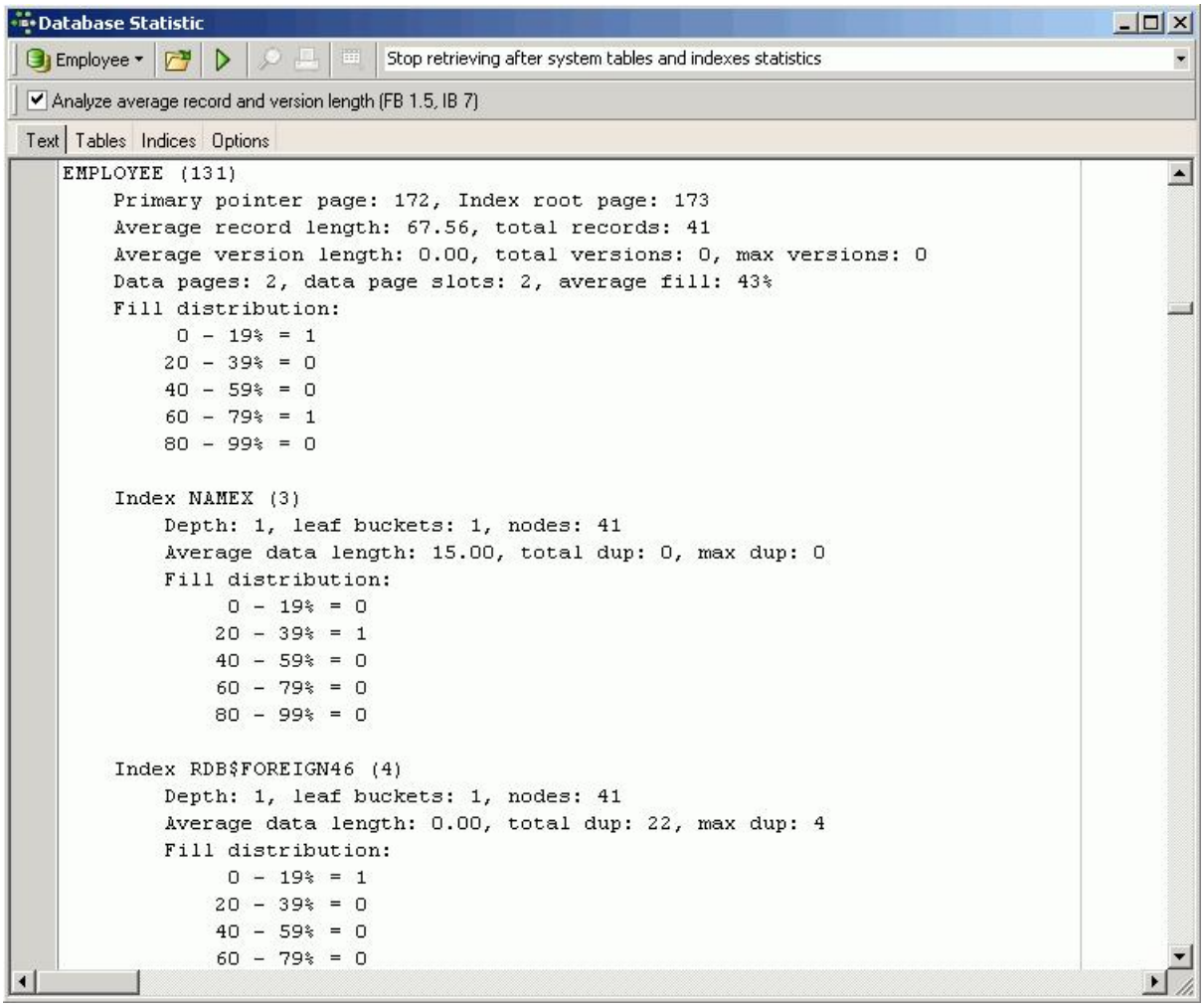


as well as in grid form (illustrated in the *Tables* page section below).

Text page

The text summary provides certain additional information (illustration above) as well as a statistical summary broken down by table (illustration below), containing the information also displayed in the grid summary.

The Database Statistics display the following information for all tables in the database, both as a log script and in tabular form: table name, location, pages, size (bytes), slots, fill (%), DP usage (%) and fill distribution (an optimal page fill is around 80%). For each table the indices statistics include: depth, leaf buckets, nodes, average data length and fill distribution.



Primary Pointer page: In the illustration above the primary pointer page (PTR) for the `EMPLOYEE` table is number 172. It begins at the byte that equals the page number 172 multiplied by the page size. This is a sort of table of contents for the `EMPLOYEE` table, it points to the data pages which contain the table's data.

Index root page: The same information is displayed for the index root pages (IRT) for the indices in this table and where they can be found.

Average record length: This displays how long the data record versions are on average. When a dBase table is created, for example, with 2 fields, each `CHAR(100)`, the average data set length would always be 200. Firebird however does not store adjacent empty spaces. For example with a `CHAR(100)` field containing a string length of 65 followed by 35 empty spaces, Firebird stores the string of 65 plus 1 empty space multiplied by 35. This is why, when data is imported into Firebird from another database, the data is sometimes smaller following the import than it was before.

Total records: How many data sets are there in the individual tables.

Average version length: The length of the record versions on average. When updates are made, you can see here how many bytes on average have altered, compared to the original data set.

Total versions: How many record versions exist for this table.

Max versions: The maximum number of versions for a record.

Data pages: How many data pages are used.

Average fill: The amount of data page fill in %

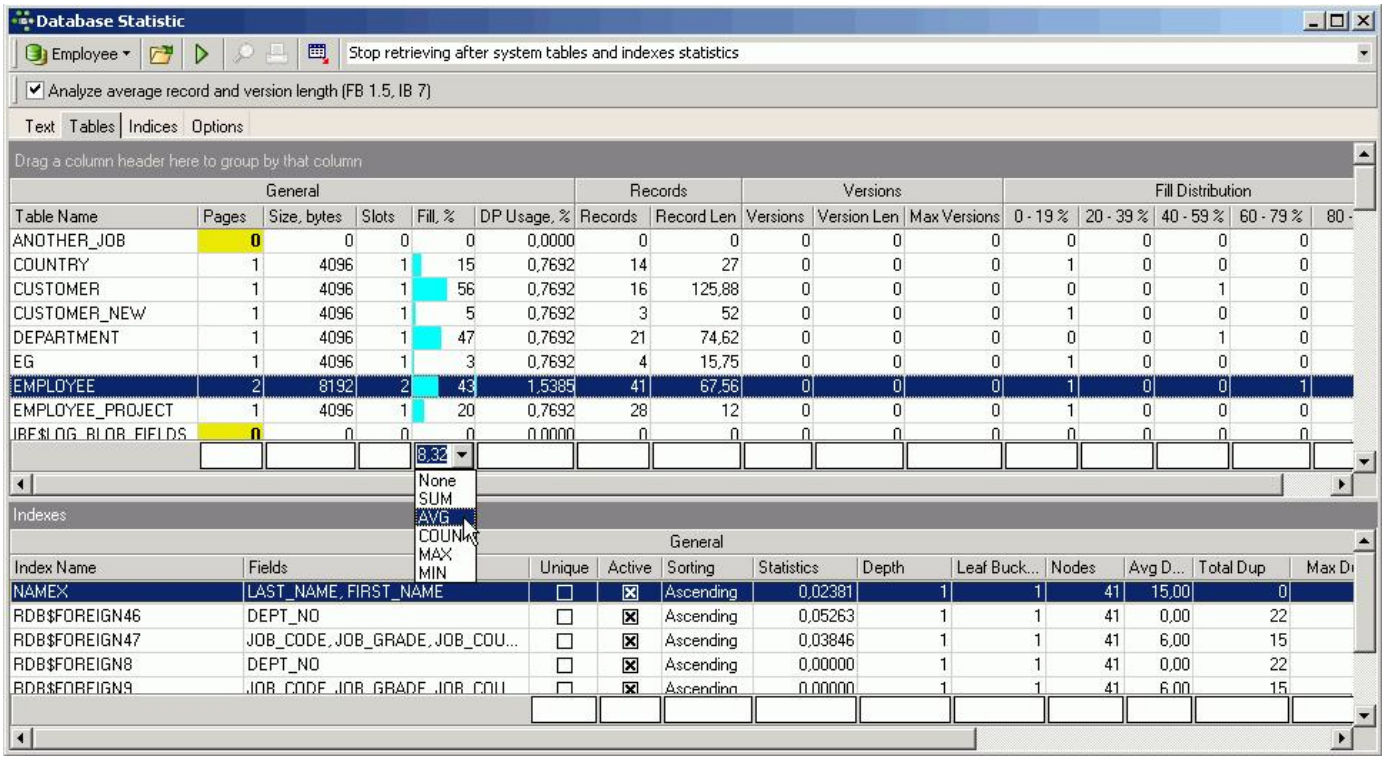
Fill distribution: The average fill is calculated how much data is already contained on the data pages. The Firebird server normally fills pages up to a maximum of 80%. The free room is needed for back version storage; if an update to one of the data sets stored on this page is made, the new data set can be stored on the same page as the original version. This saves the number of pages which need to be loaded, should it be necessary to return to the original data set.

The fill distribution also indicates whether the fill for an individual table is an anomaly or if similar problems occur on all tables.

There are certain situations when you might wish for a 100% fill (e.g. when wishing to store an address database on a CD). This can be done with the *Use all space* option when performing a database restore.

Tables page

The tables are listed alphabetically by name but, as always in IBExpert, they can be moved or sorted by any of the listed criteria by clicking on the corresponding column header. Column headers can be dragged to the top of the *Tables* page to display data sorted by that column.



It is possible to calculate certain aggregate functions on the individual columns (see the *Fill %* column in the illustration above).

The table grid gives some nice feedback about fill and database usage on your tables, e.g. you can quickly spot a table with thousands of pages at 50% fill - wasting half the space and using up cache buffers twice as fast as you could be if the pages were full. This indicates tables with a lot of inserts and deletes, that space will be reused. It could however also be due to bad page size, e.g. with a page size of 4K or 8K and tables that have perhaps had fields added over a period of time. If the data sets are so large that only one or two records fit onto the page, this will leave a large amount of space.

Below the table grid, an index grid displays the statistics for all indices for a selected table. The following information is displayed for indices: index name, fields, unique, active, sorting order, statistics, depth, leaf buckets, nodes, average data length, total dup and fill distribution. Further information can be found under *Indices page*.

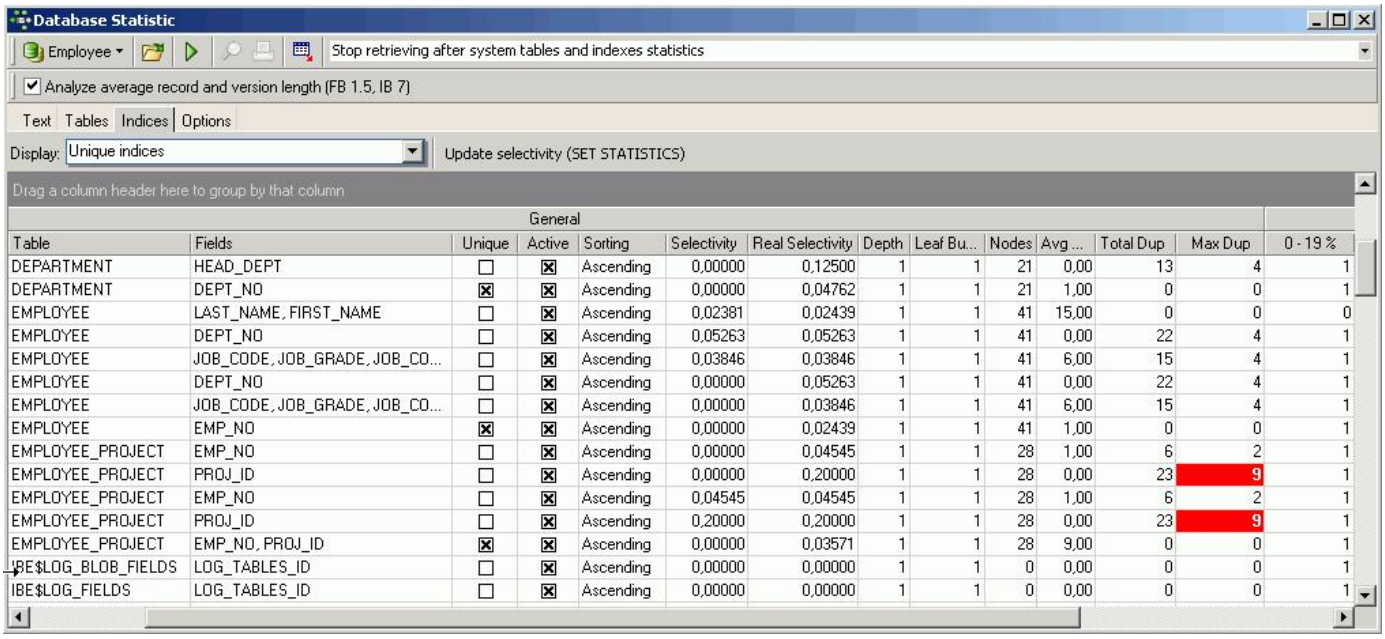
This information can be exported (see Export Data) to save the information to file, or printed out.

Indices page

In addition to the summary information displayed on the *Tables* page, the *Indices* page allows you to analyze all your database indices in depth.

Using the drop-down list, you can specify which index types you wish to view:

- All indices
- Bad indices
- Useless indices
- Too deep indices
- Active indices
- Inactive indices
- Unique indices



				General									
Table	Fields	Unique	Active	Sorting	Selectivity	Real Selectivity	Depth	Leaf Bu...	Nodes	Avg ...	Total Dup	Max Dup	0 - 19 %
DEPARTMENT	HEAD_DEPT	☐	☒	Ascending	0,00000	0,12500	1	1	21	0,00	13	4	1
DEPARTMENT	DEPT_NO	☒	☒	Ascending	0,00000	0,04762	1	1	21	1,00	0	0	1
EMPLOYEE	LAST_NAME, FIRST_NAME	☐	☒	Ascending	0,02381	0,02439	1	1	41	15,00	0	0	0
EMPLOYEE	DEPT_NO	☐	☒	Ascending	0,05263	0,05263	1	1	41	0,00	22	4	1
EMPLOYEE	JOB_CODE, JOB_GRADE, JOB_CO...	☐	☒	Ascending	0,03846	0,03846	1	1	41	6,00	15	4	1
EMPLOYEE	DEPT_NO	☐	☒	Ascending	0,00000	0,05263	1	1	41	0,00	22	4	1
EMPLOYEE	JOB_CODE, JOB_GRADE, JOB_CO...	☐	☒	Ascending	0,00000	0,03846	1	1	41	6,00	15	4	1
EMPLOYEE	EMP_NO	☒	☒	Ascending	0,00000	0,02439	1	1	41	1,00	0	0	1
EMPLOYEE_PROJECT	EMP_NO	☐	☒	Ascending	0,00000	0,04545	1	1	28	1,00	6	2	1
EMPLOYEE_PROJECT	PROJ_ID	☐	☒	Ascending	0,00000	0,20000	1	1	28	0,00	23	9	1
EMPLOYEE_PROJECT	EMP_NO	☐	☒	Ascending	0,04545	0,04545	1	1	28	1,00	6	2	1
EMPLOYEE_PROJECT	PROJ_ID	☐	☒	Ascending	0,20000	0,20000	1	1	28	0,00	23	9	1
EMPLOYEE_PROJECT	EMP_NO, PROJ_ID	☒	☒	Ascending	0,00000	0,03571	1	1	28	9,00	0	0	1
IBE$LOG_BLOB_FIELDS	LOG_TABLES_ID	☐	☒	Ascending	0,00000	0,00000	1	1	0	0,00	0	0	1
IBE$LOG_FIELDS	LOG_TABLES_ID	☐	☒	Ascending	0,00000	0,00000	1	1	0	0,00	0	0	1

The indices are listed by table and field but, as always in IBExpert, they can be moved or sorted by any of the listed criteria by clicking on the corresponding column header. Column headers can be dragged to the top of the *Indices* page to display data sorted by that column. You can immediately discern the index type (unique, active, ascending or descending).

The *Selectivity* column displays the actual selectivity which is taken into consideration by the Firebird server, when working out how best to process a query. The *Real Selectivity* column displays the level of selectivity that could be attained if the index was recomputed. Should you discover discrepancies in these two columns, click the *Update selectivity (SET STATISTICS)* button to recompute the selectivity. These discrepancies arise because the selectivity is only computed at the time of creation, or when the IBExpert menu item Recompute Selectivity or Recompute All is used (found directly in the *Statistic* dialog, in the IBExpert Database menu, or in the right-click DB Explorer menu). Alternatively the

```
SET STATISTIC INDEX {INDEX_NAME}
```

command can be used in the SQL Editor to recompute individual indices.

This is automatically performed during a database backup and restore, as it is not the index, but its definition that is saved, and so the index is therefore reconstructed when the database is restored.

The next column displays the index depths can be viewed. An index depth of 2, for example, indicates that InterBase/Firebird needs to perform two steps to obtain a result. Normally the value should not be higher than three. Should this be the case, a database backup and restore should help.

Leaf buckets display the number of registration leaves, where InterBase/Firebird can access immediately. Further statistics include nodes, duplicates (total and maximum) and fill distribution.

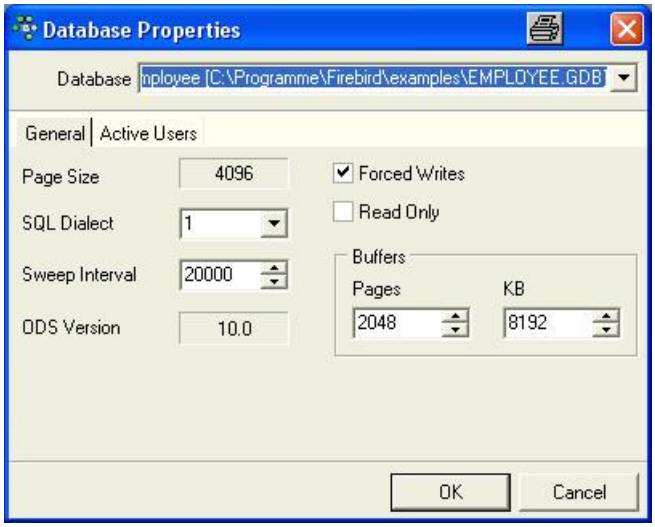
Options page

IBExpert version 2007.09.25 added the possibility to automatically analyse tables/indices statistics and the highlight possible problem tables/indices. This feature based on the IBEBlock functionality and is therefore is fully customizable.

See also:
Multi-generational architecture (MGA) and record versioning
Index
Firebird for the database expert: Episode 2 - Page types
Firebird for the database expert: Episode 4 - OAT, OIT and Sweep
Transaction
GFIX

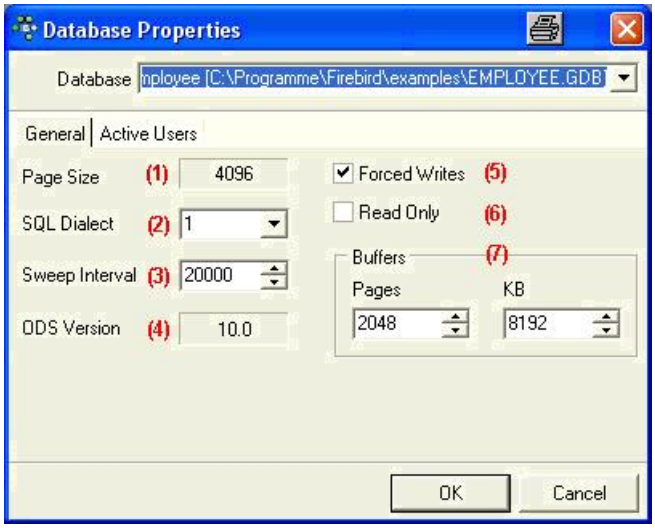
Database Properties

The Database Properties Editor can be started from the IBExpert Services menu. It can be used to specify certain properties and view others appertaining to the database specified in the *Database* pull-down list (in the upper part of the editor).



There are two tabs labeling the General page and the Active Users page.

General page



The *General* page displays the following information for the selected database:

(1) Page Size: displays the current specified page size. The page size can only be altered by performing a database backup followed by a restore (IBExpert menu: Services / Restore Database) and redefining the database page size.

(2) SQL Dialect: shows which SQL dialect was specified at the time of database registration. This may be altered here, if wished (although watch out for possible dialect incongruencies, for example, the different `date` and `time` types).

(3) Sweep Interval: This displays the number of transactions which may be made in the database before an automatic garbage collection is executed by InterBase/Firebird. If this number is specified at zero (0) it is not performed automatically at all. It could then be carried out, for example, at night as a sweep or backup using `GFIX` and the `at` Windows command or the Linux `chron` command. Please refer to database sweep for further information.

(4) ODS Version: The ODS (= On-Disk Structure) version shows with which database version the database was created, e.g. InterBase 5 = ODS version 9, InterBase 6 = ODS version 10.0, InterBase 6.5 = ODS version 10.1, InterBase 7 = ODS version 11. Firebird versions start at ODS version 10.0.

(5) Forced Writes: This enables the forced writing onto disk mode. when committing. Please refer to forced writes for further information.

(6) Read Only: A database can be set to *Read Only* when, for example, saving the database onto a CD, or in the case of a reference or archive database. The *Read Only* property is forced in the TIP page, by preventing all insert, alter and delete commands.

(7) Buffers: Here it is possible to specify how much cache the database server should reserve. A good number of buffer pages is 10,000 (based on a 4K page size to allow 40MB cache). The amount of buffers/cache reserved can be viewed in IBExpert here (default = 2,048). If this is increased the database can load considerably more pages. Please refer to buffers for details.

Buffers

The buffers/cache can be set using the IBExpert menu item Database Properties, found in the Services menu, or using the command-line utility `GFIX`. The amount of buffers/cache reserved can be viewed in IBExpert under Services / Database Properties. The IBExpert Performance Analysis also displays the number of data pages that are being held as cache on the server (from InterBase 6 onwards the standard is 2,048). Please refer to Performance Analysis / Additional for further information. This can be altered for the current database if wished.

If this is increased the database can load considerably more pages. For instance, it is much more efficient to load 10,000 pages, than loading 2,000 and then exchanging for new pages once the 2,000 have been loaded. The only limit to amount of cache is the physical size of the RAM (e.g. 10,000 x 4K page size). The total KB is calculated according to the current database page size. For an alteration to become effective, it is therefore necessary for all users to disconnect from the database and then reconnect.

Buffers are only reserved if they are really necessary.

Database sweep / sweep interval

When a database is swept, all old invalid data is removed from the data pages, thus reducing the total size of the database and making room for new data sets.

A database sweep performs a garbage collection in the database, and is performed automatically during a database backup or when a `SELECT` query is made on a table (and not by `INSERT`, `ALTER` or `DELETE`). Furthermore database sweeps are, as standard, executed automatically after every 20,000 operations. With very consistent databases however a database sweep can be started unnecessarily and thus cost unnecessary performance losses during normal user processing. The default database sweep interval value of 20,000 (operations) can be overwritten using the IBExpert Services menu item Database Properties.

Under *Sweep Interval* the number of operations can be specified before a database sweep should be automatically performed. A database sweep or backup can be performed during 24 hour operation online without any problems (i.e. the server does not need to be shut down). This however does slow performance during the sweep which may not be desired.

If the sweep interval is specified at zero (0) it is not performed automatically at all. It could then be performed explicitly, for example, at night as a sweep or backup using `GFIX` and the `at` Windows command or the Linux `chron` command.

New to Firebird 2.0: Superserver garbage collection changes

See also:
Database repair and sweeping using `GFIX`
Firebird for the database expert: Episode 4 - OAT, OIT and Sweep

Forced writes

This enables the forced writing mode on disk. If the forced writes option is selected all data is saved immediately to disk, i.e. every time a commit is made everything is written to the hard drive, and then to the TIP (=Transactions Inventory Page).

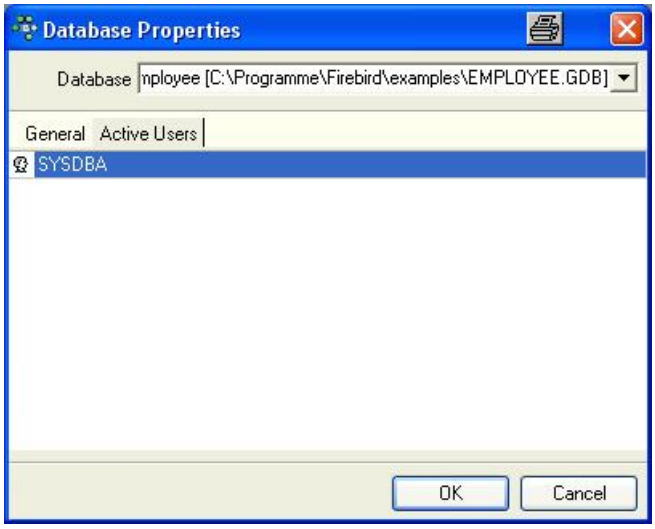
Without forced writes the process is minimally quicker, but when working on a Windows platform, Windows decides what should be saved to file, where and when, and the data pages are saved to file last i.e. the TIP changes are written first, and then the data sets - which could possibly lead to inconsistencies, particularly if it crashes during the process, as the TIP thinks alls data sets have been written to file when they are in fact incomplete. The Windows cache simply starts at the beginning and works through to the end.

The Firebird Forced Writes mechanismus writes the data where it needs it, for example, if it needs to open a new data page to write data into, it makes the necessary note in the contents that this page contains data for the table concerned, and also makes a note in the primary pointer pages for the table itself. Finally, when everything has successfully committed an entry is made in the TIP of what has been done and that it has been committed.

Using forced writes is therefore always recommended, and should never be deactivated unless really necessary.

See also:
Disabling forced writes
Forced writes - cuts both ways

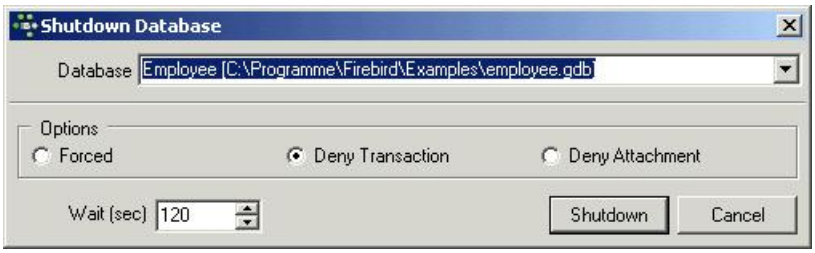
Active Users page



This page displays those users logged in to the current database with an open attachment. If an application has several attachments, or a single user is connected more than once, this is also visible here. This is important should the database need to be shut down at short notice.

Database Shutdown

There are a few occasions when a database needs to be shut down. For example, when a new foreign key needs to be inserted the database should be shut down in order to avoid the annoying message "Object in use". A registered database can be shut down simply and quickly using the IBExpert Services menu item Database Shutdown.



Select the registered database which is to be shut down. Then select one of the following options, to specify how active transactions should be dealt with:

- **Forced:** In this mode all transactions, that are still active at the stated time, are aborted regardless of their type or importance, and all users are forcefully disconnected. As InterBase/Firebird transactions function stably and securely, there are very few areas of application where this forced mode should not be used.
- **Deny new transactions:** In this mode all transactions must be executed by the stated time. Any new transactions that are started are blocked. If there are any transactions that are still active by the stated time, the database shutdown is not executed.
- **Deny new attachments:** With this option all active user attachments must finish their work by the stated time. If some attachments are still active by the stated time, the database shutdown is not executed.
- **Wait:** The period of time (in seconds) until the shutdown is executed can be specified here.

Then simply click *Shutdown* to shutdown the database. To bring the database back online, choose the IBExpert Services menu item Database Online.

See also:
Database shutdown using `GFIX`

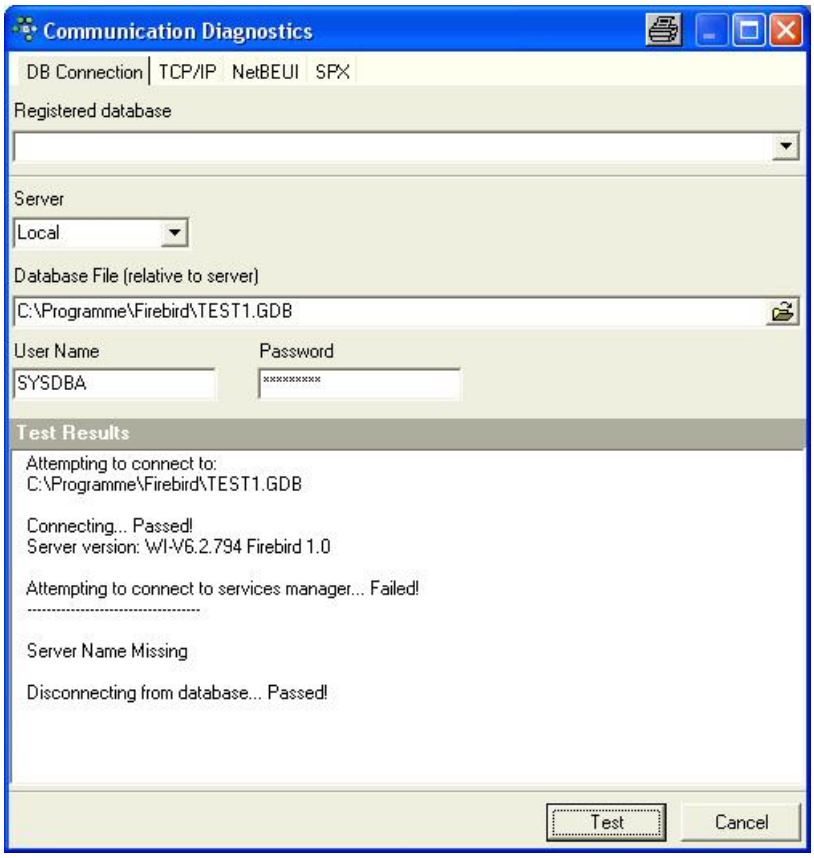
Database Online

The IBExpert Services menu item Database Online is used to bring a database back online again after it has been shut down (please refer to Database Shutdown for further information).

Simply select a registered database and bring the database online.

Communication Diagnostics

The Communication Diagnostics dialog can be started from the IBExpert Services menu. It also appears automatically when registering a database and the *Test Connect* button is pressed. IBExpert's Communication Diagnostics delivers a detailed protocol of the test connect to a registered InterBase/Firebird server and the results:



This is particularly useful when attempting to connect to a remote database server, as detailed status information concerning the various steps taken to make the connection is displayed, indicating problem areas if the connection is not achieved. If using an alias path for a remote connection, please refer to the article Remote database connect using an alias.

The following protocols are supported:

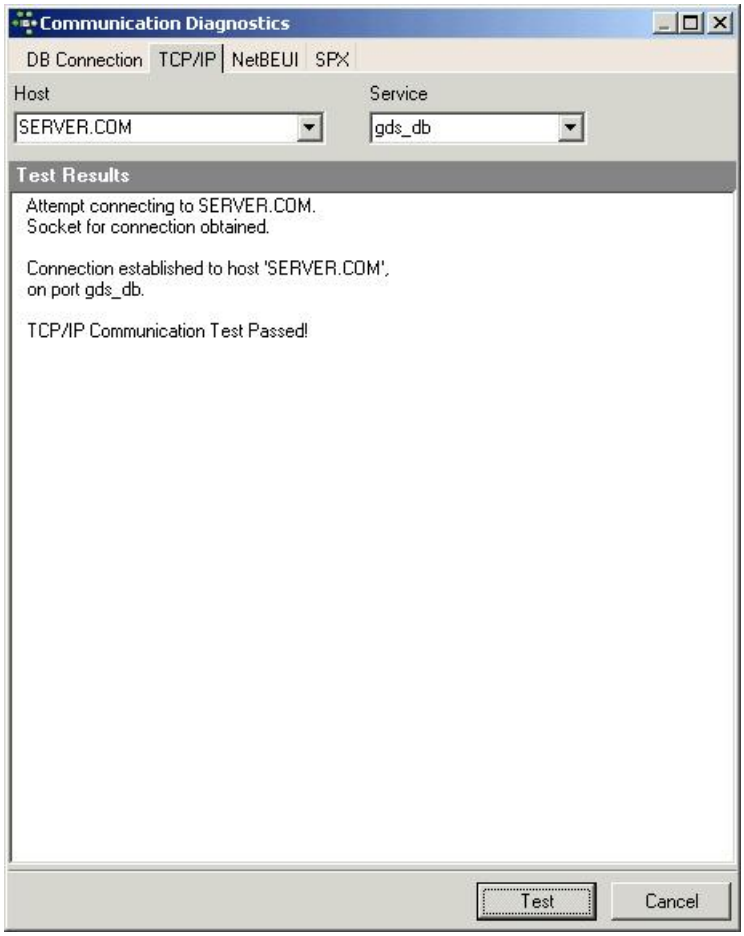
- TCP/IP (worldwide standard)
- SPX - which used to be used by Novell; now even Novell supports TCP/IP. a
- NetBEUI - which is not really a network protocol, it simply accesses the line. It is slow as it makes everything available everywhere and anyone can access the information. This is also purely a Windows protocol.

Should problems occur, switch to the relevant protocol page and test again.

The TCP/IP protocol offers the following services:

- **21 and FTP:** Each port receives a name. With Firebird this is actually optional, with InterBase: `Win\System32\ drivers\etc\services -> ftp (= the name for-) 21/tcp`.
- **3050:** This is the standard port for InterBase and Firebird. However this is sometimes altered for obvious reasons of security, or when other databases are already using this port. If a different port is to be used for the InterBase/Firebird connection, the port number needs to be included as part of the server name. For example, if port number `3055` is to be used, the server name is `SERVER/3055`.
- **gds_db:** For InterBase: `name = gds_db = 3050 / tcp` (a different port to the standard `3050` can be specified if wished). If this entry is nonexistent Firebird does not care; InterBase however does! The name `gds_db` has to be present.
- **Ping:** can be used if the connection was unsuccessful and the reason is not known. This DOS command checks which input is correct, and works regardless of whether `InterBase.exe` or `Firebird.exe` is installed. The results show whether a database has been found, and at which address. This should, as a rule, always work unless of course the server uses a Firewall which does not allow a Ping to be answered. In this case, use the service FTP (as a rule the same as the 21 service).

Note: in DOS the `TRACERT` command lists the protocol route. TCP/IP intelligently takes another direction if one or part of the lines on the quickest route is blocked or down.



Problems may occasionally arise when attempting to connect to a remote server, due to Firewall issues. These can usually be solved by simply changing the port assignment in `firebird.conf` from `3050` to `3051`.

See also:
Comdiag
Register Database
Remote database connect using an alias

HK-Software Services Control Center

The HK-Software Services Control Center includes the following services, each documented individually:

IBExpertBackupRestore

IBExpertInstanceManager

IBExpertJobScheduler

IBExpertSQLMonitor

IBExpertTransactionMonitor

IBExpert PlugIns menu

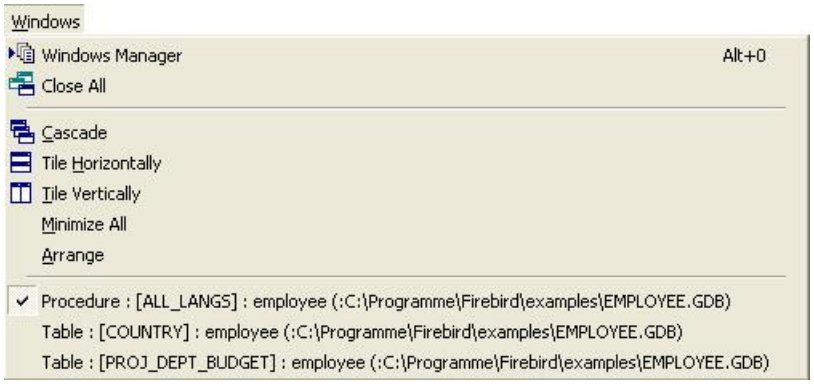
The IBExpert PlugIns menu is intended for user-specified menu items for third party components. Two Delphi PlugIn examples are delivered as part of IBExpert and can be found in the `IBExpert/PlugIn` directory. Should you have problems finding these files they can also be downloaded free of charge from the web: http://www.ibexpert.com/download/PlugIns (a direct link can be found in the IBExpert Help menu item, IBExpert Direct). You need to have Delphi, InterBase or Firebird and, of course, IBExpert installed.

Installation of the components is explained in detail in the `Readme.txt` files enclosed.

See also:
Environment Options / Additional Tools
IBExpert Help menu / Additional Help files

IBExpert Windows menu

The IBExpert Windows menu offers a number of options to visually arrange all open windows in IBExpert.



Please note that all open windows are also displayed as buttons on the Windows bar (directly above the status bar), and in the DB Explorer on the Windows page (please refer to Windows Manager for further information).

Windows Manager

The Windows Manager can be opened using the IBExpert Windows menu item Windows Manager, by using the key combination [Alt + O], or simply by clicking on the Window tab heading directly in the DB Explorer.

For more information regarding this, please refer to DB Explorer / Windows Manager.

Close All

Close All is an option to close all open windows with one simple mouse click, ideal when closing all open work for one project or database, before beginning work on a new project or database, or finally finishing work for the day (...or night!).

Cascade / Tile / Minimize / Arrange

The IBExpert Windows menu offers the following options, for arranging all open windows:

- **Cascade:** all open windows are arranged one behind the other, in a cascading format, displaying the title bar of each window.
- **Tile Horizontally:** all open windows are displayed adjacently, one below the other.
- **Tile Vertically:** all open windows are displayed adjacently, one next to the other.
- **Minimize All:** this option minimizes all open windows simply and quickly with a single mouse click.
- **Arrange:** this option arranges the windows as currently viewed, e.g. all minimized windows are arranged in a horizontal row alongside each other.

If the SDI User Interface has been specified under Environment Options / User Interface, then only the *Cascade* option is offered here.

See also:
User Interface
Windows Bar
Windows Manager

IBExpert Help menu

The IBExpert Help Menu offers a number of provisions to offer support for IBExpert.

Since IBExpert version 2004.2.26.1, there is a new context-sensitive help system. Pressing [F1] in any of the IBExpert forms now opens a new web-based Help page. It is also possible to download all Help Pages from http://www.ibexpert.info/documentation/documentation.zip and unzip this in the IBExpert main directory with subdirectories (there must be a new subdirectory called `documentation`). If a local Help document is available, it will be opened in the browser. Otherwise the browser will open the page from our web server. If you have any comments or questions please use our newsgroup (please see below).

The complete help files are also available directly online: http://ibexpert.net/ibe/pmwiki.php?n=Doc.IBExpert.

The first view displays the Help structure. If you are looking for help about a specific subject use the Search: function.

To integrate the online Help Files into IBExpert itself, follow these five steps:

1. Download the help file (http://www.ibexpert.info/documentation/documentation.zip)
2. If you have an older version of IBExpert, delete the `Help` directory.
3. Create a new directory: `Documentation` in the IBExpert main directory.
4. Extract and copy the `documentation.zip` file into the `IBExpert\Documentation` directory.
5. When you start IBExpert and press [F1] from any dialog, the DB Explorer or the SQL Assistant, it will open an html file in `C:\program files\HK-Software\IBExpert 2.0\Documentation\helpcontext` showing you the relevant help information.

Should you not be able to find a solution to your problem here, please use one of our newsgroups:

Username: **ibexpert**
Password: **ibexpert**

news://ibexpert.info/interbase.ibexpert.de German language
news://ibexpert.info/interbase.ibexpert.en English language
news://ibexpert.info/interbase.ibexpert.ru Russian language
news://ibexpert.info/interbase.ibexpert.fr French language

or send us an email to support@ibexpert.com or use our Bug Track System in the IBExpert Help Menu.

Should you have any comments or queries directly regarding the Help documentation, or wish to contribute your own articles, please contact documentation@ibexpert.com

IBExpert Customer Area

New to IBExpert version 2005.3.12.1: this menu item allows all registered users of full versions (not the Trial Version or IBExpert Personal Edition) direct access to the protected customer area, without having to search for their current registration keys.

Simply click the menu item, and IBExpert uses your registration keys to automatically access the online IBExpert Customer Area. This does nothing other than open a URL such as the following example:

```
http://1234567887654321:ibexpert@www.ibexpert.com/customer
```

where `1234567887654321` is a combination of *Key A* and *Key B* which is already stored in the registry. (There is no point testing the above link, as the keys quoted are for example only!).

Warning: Although this function works faultlessly with browsers such as Firefox, problems may be experienced with Windows Internet Explorer. In this case, it is necessary to access the protected customer area under http://www.ibexpert.com/customer in the usual way, by inputting your customer keys and password, and then download the `customer_area.reg` to the local drive and then merge in `regedit` (Windows menu Start / Execute; type `regedit`, right-click menu item *Merge* and merge the files).

Alternatively it is possible to create the following registry key manually:

1. In Windows click the bottom left menu Start.
2. Execute.
3. Type `regedit` and enter (or click *OK*).
4. `HKEY_LOCAL_MACHINE` ist the root-key. Open the folders `SOFTWARE`, `Microsoft`, `Internet Explorer`, `Main` and `FeatureControl`.
5. Here you need to add a new feature `FeatureControl`.
6. You should then add `FEATURE_HTTP_USERNAME_PASSWORD_DISABL`E and using the right-click menu in the empty right dialog area, select *New* and then *Key*, and type `IExplore.exe` in the input field.
7. On the left you will now find a new folder, `IExplore.exe`, in the *FeatureControl* list. Highlight this, use the context-sensitive right-click menu to select *New/DWORD value*.
8. Add new `DWORD` with name `IExplore.exe` and value "0" ("IExplore.exe"=dword:00000000).

What's New?

IBExpert 2008.08.08

The new IBExpert version 2008.08.08 includes the new feature IBExpert Instance Manager and many improvements and small bug fixes.

1. IBExpertInstanceManager

The IBExpertInstanceManager is a new module in HK-Software Control Center. It can be started using the IBExpert Services menu item, HK-Software Services Control Center. It allows you to install several instances of the Firebird server on one Windows machine using different ports. Additional functions allow monitoring and other useful options.

Step by step instructions:

1. Be sure that there is already a Firebird Instance installed on the machine using the default Firebird installer.
2. Install the new IBExpert version.
3. Start the Services-HK Software Services Control Center.
4. Select the IBExpertInstanceManager service.
5. Right click on it and select *Add task*.
6. For the newly added task select *BaseService* from the list of Firebird instances installed on your PC.
7. Set the port number for the Firebird instance you are going to create. All other instance configuration settings will be generated automatically.
8. Setup mail notification if needed.
9. Setup validation parameters if needed. Validation is just a test connection to `security.fdb` of the new instance, using the instance's port number.
10. Set the task's *Active* parameter to *True*.
11. To rename the task, click on the task name with the [Ctrl] key pressed down.
12. Run the service. When properly configured the running task should show runtime info on the first run.

The full documentation can be found here.

That's it! Using multiple instances of the Firebird Server has different advantages, for example using different `SYSDBA` passwords, using multiple CPUs more effectively, using old and new Firebird version on one machine etc.

To distribute the IBExpertInstanceManager with your application, you need a Junior VAR License or a VAR License.

2. IBEBlock *Results* form:

- Added the possibility to sort data by clicking on a grid column caption.
- Added the option to export data.

3. Table Data Comparer:

- It is now possible to generate `UPDATE OR INSERT` instead of `UPDATE/INSERT` for Firebird 2.1 databases.

4. IBEBlock:

- `ibec_CompareTables` function.

Here it is now possible to compare more than one table in a single operation. Just specify the list of necessary tables, delimited with a comma or semicolon, as `MasterTable` and `SubscriberTable`.

Example:

```
ibec_CompareTables@@(DB1, DB2,'TABLE1, TABLE2, "Table3"',
                     'TABLE1, TABLE2, "Table3"',
                     'D:\Diff.sql', 'UpdateOrInsert', cbb);'
```

- Added `UpdateOrInsert` option (`UseUpdateOrInsert` is valid too).

This allows you to generate `UPDATE OR INSERT` statements instead of `UPDATE/INSERT` for Firebird 2.1 databases. See example above.

- `ibec_ds_Sort` function implemented.

Syntax:

```
function ibec_ds_Sort(Dataset : variant; SortFields : string) : variant;
```

`ibec_ds_Sort` function sorts `Dataset` according to the specified `SortFields`.

Example:

```
execute ibeblock
as
begin
  select * from rdb$relation_fields
  as dataset ds;
  try
    ibec_ds_Sort(ds, 'RDB$RELATION_NAME ASC, RDB$FIELD_POSITION ASC');
    ibec_ds_Sort(ds, 'RDB$RELATION_NAME, RDB$FIELD_POSITION');
    ibec_ds_Sort(ds, '1, 2 DESC');       finally
    ibec_ds_Close(ds);
  end;
end;
```

- `ibec_ds_Locate` function implemented.

Syntax:

```
function ibec_ds_Locate(Dataset : variant; KeyFields : string;
                        KeyValues : array of variant; Options : integer) :  boolean;
```

`ibec_ds_Locate`	searches `Dataset` for a specified record and makes that record the active record.
`KeyFields`	is a string containing a semicolon-delimited list of field names in which to search.
`KeyValues`	is a variant array containing the values to match in the key fields.

If `KeyFields` lists a single field, `KeyValues` specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as `KeyValues`, or construct a variant array on the fly using the `ibec_Array` function.

Examples:

```
ibec_ds_Locate('Company;Contact;Phone', ibec_Array('Sight Diver', 'P', '408-431-1000'), __loPartialKey);
```

or

```
Keys[0] = 'Sight Diver';
Keys[1] = 'P';
Keys[2] = '408-431-1000';
ibec_ds_Locate('Company;Contact;Phone', Keys, __loPartialKey);
```

`Options` is a set of flags that optionally specifies additional search latitude when searching on string fields. If `Options` contains the `__loCaseInsensitive` flag, then `ibec_ds_Locate` ignores case when matching fields. If `Options` contains the `__loPartialKey` flag, then `ibec_ds_Locate` allows partial-string matching on strings in `KeyValues`. If `Options` is `0` or `NULL` or if the `KeyFields` property does not include any string fields, `Options` is ignored.

This function returns `True` if a record is found that matches the specified criteria and the cursor repositioned to that record. Otherwise it returns `False`.

Example:

```
execute ibeblock
returns (FieldName varchar(100))
as
begin
  select * from rdb$relation_fields
  as dataset ds;
  try
    ibec_ds_Sort(ds, 'RDB$RELATION_NAME, RDB$FIELD_POSITION');
    res = ibec_ds_Locate(ds, 'RDB$RELATION_NAME', 'RDB$FIELDS', __loPartialKey);
    while (res) do
    begin
      FieldName = ibec_ds_GetField(ds, 'RDB$FIELD_NAME');
      FieldName = ibec_Trim(FieldName);
      suspend;
      ibec_ds_Next(ds);
      res = not ibec_ds_EOF(ds);
      if (res) then
      begin
        RelName = ibec_Trim(ibec_ds_GetField(ds, 'RDB$RELATION_NAME'));
        res = RelName = 'RDB$FIELDS';
      end;
    end;
  finally
    ibec_ds_Close(ds);
  end;
end
```

- `ibec_ExecSQLScript` function implemented.

Syntax:

```
function ibec_ExecSQLScript(Connection : variant; SQLScript : string; Options : string; ProgressBlock : variant) : variant;
```

`ibec_ExecSQLScript` executes an SQL script from a variable or a file.

`Connection` is an active connection created with the `ibec_CreateConnection` function which will be used while executing a script. If `Connection` is not specified (`NULL`) the script must contain the `CREATE DATABASE` or the `CONNECT` statement, otherwise an exception will be raised.

`SQLScript`	script text or name of script file.
`Options`	additional options. There are two additional options currently available: `ServerVersion` and `StopOnError`.
`ProgressBlock`	an IBEBlock which will be executed for every progress message generated during script execution.

`ibec_ExecSQLScript` returns `NULL` if there were no errors while executing a script. Otherwise it returns an error(s) message.

Example:

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (BlockData variant)
         as
         begin
           sMessage = BlockData;
           if (sMessage is not null) then
             ibec_Progress('SQL Script: ' + sMessage);
         end';

  db = ibec_CreateConnection(__ctFirebird, ...);
  try
    Scr = 'INSERT INTO MYTABLE (ID, DATA) VALUES (1, 'Bla-bla'); ' + 'INSERT INTO MYTABLE (ID, DATA) VALUES
   (2, 'Bla-bla'); '  + 'COMMIT;';
    ibec_ExecSQLScript(db, Scr, 'ServerVersion=FB21; StopOnError=FALSE', cbb); ...
    ibec_ExecSQLScript(db, 'D:\Scripts\CheckData.sql', 'ServerVersion=FB21', null); finally
    ibec_CloseConnection(db);
  end
end
```

- `ibec_GetViewRecreateScript` function implemented.

Syntax:

```
function ibec_GetViewRecreateScript(Connection : variant; ViewName : string;
Options : string; ProgressBlock : variant) : string;
```

`ibec_GetViewRecreateScript` creates a *Recreate* script for a specified view(s) and returns it as a result.

`Connection`	is an active connection created with the `ibec_CreateConnection function`.
`ViewName`	list of names of view(s), delimited with semicolon or comma, for which a *Recreate* script will be created.
`Options`	list of options delimited with semicolon; possible options are:
`GenerateCreate`	determines whether a `CREATE DATABASE` statement should be included at the beginning of the generated script.
`GenerateConnect`	determines whether a `CONNECT` statement should be included at the beginning of the generated script.
`IncludePassword`	determines whether the password should be included into the `CREATE DATABASE` or the `CONNECT` statement in the resulting SQL script.
`SupressComments`	use to supress comments in the resulting script.
`ExtractDescriptions`	determines whether database objects' descriptions should be included in the generated script. By default this option is enabled.
`DescriptionsAsUpdate`	determines whether the raw `UPDATE` statement should be used for object descriptions instead of the IBExpert specific `DESCRIBE` statement.
`UseComment`	generates the `COMMENT ON` statement for object descriptions (Firebird 2.x).
`DontUseSetTerm`	don't use `SET TERM` statements, all statements will be separated by semicolon only.
`UseCreateOrAlter`	generates `CREATE OR ALTER` instead of `CREATE/ALTER` where possible.
`ProgressBlock`	an IBEBlock which will be executed for every progress message generated during script execution. May be `NULL` or empty.

Example:

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (MsgData variant)
         as
         begin
           ibec_Progress(MsgData);
         end';
  ...
  RecreateScript = ibec_GetViewRecreateScript(mydb, 'VIEW_A; VIEW_B; VIEW_C',
      'GenerateConnect; IncludePassword; UseCreateOrAlter', cbb);
  Res = ibec_ExecSQLScript(null, RecreateScript, 'ServerVersion=FB21', cbb);
end
```

5. DB Explorer context menu, *Apply Block*:

- Added the possibility to recreate selected views based on IBEBlock and the `ibec_GetViewRecreateScript` function.

6. New installer

7. A lot of minor bug fixes and small improvements...

8. Forum for Firebird and IBExpert news: http://www.firebirdexperts.com

---

IBExpert 2008.05.03

The newest IBExpert version has a lot of improvements and bug fixes. The most important are:

- improved support for Firebird 2.1
- new IBEBlock functions for
  - creating reports
  - POP3 E-Mail access
  - SMTP support
  - and much more.

The command-line versions `ibescript.exe` and the DLL version `ibescript.dll` have been improved. The new customer version is available for download here: http://www.ibexpert.com/customer.

If you have already downloaded version 2008.05.03 and encounter a problem closing the Script Executive, please download the new bug-fixed version.

1. IBExpert websites completely redesigned:

We have changed almost all the IBExpert websites over the last weeks:

- The new IBExpert website can be found here: http://www.ibexpert.net/ibe.
- The new online shop can be found here: http://www.ibexpert.net/shop.
- The new documentation can be found here: http://ibexpert.net/ibe/index.php?n=Doc.Doc.

2. The Firebird forum for beginners and professionals: http://www.firebirdexperts.com

The forum is focused on Firebird-specific topics for developers. The main topics are Delphi, .NET, Java and PHP. If requested, we can also add new areas or language-specific boards. The forum runs on Windows 2003 Server, Apache web server, the current PHP version, phpBB 3.01, and Firebird 2.1. The installation is easier than most people think, especially since there is an integrated Apache and PHP version with Firebird support in the current IBExpert customer version.

3. Scripting language IBEBlock:

Reports are now available for batch creation. Some new IBEBlock commands are now available for executing reports created with IBExpert's Report Manager in command-line mode, for example with batch files. The monthly sales report, invoices or other reports can be designed in the Report Manager and executed with simple SQL statements. The result can be saved in the database as a pdf or other formats and sent by E-Mail. Further details can be found in our docmentation at http://ibexpert.net/ibe/index.php?n=Doc.IBEBlock.

IBExpert 2008.02.19

1. IBExpertWebForms now included in IBExpert Customer Version:

What is required for using IBExpertWebForms?

Since IBExpert version 2008.01.28 all IBExpert fully licensed versions, i.e. single, multiple, Site, Junior VAR and full VAR licenses, include our fully integrated IBExpertWebForms module.

If you have a customer version of IBExpert, you are allowed to use IBExpertWebForms on your registered computer. If you have a Site License, you can use IBExpertWebForms on any computer in your company. If you have a VAR or Junior VAR License, you are allowed to distribute IBExpertWebForms together with your applications to your customers.

With IBExpertWebForms you can create database-based web applications. Just place your VCL components in the integrated Form Designer, connect them with your tables or queries as a data source using the integrated object inspector, and create your events as stored procedures inside your Firebird or InterBase database.

The result is handled by a PHP script, which is used by the Apache web server on Windows, Linux or any other operating system which supports Apache, PHP and Firebird or InterBase.

The main advantage: you do not need any know-how regarding Java script, HTML, Ajax, PHP, etc. to create your database web application. All operations are done inside your database and you just need to learn some very simple extensions and rules based on your existing Firebird and InterBase knowledge. Start your database web development in just 10 minutes after reading this document!

http://www.ibexpert.com/download/IBExpertWebForms/IBEWebFormsFirstSteps.pdf

2. Database Explorer:

- Drag 'n' drop of objects from the Database Explorer into the Code Editor.
  Since this version it is possible to create your own sets of statements that will be composed when you drag 'n' drop object(s) from the Database Explorer into any code editor. This feature is based on IBEBlock; refer to the example below for more details.
- Context Menu / Apply IBEBlock to selected object(s).
  This feature is also based on the IBEBlock functionality and allows you to create your own set of code blocks to process selected object(s). Inplace debugging is available. See example below for more details.

3. Script language, `OUTPUT` statement:

- `AsUpdateOrInsert` option added.

Example:

```
   OUTPUT 'C:MyScriptsdata.sql' ASUPDATEORINSERT;
SELECT * FROM MYTABLE ORDER BY ID;
OUTPUT;
COMMIT;:
```

This produces a script containing `UPDATE` or `INSERT` statements.

4. Database Registration Info / Log Files:

Added the possibility to include a date part into log file names. This allows you to create daily/monthly logs automatically. The following substrings in a log file name will be replaced with a current date:

```
=date=yyyy-mm-dd
=date=yyyy-mm-dd%=<date format string>%
```

`=date=yyyy-mm-dd` is a short form of the date template and is equal to `=date=yyyy-mm-dd%=yyyy-mm-dd%`

Examples:

`D:MyLogsTestDB=date=yyyy-mm-dd.sql` - file name for a simple daily log.

`D:MyLogsTestDB=date=yyyy-mm-dd%=mmmm of yyyyyyyy-mm-dd%=date=yyyy-mm-dd%=yyyy.mm.dd%.sql` - a separate directory (`'January 2008'` etc.) will be created for each month.

5. Blob Viewer:

Added support for TIFF images.

6. ODBC Viewer:

Fixed the problem with exporting of memo-fields.

7. IBEBlock:

The following functions have been implemented:

- `ibec_GetRunDir` - returns the path of the currently executing program. (`IBExpert.exe` or `IBEScript.exe`).

Syntax:

```
function ibec_GetRunDir : string;
```

- `ibec_GetUserDBConnection` - returns pointer to the User Database (Options / Environment Options / User Database) if one is used. Otherwise this function returns `NULL`.

Syntax:

```
function ibec_GetUserDBConnection : variant;
Example:
    execute ibeblock
 as
 begin
   CRLF = ibec_CRLF();
   sTab = ibec_Chr(9);
   sLine = '==============================================';
        UserDB = ibec_GetUserDBConnection();
   if (UserDB is not null) then
   begin
     sMes = '';
          sHost = ibec_GetConnectionProp(UserDB, 'HostName');
     sFile = ibec_GetConnectionProp(UserDB, 'FileName');
     sServerVersion = ibec_GetConnectionProp(UserDB, 'ServerVersion');
     sDBSqlDialect = ibec_GetConnectionProp(UserDB, 'DBSqlDialect');
     sClientLib = ibec_GetConnectionProp(UserDB, 'ClientLib');
     sUser = ibec_GetConnectionProp(UserDB, 'UserName');
     sPass = ibec_GetConnectionProp(UserDB, 'Password');
     sNames = ibec_GetConnectionProp(UserDB, 'lc_ctype');
          iPageSize = ibec_GetConnectionProp(UserDB, 'PageSize');
```

```
        iSweep = ibec_GetConnectionProp(UserDB, 'SweepInterval');
        iODSMinorVersion = ibec_GetConnectionProp(UserDB, 'ODSMinorVersion');
        iODSMajorVersion = ibec_GetConnectionProp(UserDB, 'ODSMajorVersion');
             sMes = 'User Database properties' + CRLF + sLine + CRLF;
        sMes .= 'Database host: ';
        if (sHost = '') then
          sMes .= sTab + '(local)';
        else
          sMes .= sTab + sHost;
             sMes .= CRLF +
                'Database file: ' + sTab + sFile + CRLF +
                'Server version: ' + sTab + sServerVersion + CRLF +
                'Client library: ' + sTab + sClientLib + CRLF + CRLF +
                     'Page size, bytes: ' + sTab + ibec_Cast(iPageSize,

__typeString) + CRLF +
                'Sweep interval: ' + sTab + sTab + ibec_Cast(iSweep,

__typeString) + CRLF +
                'ODS version: ' + sTab + sTab + ibec_Cast(iODSMajorVersion,

__typeString) + '.' +
                     ibec_Cast(iODSMinorVersion, __typeString) + CRLF + CRLF

+
                     'Connection username: ' + sTab + sUser + CRLF +
                'Connection password: ' + sTab + sPass + CRLF +
                'Connection charset: ' + sTab + sNames + CRLF;
                  ibec_UseConnection(UserDB);
             sMes .= CRLF + CRLF + 'User Database tables' + CRLF + sLine + CRLF;
        for select rdb$relation_name
            from rdb$relations
            where (rdb$system_flag is null) or (rdb$system_flag = 0)
            order by rdb$relation_name
            into :RelName
        do
        begin
          RelName = ibec_Trim(RelName);
          sMes .= RelName + CRLF;
        end
        commit;
             ibec_ShowMessage(sMes);
      end
    end
```

- `ibec_ibe_GetActiveDatabaseID` - returns the unique identifier of the active (currently used) database within IBExpert. If there is no active database `ibec_ibe_GetActiveDatabaseID` returns `-1`.

Syntax:

```
function ibec_ibe_GetActiveDatabaseID : integer;
```

- `ibec_ibe_GetDatabaseProp` - returns the value of a specifed database property.

Syntax:

```
function ibec_ibe_GetDatabaseProp(DatabaseID : integer; PropertyName : string) : variant;
```

The following properties are available:

`ALIAS`	alias of the registered database
`CLIENTLIB`	name of client library file specified in the database registration info
`SERVERNAME` or `HOSTNAME`	server name
`FILENAME` or `DBNAME`	database file name
`PASSWORD`	password specified in the database regstration info
`USERNAME` or `USER_NAME` or `USER`	user name
`ROLENAME` or `ROLE_NAME` or `ROLE`	role name
`NAMES` or `LC_CTYPE` or `CHARSET`	connection charset
`CONNECTIONSTRING` or `CONNECTION_STRING`	connection string
`ACTIVE` or `CONNECTED`	returns `TRUE` if the database is active and `FALSE` if it is not

Example:

```
      execute ibeblock as
    begin
      CRLF = ibec_CRLF();
      ActiveDB = ibec_ibe_GetActiveDatabaseID();
      if (ActiveDB is not null) then
      begin
        if (ActiveDB = -1) then
          Exit;
        sAlias = ibec_ibe_GetDatabaseProp(ActiveDB, 'Alias');
        sClientLib = ibec_ibe_GetDatabaseProp(ActiveDB, 'ClientLib');
        sHost = ibec_ibe_GetDatabaseProp(ActiveDB, 'HostName');
```

```
    sFileName = ibec_ibe_GetDatabaseProp(ActiveDB, 'FileName');
    sPassword = ibec_ibe_GetDatabaseProp(ActiveDB, 'Password');
    sUser = ibec_ibe_GetDatabaseProp(ActiveDB, 'User');
    sRole = ibec_ibe_GetDatabaseProp(ActiveDB, 'Role');
    sCharset = ibec_ibe_GetDatabaseProp(ActiveDB, 'Names');
    sConnectionStr = ibec_ibe_GetDatabaseProp(ActiveDB, 'ConnectionString');
    bActive = ibec_ibe_GetDatabaseProp(ActiveDB, 'Connected');
         s = 'Database alias: ' + sAlias + CRLF +
        'Client library: ' + sClientLib + CRLF +
        'Server name: ' + sHost + CRLF +
        'Database file name: ' + sFileName + CRLF +
        'User name: ' + sUser + CRLF +
        'Password: ' + sPassword + CRLF +
        'Role: ' + sRole + CRLF +
        'Charset: ' + sCharset + CRLF +
        'Connection string: ' + sConnectionStr;
         if (bActive) then
      s .= CRLF + CRLF + 'Database is active.';
         ibec_ShowMessage(s);
  end
end
```

8. Integrated web-based groupware "PHProjekt IBExpert Edition" usable with Firebird 1.5

IBExpert customers can now use PHProjekt with Firebird 1.5. The fully functional web-based groupware system offers many useful tasks and functions such as calendar, chat, trouble ticketing, contacts, mailing lists etc.

How to start it? Just start the example from and starting the WebForm the first time (on port 80), just enter http://localhost/phprojekt in your web browser and follow the instructions to install.

Attention: the first page shows a panic information and a link to the setup form. Just follow the link to start the installer. The current version still has some problems with Firebird 2.x, so we recommend using it at the moment only with Firebird 1.5.

PHProjekt is an Open Source Project and free software. For IBExpert Customers, we made some changes to the source code, to make it possible to use it with Firebird. The original version which can be downloaded from phprojekt.com still contains some errors for Firebird users.

The documentation for phprojekt can be found here in English: http://www.ibexpert.com/download/phprojekt/phprojekt_en.pdf

and here in German: http://www.ibexpert.com/download/phprojekt/phprojekt_de.pdf

Important: We offer no official support for this product, but weve been using it for a long time with InterBase and Firebird and we really like it.

9. A lot of minor bug fixes and small improvements.

IBExpert 2007.12.08

1. IBExpertXOCR Command Line Version available:

What is IBExpertXOCR?

IBExpertXOCR is an optical character recognition command line utility, able to convert scanned images into text files. This increases the value of all your documents, since it makes it easy to store these files in a database. A full text search engine can be created using simple SQL statements.

What are the System requirements?

The installation requires about 15 MB. A typical OCR process takes between 2 and 5 seconds per page, depending on the processor speed and the complexity and quality of your scanned image. For best results, the scanner should work with a minimum of 300 dpi and store the images in TIFF format. It can be used under Windows 2000 or upwards. It can also be used under Linux and Wine.

What are the major advantages for processing the recognized documents inside a database? Howcan I process the documents automatically?

A fully functional RDBMS such as Firebird allows you to easily store the images and text files in the database and use simple `SELECT` statements to define your result set. Operators such as `CONTAINING`, `IN` or `LIKE` provide fast access even in medium-sized databases. A typical search on a 5 GB database with about 50,000 documents takes less than a single second even for complex results. Based on the recognized text, you can add, for example, a database trigger to create links to existing records in your customer table or whatever you want. When a customer invoice is scanned, there is often a text such as `Customer No:` in front of the required number. All new text records can be searched using a trigger and simple functions from UDF libraries to detect the document type and extract such numbers. The Firebird database can be used with billions of data sets. For very large amounts of documents, we can integrate the scalable memory-based full-text search engine IBExpertFTS. This can handle millions of documents and display the result extremely fast.

Howto integrate IBExpertXOCR in my environment? Howto connect a scanner?

The calling interface is extremely simple. Just place your documents in a directory, call xocr.exe with the file names as a parameter and after processing, it will store the recognized text in a text file with the same file name and a changed file name extension. Most modern scanners have a programmable TWAIN interface, but in our experience it is usually incompatible to other scanners. We prefer using a scanner with a file interface. Very reliable machines can be found at Fujitsu or Plustek. IBExpertXOCR includes the command line interface that can be used from any development environment, for example Delphi, C++, VB, C#, batch files or any other software, which supports calling other applications. The created text files and scanned images can be loaded in any other database that supports blob columns.

What characters are supported?

All supported characters can be found here:

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
€						†	‡			Š		Œ		Ž							•					š		œ		ž	Ÿ
	¡	¢	£	¤	¥		§		©		«			®	¯	°	±				µ	¶	·				»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

All typical western European characters and business fonts such as Arial, Times, etc. are supported. Handwriting or artistic fonts are not supported.

The IBExpertXOCR Trial Version can be downloaded here: http://www.ibexpert.com/xocrtrial/

Pricing?

The IBExpertXOCR Single License costs EUR 499.00. The license is created for a specific computer name. You can purchase IBExpertXOCR in our shop in (select the product group software): .

IBExpert 2007.12.01

We recommend you uninstall older versions before installing the new IBExpert Version. Please select all IBExpert products in the Windows - ControlCenter / Add or Remove Software. All registered databases are stored in the directory `C:Documents and Settings[user]ApplicationdataHK-SoftwareIBExpert` or, if used, in the IBExpert User Database. Please backup these before uninstalling.

1. To-do List implemented (Tools / To-do list):

- This new feature can be used to organize your database development. You can add ***ToDo Items*** for each object in the database.

2. Database Comparer:

- Firebird 2.1 support added.

3. Log Manager:

- Generation of logging trigger bodies now based on the IBEBlock feature.

4. IBEBlock:

- Added the possibility to pass arrays into IBEBlocks (`EXECUTE IBEBLOCK`).

Example:

```
execute ibeblock
as
begin
  MyBlock = 'execute ibeblock (inparam variant)
             as
             begin
               ibec_ShowMessage(inparam[0] || inparam[1] || inparam[2]);
             end';       MyVar[0] = 'Hello';
  MyVar[1] = ', ';
  MyVar[2] = 'World!';
  execute ibeblock MyBlock(MyVar);
end
```

- Support of `CREATE/ALTER SEQUENCE` (Firebird 2.x) in the `ibec_ExtractMetadata` function (`UseSequence` option).

5. Script Executive:

- Added the possibility to show DML statements (`INSERT`, `UPDATE`, `DELETE`) in the Script Explorer tree. Use the Script Explorer context menu to display DML statements.

6. Database Monitor:

- Fixed the problem with the loading of monitor queries when working with Firebird 2.1.

7. Table and View Editor, Triggers tab:

- Added the option to set active/inactive for more than one trigger simultaneously.

8. Extract Metadata:

- Now supports `CREATE/ALTER SEQUENCE` (Firebird 2.x).

- Added the possibility to extract table data when extracting into VCS files.
- Fixed the problem with the extraction of array domains dimensions.

9. Database Registration:

- *Trusted Authentication* option added (Firebird 2.1).

10. A lot of minor bug fixes and small improvements.

11. Changes in the installer and updated Service Tools

IBExpertSQLMonitor, IBExpertJobScheduler, IBExpertTransactionMonitor and the IBExpertBackupRestore Service were updated. Due to changes in the installer, we strongly recommend uninstalling older versions before installing the new IBExpert Version. Please select all IBExpert products in the Windows ControlCenter - Add or Remove Software. All registered databases are stored in the directory `C:Documents and SettingsApplicationdataHK-SoftwareIBExpert` or, if used, in the IBExpert User Database. Please backup these files before uninstalling.

IBExpert 2007.09.25

1. Tools / ODBC Viewer:

- The ODBC Viewer allows you to browse data from any ODBC source available on your PC and also export data from a ODBC source into an SQL script or directly into a Firebird/InterBase database.

2. Services / Database Statistics:

- Added the possibility to automatically analyse tables/indices statistics and the highlighting of possible problem tables/indices. This feature based on the IBEBlock functionality and is therefore is fully customizable.

3. Tools / Extract Metadata:

- Added the Use `CREATE OR ALTER` for procedures and triggers option.
- Added the Dont use `SET TERM` command. `SET TERM` is not necessary for scripts executed by IBExpert/IBEScript but may be necessary when working with other tools.
- Now it is possible to create scripts larger than 2 GB.

4. Tools / Script Editor; IBEScript:

- Both now work with scripts larger than 2 GB.

5. SP/Trigger Parser:

- The SP/Trigger Parser now displays variables/parameters that may be not initialized or assigned but never used.

6. Blob Editor:

- Added syntax highlighting for Delphi forms (dfm).

7. Tools / Table Data Comparer:

- Added the possibility to synchronize generators.

8. Table Editor / Fields:

- Added the possibility to create a Foreign Key from the context menu of the columns list.

9. Tools / Script Editor:

- The Script Explorer now displays IBEBlocks and Firebird Blocks.

10. A lot of minor bugfixes and small improvements.

Contents

The IBExpert online documentation can be viewed online under http://ibexpert.net/ibe/pmwiki.php?n=Doc.IBExpert. It can be downloaded from http://www.ibexpert.info/documentation/documentation.zip. (For download instructions please refer to the IBExpert Help menu.)

The first view displays the complete list of contents. If you are looking for help about a specific subject use the Search: function (top right).

In the meantime, should you be unable to find a solution to your problem here, please use one of our newsgroups (in English, German, French and Russian). Should you have any comments or queries directly regarding the Help documentation, or wish to contribute your own articles, please contact documentation@ibexpert.com.

Additional Help files

This menu item has been included for third party help files, intended for those third party components included in the IBExpert PlugIns menu. Such Help files can be installed using the IBExpert Options menu: Environment Options / Additional Help.

The installed help files appear here as an additional menu item.

Product Home Page

The IBExpert Help menu item Product Home Page does none other than open the http://ibexpert.net/ibe/ homepage, which provides product information, news, support, downloads, plugins, purchase and a contact email, in English and German languages.

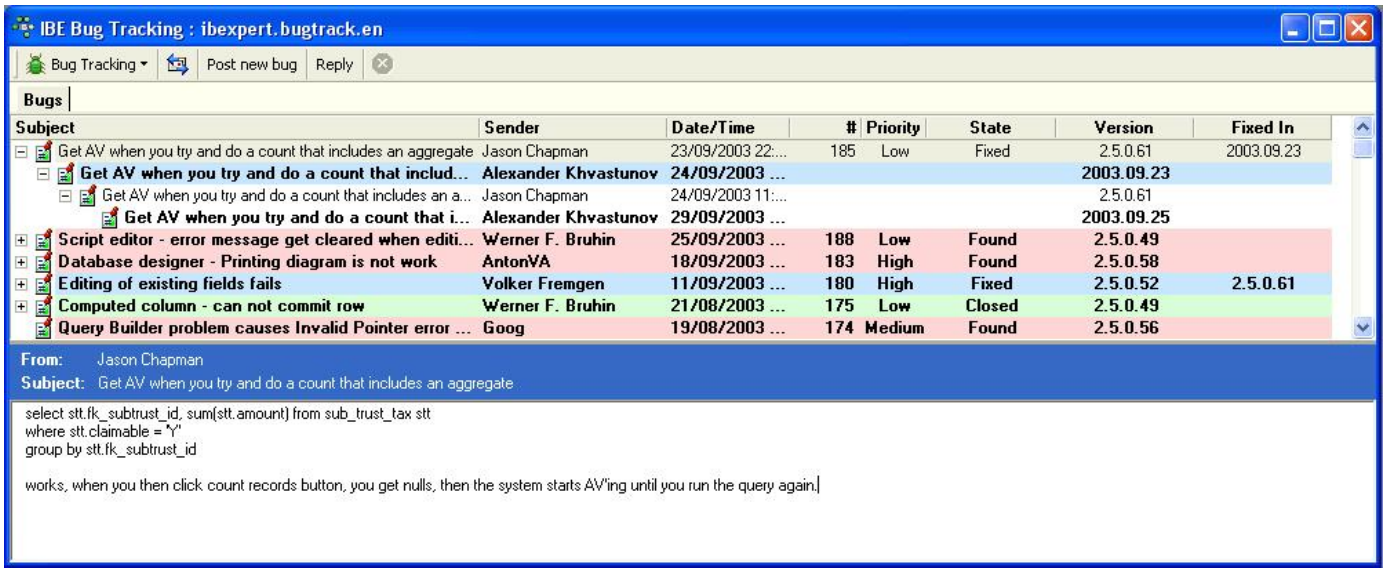
Send bug reports to

The IBExpert Help menu item *Send Bug Reports To* allows you to inform us at IBExpert of any bugs discovered or suggestions you may wish to make. The *From*, *To* and *Re* fields are automatically filled; it is merely necessary to type in the message, if possible with an example, in order to enable us to reproduce the operations leading to the problem, and send.

All bug reports can be followed in the Bug Track System.

Bug Track System

The IBExpert Bug Track System was introduced on the 28.04.2003 in version 2.5.0.38. It allows all users to post and follow all bugs discovered and their current status.



There are currently two bug track groups: English and Russian. Each bug reported receives a number and priority. It is also possible to follow the status (i.e. *closed*, *found*, *fixed*), follow correspondence (by clicking on the + button or using the [+] key), and view the IBExpert version and date including the fix.

If you want to post a bug directly from the Bug Track System (as an alternative to the IBExpert Help menu item Send Bug Reports To), it is first necessary to specify your signature. Simply click on the *Configure Bug Tracking System* icon, to spring to the Environment Options / IBExpert Bug Track window and input the required information.

Using either the Bug Track pull-down menu or the relevant icons in the toolbar, it is possible to reply to items and send and receive.

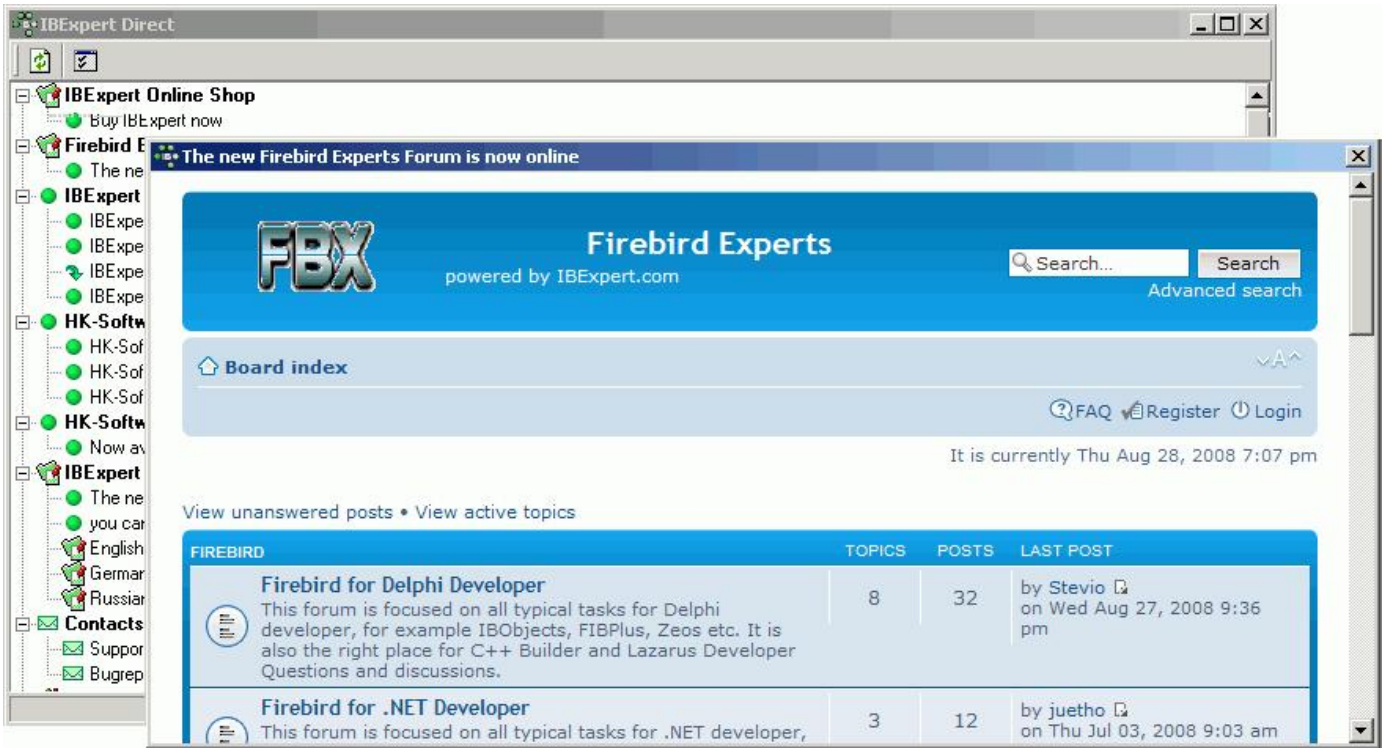
About

The IBExpert Help menu item About calls the so-called IBExpert splash screen, including the IBExpert logo and current installed version number, with a full copy of the software license on the second page (click the License tab).

Since October 2003 we have introduced a new version numbering system based on the date, as opposed to the more traditional version numbering system.

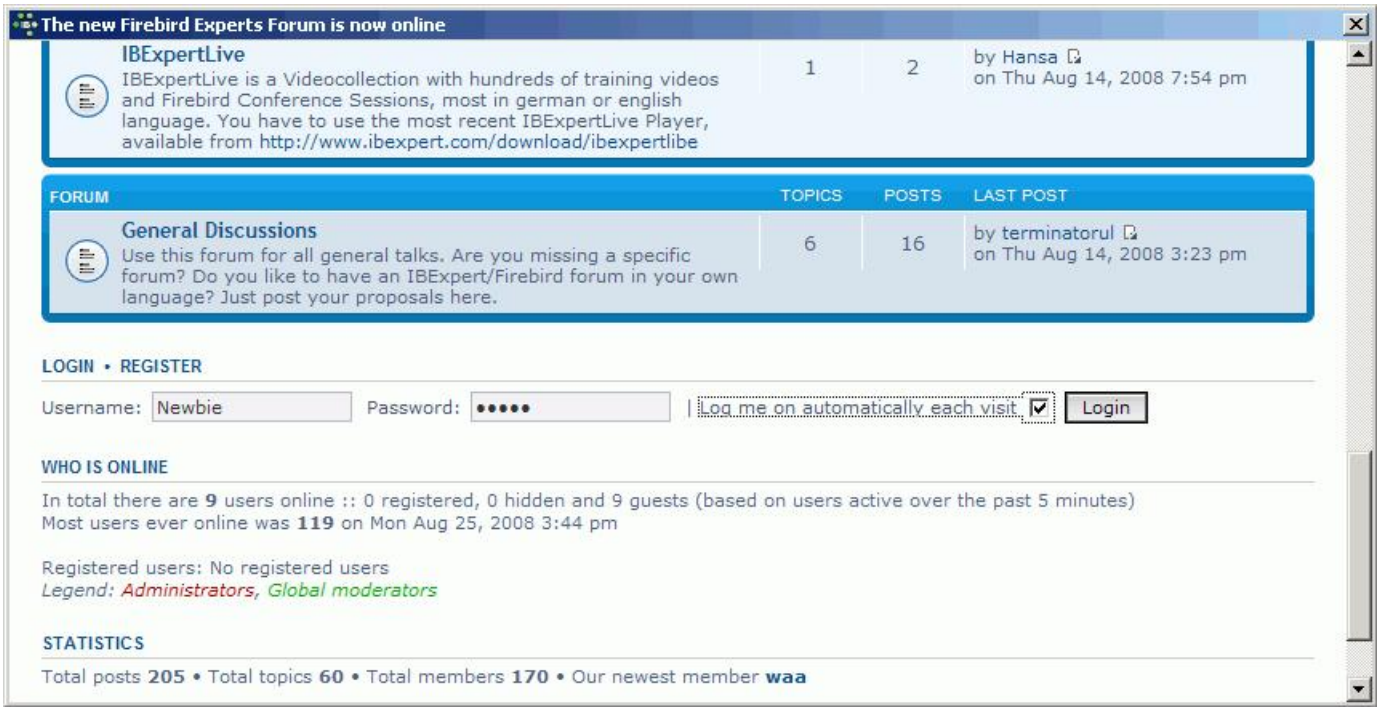
IBExpert Direct...

The IBExpert Help menu item *IBExpert Direct...* opens two windows offering comprehensive user information and support.

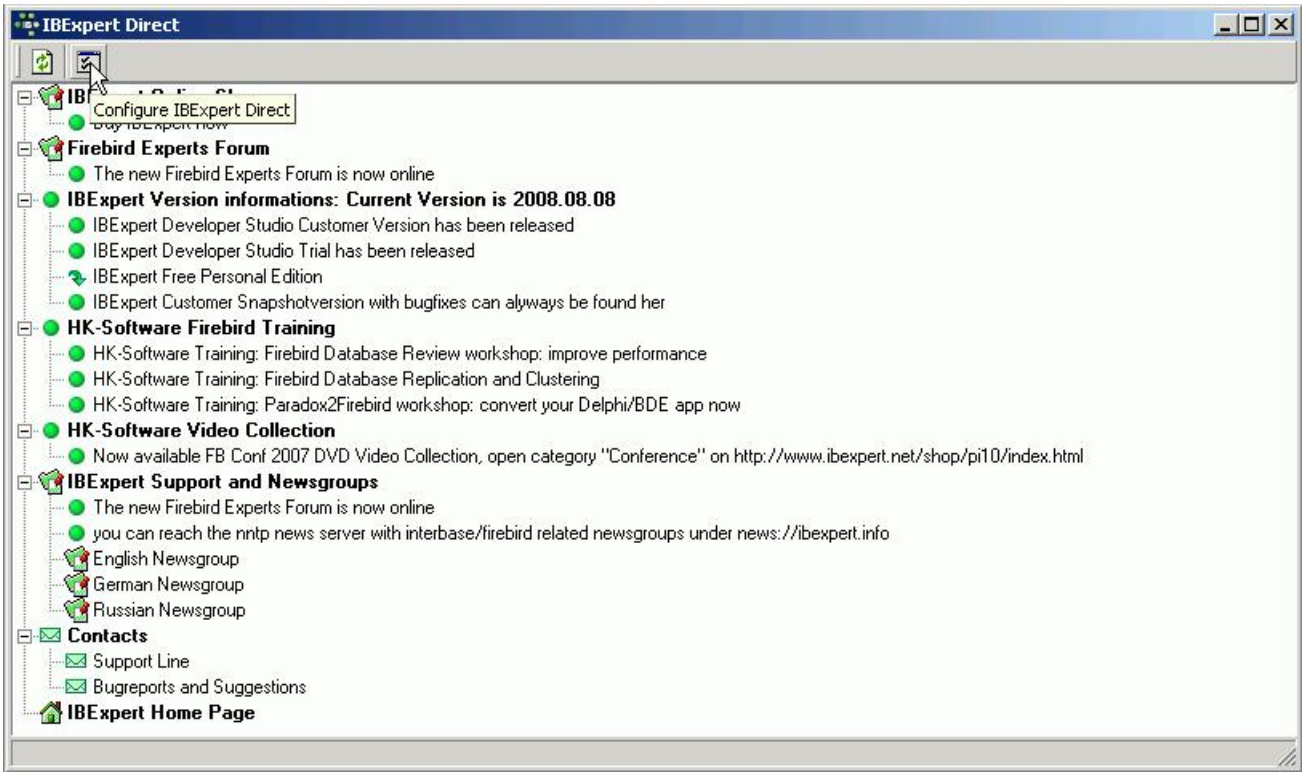


The IBExpert Firebird Experts forum provides help and answers to your questions regarding all IBExpert Developer Studio tools as well as Firebird. IBExpert Direct provides all users with important information concerning IBExpert, such as new versions, documentation, downloads, plugins, newsgroups, as well as contact addresses and a direct link to the IBExpert home page, http://ibexpert.net/ibe/.

In the forum you can view postings and follow discussions as a guest, or register (user name and password) in order to participate.



The IBExpert Direct window provides direct links to its online shop, forums, software download areas, and also links to information about training, online videos, newsgroups and contact adresses.



The *Configure IBExpert Direct* icon opens the IBExpert Options menu item, Environment Options / IBExpert Direct dialog, where it is possible to specify how often the network should be polled for new items, and to configure a proxy server if wished.

Download Firebird / Purchase InterBase

These last three items in the IBExpert Help menu provide direct links to the software producers, for those wishing to purchase or download InterBase or Firebird.

- **Download InterBase Open Edition:** currently invalid.
- **Buy Borland InterBase:** opens the link: http://www.borland.com/interbase/. By clicking *DOWNLOADS*, it is possible to download the newest trial version (currently InterBase 2007 Developer, April 9, 2008). By clicking *PURCHASE*, you can gain access to the online web shop, or search for your nearest retailer.
- **Download Firebird:** opens the link: http://www.firebirdsql.org/. Please refer to Download and Install Firebird for further details.

FAQs

Here we have attempted to list some of the more frequently asked questions regarding IBExpert. Should you not be able to find a solution to your problem under the links provided here or elsewhere within the IBExpert documentation, please contact one of our newsgroups:

- Username: **ibexpert**
- Password: **ibexpert**

news://ibexpert.info/interbase.ibexpert.de German language
news://ibexpert.info/interbase.ibexpert.en English language
news://ibexpert.info/interbase.ibexpert.ru Russian language
news://ibexpert.info/interbase.ibexpert.fr French language

or send an email to documentation@ibexpert.com or support@ibexpert.com, or use our Bug Track System in IBExpert.

How do I connect to a database?

See Connect to an existing Database and Register Database.

If you are experiencing problems with a remote connection, please refer to Communication Diagnostics.

Why do I need to register a database?

See Register Database.

How do I create a new database?

See Create Database.

How do I use the SQL Editor?

See SQL Editor.

Why are new fields not displayed on the Data page in the Table Editor?

We have often been asked the question why, after creating a new field on the Fields page, the new field is not immediately displayed on the Data page.

This is because you have to commit or rollback the current data transaction using the corresponding icons on the Table Editor toolbar. As this transaction was started before you added a new field you can't see it until you have committed.

What is the Performance Analysis for?

See Performance Analysis.

What is the Query Plan?

See Plan Analyzer.

How can I optimize an SQL Statement?

See Optimizing an SQL Statement.

How do I debug a stored procedure?

See Debug Procedure.

Are there typical windows for all Object Editors?

See Database Objects.

How can I use the view and procedure version control?

See View / Version History.

What is the Project View in the DB Explorer for?

See Project View.

What is the Recent list in the DB Explorer for?

See Recent List.

How do I use the integrated Report Manager?

See Report Manager.

Why can I not see the index statistics in the Table Editor?

Use the right-click menu directly on the Indices page in the Table Editor and select the menu item *ShowStatistics*.

Why does the index selectivity/statistics not change?

See Recompute Selectivity of all Indices.

Indices do not seem to work on my newly installed application

See Recompute Selectivity of all Indices.

How can I integrate the online Help files into IBExpert?

Please refer to IBExpert Help menu.

Import CSV Files

Here are a few questions that have arisen with regard to importing CSV files.

1. In the examples a database field gets the correct value if the imported data is numeric. Does truncation occur if it is not an integer?

`INSERTEX` itself doesn't truncate numeric values. Of course, if you're inserting numeric value into `Integer` fields the server will truncate it.

2. Can I import dates and if so what ASCII format does it accept for `DATE` or `TIMESTAMP` columns or do I need to perform my own external conversion of dates & times to a 32 bit integer?

You can import dates and `INSERTEX` accepts any date format known by the server. For example, `1.08.2004` or `1-AUG-2004`.

3. If the imported string is longer than I specify for `VARCHAR` or `CHAR` does truncation occur?

Yes, it does.

When I use Norton AntiVirus which IBExpert files must I include in the *Exclusion List*?

`IBExpert.stg`.

You will find this file under:

```
\Documents and Settings\<user>\Application Data\HK-Software\IBExpert
```

But a much better solution is to use the IBExpert User Database. Please refer to the IBExpert menu item Options / Environment Options / IBExpert User Database? for further information.

Can I alter IBExpert Table Editor default to show the Data page instead of the Fields page?

This question has often been raised, particularly by developers and administrators who only use the *Fields* page during the database design stage, but regularly use the *Data* page to adminstrate existing database tables.

The default setting can be specified under the IBExpert menu item Options / Object Editor Options / Tables Editor / Active Page.

I cannot change the language in Environment Options

Should you not be able to see the full list of languages in the drop-down list, either delete the `ibexpert.lng` file or rename the `english.lng` file, found in the IBExpert Languages directory, to `ibexpert.lng`, and place this in the main IBExpert directory.

How do I find the procedures, trigger and views, that do not use an index in their operations?

Just open the IBExpert menu item Tools / Stored Procedure/Trigger/View Analyzer and press [F9]. This analyzes all objects and displays all parts that do not use an index in red. To modify these objects, just double click the line. A well-designed database should have no red line.

How do I find the procedures and triggers that have typical type casting problems?

A typical problem that is often not so easy to find is when a `varchar(20)` column is copied into a `varchar (10)` variable. In most cases it causes no problems, but when the source has more than 10 characters you get a runtime error. This will typically only happen in your customer's database! To find these errors, just open the IBExpert menu item Tools / Stored Procedure/Trigger/View Analyzer and press [F9].

How do you know if your database server garbage collection is working?

Just open your database, open the IBExpert menu item Services / Database Statistics and press [F9]. On the summary page you can see a versions column with subcolumns *versions*, *version length* and *max versions*. When the garbage collection is working properly, there should be only very low values for *versions* and *max versions*. If there are higher values, your garbage collection does not work properly, which might be due to several reasons, is however typically due to improper transaction handling in your application.

How do I change the Character Set of all tables in a database?

In the IBExpert menu item Tools / Extract Metadata, you can create a script that recreates the database and also inserts the data including blob data. In this script you can perform a search and replace for the character set name and after renaming the original file execute the script again.

Addenda

- Firebird License Agreement
- Copy of Firebird Information File
- IBExpert Toolbars

Firebird License Agreement

INTERBASE PUBLIC LICENSE

Version 1.0

1. Definitions.

1.0.1. "Commercial Use" means distribution or otherwise making the Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.
B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with, all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1. The Initial Developer Grant. The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and
(b) under Patents Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).
(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.
(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by: i) the modification of the Original Code or ii) the combination of the Original Code with other software or devices.

2.2. Contributor Grant. Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and
(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).
(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version; 3) for infringements caused by: i) third party modifications of Contributor Version or ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Application of License. The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code. Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications. You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters

(a) Third Party Claims.

If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must include a text file with the Source Code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.

If Contributor's Modifications include an application programming interface and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations.

Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

3.5. Required Notices. You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipient's rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions. You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipient's rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works. You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code.

6. Versions of the License.

6.1. New Versions. Inprise Corporation ("Inprise") may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. Effect of New Versions. Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Inprise. No one other than Inprise has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works. If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must (a) rename Your license so that the phrases "Mozilla", "MOZILLAPL", "MOZPL", "Netscape", "MPL", "NPL", "Inprise", "ISC", "InterBase", "IB" or any confusingly similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Mozilla Public License and Netscape Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

6.4 Origin of the InterBase Public License. The InterBase Public License V 1.0 is based on the Mozilla Public License V 1.1 with the following changes:

1. The license is published by Inprise Corporation. Only Inprise Corporation can modify the terms applicable to Covered Code.
2. The license can be modified and used for code which is not already governed by this license. Modified versions of the license must be renamed to avoid confusion with Netscape's or Inprise Corporation's public license and must include a description of changes from the InterBase Public License.
3. The name of the license in Exhibit A is the "InterBase Public License".
4. The reference to an alternative license in Exhibit A has been removed.
5. Amendments I, II, III, V, and VI have been deleted.
6. Exhibit A, Netscape Public License has been deleted
7. A new amendment (II) has been added, describing the required and restricted rights to use the trademarks of Inprise Corporation.

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as "Participant") alleging that:

> (a) such Participant's Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1 and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.
> (b) any software, hardware, or device, other than such Participant's Contributor Version, directly or indirectly infringes any patent, then any rights granted to You by such Participant under Sections 2.1(b) and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed, or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Participant's Contributor Version directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal

Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorney's fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

13. MULTIPLE-LICENSED CODE.

Initial Developer may designate portions of the Covered Code as "Multiple-Licensed". "Multiple-Licensed" means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the IPL or the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A. EXHIBIT A - InterBase Public License.

The contents of this file are subject to the InterBase Public License Version 1.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.Inprise.com/IPL.html Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code was created by Inprise Corporation and its predecessors. Portions created by Inprise Corporation are Copyright (C) Inprise Corporation. All Rights Reserved. Contributor(s): _____.

AMENDMENTS

I. Inprise and logo. This License does not grant any rights to use the trademarks "", "InterBase," "Java" or "JavaScript" even if such marks are included in the Original Code or Modifications.

II. Trademark Usage.

II.1. Advertising Materials. All advertising materials mentioning features or use of the covered Code must display the following acknowledgement: "This product includes software developed by Inprise Corporation."

II.2. Endorsements. The names "Inprise," "InterBase," "ISC," and "IB" must not be used to endorse or promote Contributor Versions or Larger Works without the prior written permission of Inprise.

II.3. Product Names. Contributor Versions and Larger Works may not be called "Inprise" or "InterBase" nor may the words "Inprise" or "InterBase" appear in their names without the prior written permission of Inprise Corporation.

IBExpert toolbars

The individual IBExpert toolbars are listed in more detail below:

Toolbar Database

This standard toolbar can be viewed in the main IBExpert window. It can be blended in and out using the IBExpert View Menu / Toolbar (check boxes).

The icons (from left to right) can be used to execute the following operations:

1. Register Database [Shift + Alt + R]
2. Unregister Database [Shift + Alt + U]
3. Connect to Database [Shift + Ctrl + C]
4. Disconnect from Database [Shift + Ctrl + D]
5. Reconnect to Database
6. Create Database
7. Exit [Alt + F4]

These items can also be found in the main IBExpert Database menu. To alter, customize or reset this toolbar, please refer to Toolbars.

Toolbar Edit

This standard toolbar can be viewed in the main IBExpert window. It can be blended in and out using the IBExpert View Menu / Toolbar (check boxes).

The icons (from left to right) can be used to execute the following operations:

1. Load from File (Ctrl + L). The downward arrow produces a pull-down list of the most recent files.
2. Save to File (Ctrl + S). The downward arrow produces a pull-down list of the most recent files.
3. Cut (Ctrl + X)
4. Copy (Ctrl + C)
5. Paste (Ctrl + V)
6. Find (Ctrl + F)
7. Search again (F3)
8. Replace (Ctrl + R)

These items can also be found in the main IBExpert Edit menu.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Tools

This standard toolbar can be viewed in the main IBExpert window. It can be blended in and out using the IBExpert View Menu / Toolbar (check boxes).

The icons (from left to right) can be used to execute the following operations:

1. SQL Editor (F12)
2. New SQL Editor (Shift + F12)
3. Query Builder
4. Script Executive (Ctrl + F12)
5. SQL Monitor (Ctrl + M)
6. Search in Metadata (Shift + Alt + F)
7. Extract Metadata
8. Print Metadata
9. User Manager
10. Grant Manager
11. Report Manager
12. Blob Viewer/Editor

These items can also be found in the main IBExpert Tools menu. To customize or reset this toolbar, please refer to Toolbars.

Toolbar New Database Object

This standard toolbar can be viewed in the main IBExpert window. It can be blended in and out using the IBExpert View Menu / Toolbar (check boxes).

The icons (from left to right) can be used to execute the following operations:

1. New Domain
2. New Table
3. New View
4. New Procedure
5. New Trigger
6. New Generator
7. New Exception
8. New UDF
9. New Role

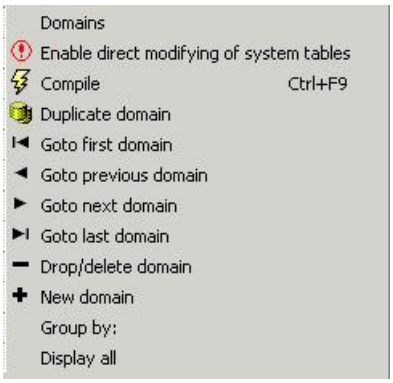
These items can also be found in the main IBExpert Database menu, or in the IBExpert DB Explorer by clicking the right mouse key to offer a context-sensitive option for the selected database object.

Alternatively [Ctrl + N] can be used in the DB Explorer to create new objects (providing an object type has been selected).

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Domain Editor

The standard toolbar for the Domain Editor includes the following icons:

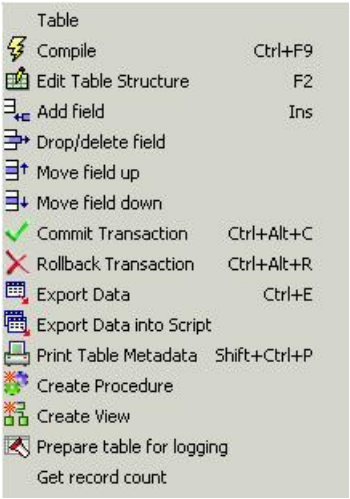


These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Table Editor

The standard toolbar for the Table Editor includes the following icons:

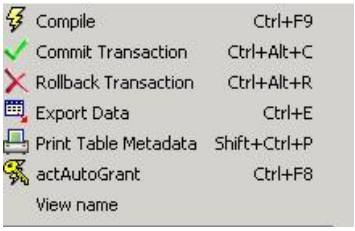


These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar View Editor

The standard toolbar for the View Editor includes the following icons:

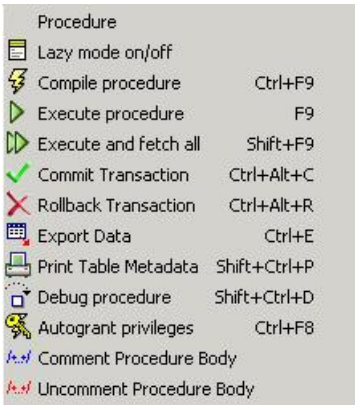


These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Procedure Editor

The standard toolbar for the Procedure Editor includes the following icons:



These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Debug Procedure

The toolbar for the Debug Procedure Editor includes the following icons:

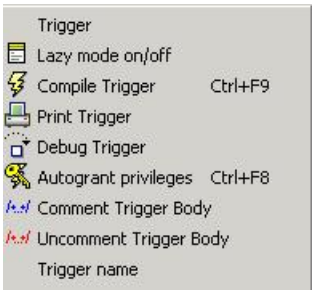
The icons (from left to right) can be used to execute the following operations:

1. Debugger drop-down menu
2. Drop-down list of registered databases
3. Toggle breakpoint [F5]
4. Reset [Ctrl + F2]
5. Parameters [Shift + Ctrl + P]
6. Run [F9]
7. Pause [Ctrl + P]
8. Skip statement
9. Step Over [F8]
10. Trace Into [F7]
11. Run to cursor [F4]

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Trigger Editor

The standard toolbar for the Trigger Editor includes the following icons:



These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Generator Editor

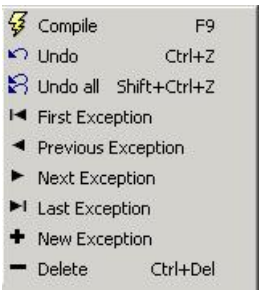
The standard toolbar for the Generator Editor includes the following icons:

These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Exception Editor

The standard toolbar for the Exception Editor includes the following icons:

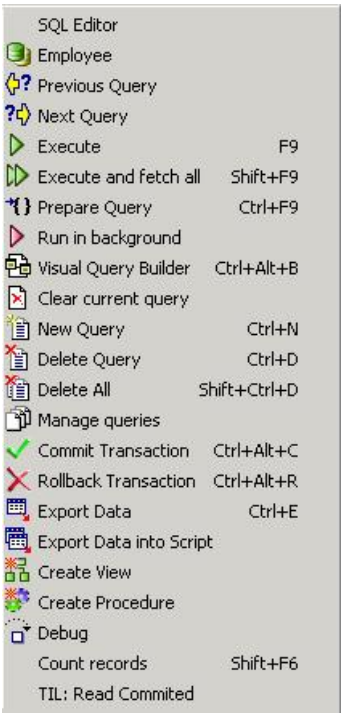


These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

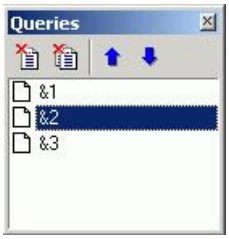
To customize or reset this toolbar, please refer to Toolbars.

Toolbar SQL Editor

This toolbar was completely revised in IBExpert version 2006.10.14. It can be viewed in the Tools / SQL Editor dialog and includes the following icons:



Further icons not displayed in the drop-down menu include Visual Query Builder, *Debug* and *Count Records* [Shift + F6]. And new to IBExpert version 2006.10.14 is the *Query Manager* icon, which allows you to move, remove and rename the most recently used queries.



Individual icons can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

It is also possible to quickly change the *Transaction Isolation Level* (TIL) for a separate SQL Editor. There is a corresponding button on the right-hand side of the SQL Editor toolbar which allows selection of one of the following isolation levels: *Snapshot*, *Read committed*, *Read-only table stability*, *Read-write table stability*.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Navigation

The navigational toolbar can be found on the Table Editor's Data page, the View Editor's Data page and in the SQL Editor on the Results page and includes the following icons:

The icons (from left to right) can be used to execute the following operations:

1. Apply filter
2. Show Filter Panel (Ctrl + Alt + F)
3. Quick Add Filter Criteria
4. Record Number
5. Data Analysis (new to IBExpert version 2004.10.25.1)
6. Show summary footer (new to IBExpert version 2004.8.5.1)
7. Display data as Unicode [F3] (new to IBExpert version 2004.8.26.1)
8. First
9. Previous
10. Next
11. Last
12. Insert
13. Delete
14. Edit
15. Save Updates
16. Cancel Updates
17. Refresh

To the right the number of records fetched is displayed.

Toolbar Filter Panel

The navigational toolbar can be found on the Table Editor's Data page, the View Editor's Data page and in the SQL Editor on the Results page when the *ShowFilter Panel* is activated, and includes the following icons:

The icons (from left to right) can be used to execute the following operations:

1. Apply Filter
2. Add New Criteria (Ins)
3. Delete Criteria (Ctrl + Del)
4. Vertical Layout (Shift + Ctrl + L)
5. Count Records
6. Count filtered records automatically (checkbox option)

Toolbar SQL Query Builder (Visual Query Builder)

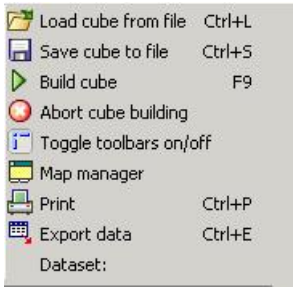
This toolbar can be viewed in the Tools / SQL Query Builder dialog and includes the following icons:

These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Data Analysis (PivotCubeForm)

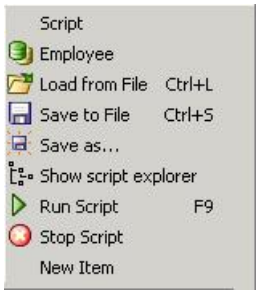
This toolbar can be viewed in the IBExpert Tools / Data Analysis dialog. The icons (from left to right) can be used to execute the following operations:



To customize or reset this toolbar, please refer to Toolbars.

Toolbar Script Executive

This toolbar can be viewed in the Tools / Script Executive dialog and includes the following icons:



The first item on the left, the pull-down menu detailing the most important operations, also includes the all-important *Add CONNECT statement*. To alter, customize or reset this toolbar, please refer to Toolbars.

Toolbar Dependencies Viewer

This toolbar can be viewed in the Tools / Dependencies Viewer dialog and includes the following icons:

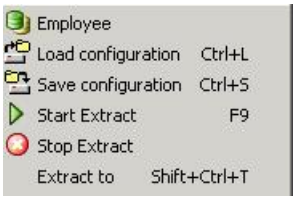
1. Refresh
2. Clear All
3. Print
4. Stop
5. Max level
6. Don't check domain dependencies (checkbox)
7. Show domains [Ctrl + D]
8. Show tables [Ctrl + T]
9. Show views [Ctrl + V]
10. Show triggers [Ctrl + R]
11. Show procedures [Ctrl + P]
12. Show generators [Ctrl + G]
13. Show exceptions [Ctrl + E]
14. Show UDFs [Ctrl + U]

These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Extract Metadata

This toolbar can be viewed in the Tools / Extract Metadata dialog and includes the following icons:



These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Print Metadata

This toolbar can be viewed in the Tools / Print Metadata dialog and includes the following icons:

The icons (from left to right) can be used to execute the following operations:

1. Select database including a pull-down list of available databases.
2. Preview
3. Print

To alter, customize or reset this toolbar, please refer to Toolbars.

Toolbar Grant Manager

This toolbar can be viewed in the Tools / Grant Manager dialog and includes the following icons:

These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Grants

This toolbar can be viewed in the Tools / Grant Manager dialog under *Grants on*, as well as in the Table Editor on the Grants page, and includes the following icons:

The icons (from left to right) can be used to execute the following operations:

1. Grant All
2. Grant All with GRANT OPTION
3. Grant to All with GRANT OPTION
4. Grant to All
5. Grant All to All
6. Revoke All
7. Revoke from All
8. Revoke All from All

Toolbar Localize IB Messages

This toolbar can be viewed in the Tools / Localize IB Messages dialog and includes the following icons:

The icons (from left to right) can be used to execute the following operations:

1. Load from File
2. Save to File
3. Undo
4. Goto Message Number
5. Find
6. Search Again
7. Export to Text File
8. Import from Text File

Toolbar Localize IBExpert

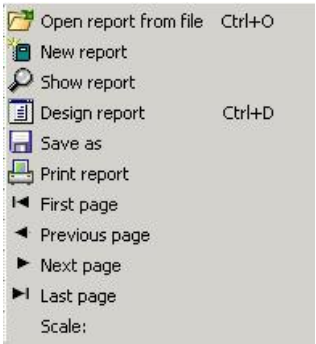
This toolbar can be viewed in the Tools / Localize IBExpert dialog and includes the following icons:

The icons (from left to right) can be used to execute the following operations:

1. Save to File
2. Find
3. Search Again
4. Export to Text File
5. Import from Text File
6. Font Charset (pull-down list)

Toolbar Report Manager

This toolbar can be viewed in the Tools / Report Manager dialog and includes the following icons:

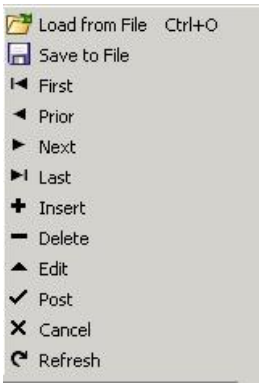


These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Blob Viewer/Editor

This toolbar can be viewed in the Tools / Blob Viewer/Editor dialog and includes the following icons:



These can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

To customize or reset this toolbar, please refer to Toolbars.

Toolbars Database Designer

These toolbars can be viewed in the Tools / Database Designer dialog. They comprise 4 individual toolbars and include the following icons:

Should IBExpert not load the toolbars automatically after starting the Database Designer, delete `IBExpert.tb` from the `\Documents and Settings\<user>\Application Data\HK-Software\IBExpert\` directory and restart IBExpert.

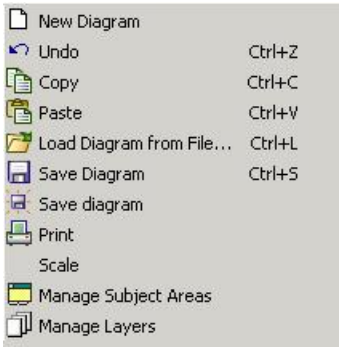
The individual menus are as follows:

1. Menu and Palette

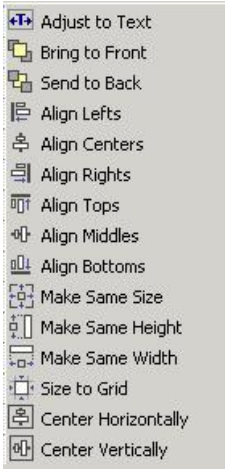
The icons (from left to right) can be used to carry out the following operations:

1. Pointer
2. Zoom in
3. Zoom out
4. Table
5. New View
6. Comment Box
7. Reference

2. Main



3. Layout



4. Font / Colors

The icons displayed in the *Main*, *Layout* and *Font / Colors* toolbars can be blended in and out by clicking the downward arrow to the right of the toolbar, and using the menu item *Add or Remove Buttons* to check the relevant icons in the menu list.

Since IBExpert version 2007.05.03. custom colors are saved in and restored from a `grc` file.

To customize or reset these toolbars, please refer to Toolbars.

Toolbar Server Properties/Log

This toolbar can be viewed in the Services / Server Properties/Log dialog and includes the following icons:

1. Select server (pull-down list of available servers)
2. Retrieve
3. Preview Log Report
4. Print

To customize or reset this toolbar, please refer to Toolbars.

Toolbar Database Statistics

This toolbar can be viewed in the Services / Database Statistics dialog and includes the following icons:

1. Select Database (pull-down list of available databases)
2. Analyze from File
3. Retrieve Statistic
4. Preview Log Report
5. Print
6. Export

To customize or reset this toolbar, please refer to Toolbars.

See also:
Toolbar options

IBEBlock

IBEBlock is a set of DDL, DML and other statements that are executed on the server and on the client side, and which include some specific constructions applicable only in IBExpert or IBEScript (excluding the free versions of these products), independent of the database server version.

IBEBlock (EXECUTE IBEBLOCK)

IBExpert version 2004.9.12.1 introduced an important, new and powerful feature EXECUTE IBEBLOCK.

What is IBEBLOCK?

It is a set of DDL, DML and other statements that are executed on the server and on the client side, and which include some specific constructions applicable only in IBExpert or IBEScript (excluding the free versions of these products), independent of the database server version.

With EXECUTE IBEBLOCK you will be able to:

- Work with different connections within the single IBEBLOCK at the same time.
- Move (copy) data from one database to another.
- Join tables from different databases.
- Compare data from different databases and synchronize them.
- Populate a table with test data using random values or values from other tables or even from other databases.
  ... and much more.

The syntax of IBEBLOCK is similar to that of stored procedures but there are many important extensions.

For example:

- You can use EXECUTE STATEMENT with any server, including InterBase 5.x, 6.x, 7.x.
- You can use one-dimensional arrays (lists) of untyped variables and access them by index.
- It isn't necessary to declare variables before using them.
- You can use data sets (temporary memory tables) to store data.
- Since IBExpert version 2005.02.12.1 there is added support for ROW_COUNT and ROWS_AFFECTED variables.
- Since version 2005.02.12.1 Code Insight also supports IBEBlock constants and functions.
  ... and much more.

You can execute single IBEBLOCKs via the SQL Editor. You can debug them in the SQL Editor too. They are debugged in the same way as stored procedures and triggers. Also you can include IBEBLOCKs into your scripts and execute these scripts as usual - using the Script Executive or IBEScript.exe.

This documentation describes the following topics:

- Procedural extensions of IBEBlock
- IBEBlock functions
- Examples of usage of IBEBlock

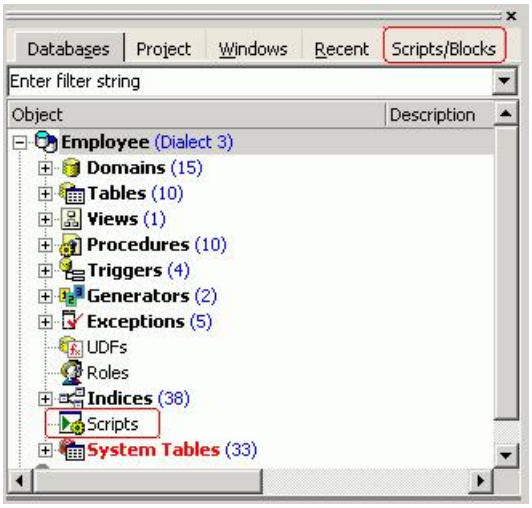
As this important feature is constantly being expanded and improved, some areas are still incomplete or in work. Check regularly for the latest revisions by using the What's New function in the online documentation.

Or post your question to: documentation@ibexpert.com.

Block Editor

The IBExpert Block Editor can be used to edit and execute IBEBlocks and IBEScripts.

The DBExplorer's *Scripts/Blocks* page was introduced in IBExpert version 2005.12.04. It displays all existing IBEScripts and IBEBlocks saved locally in the database. The DB Explorer *Database* page also has a new node, *Scripts*, displayed in all registered, connected databases. See also Drag 'n' Dropping Objects into Code Editors and the DB Explorer context-sensitive menu item, *Apply IBEBlock to selected object(s)*.



There are two ways to store blocks and scripts: (i) in a registered database or (ii) in the IBExpert User Database, which can be activated using the IBExpert Options Menu item, Environment Options / User Database.

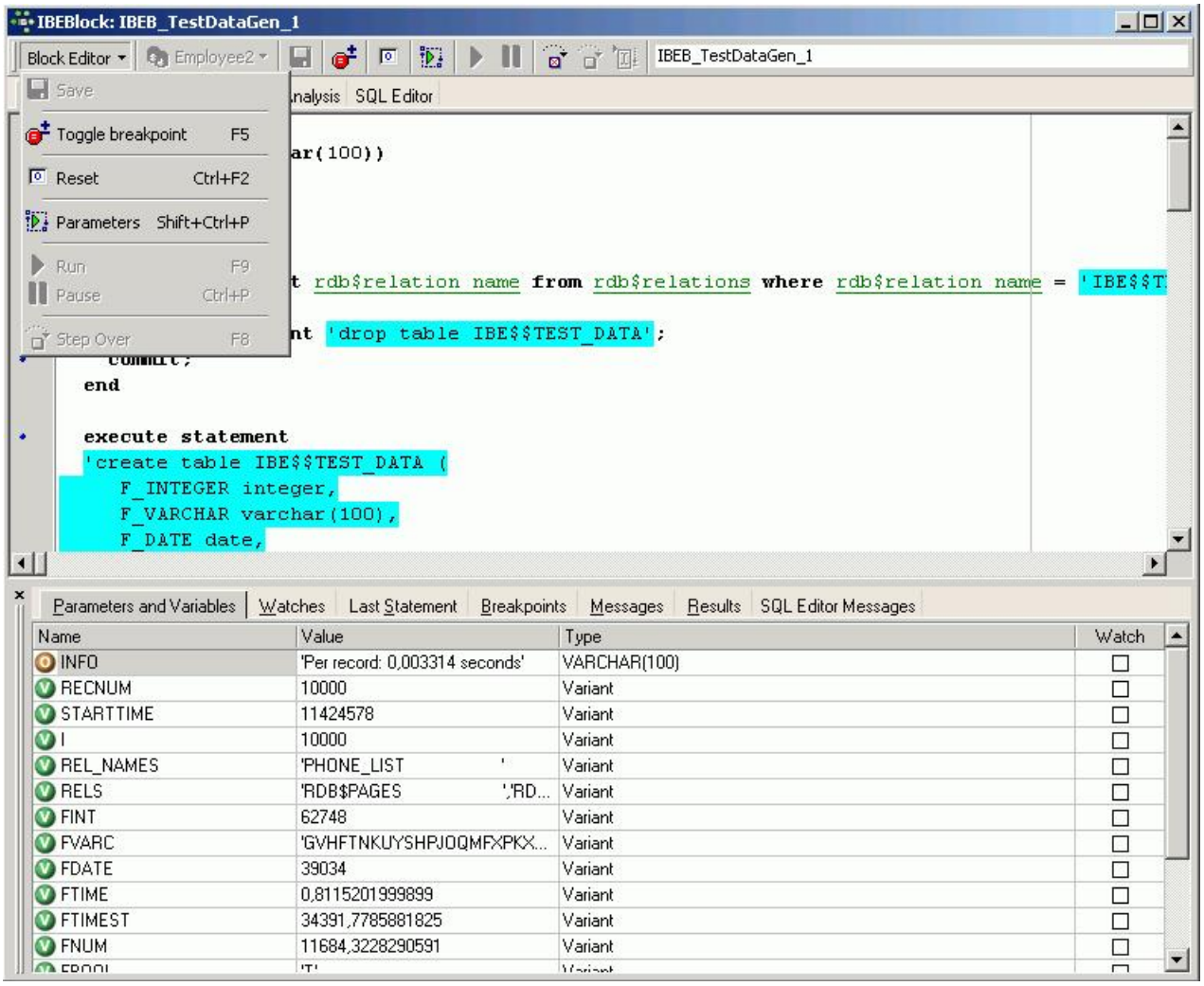
To create a new script in a registered database, click on the *Scripts* node in the connected database, and use the context-sensitive (right-click) menu to create a new script. You can also create IBEBlocks and Firebird 2 blocks (EXECUTE BLOCK) in this way within your database. Each script or block must have a unique name (up to 100 characters) within the database.

To create a new block or script in the User Database, first enable the option in the IBExpert Options menu, Environment Options / User Database and restart IBExpert. You should now see a new table in the Database Explorer: *Scripts/Blocks*. This allows you to create scripts and blocks using the context-sensitive menu from the *Scripts/Blocks* tree and also organize them in folders.

We strongly recommend using the IBExpert User Database as a main storage for IBExpert, even if you do not need the ##Scripts/Blocks" feature.

Since IBExpert version 2006.01.29 it is possible to execute Firebird 2.0 blocks stored in registered databases or in the IBExpert User Database directly from the DB Explorer. Simply use the DB Explorer right-click context menu or open the script in the Block Editor and execute using [F9].

When writing new IBEBlocks, do not foget to save the block by clicking on the disk icon, in order to commit it, before running it. Input parameters can be specified by clicking on the *Parameters* icon (or using [Shift # Ctrl # P]), and the block run in the usual IBExpert way by using [F9] or the green arrow icon.



Please refer to IBEBlock and IBEScripts for further information and examples of these comprehensive features. Similar to the Procedure and Trigger Debugger, the Block Éditor allows you to debug your script or block. It offers the same informational pages: Parameters and Variables, Watches, Last Statement, Breakpoints, Messages, Results and SQL Editor Messages.

Please refer to Debugger for further details.

IBExpert version 2008.08.08 introduced the possibility to sort data on the IBEBlock *Results* page by clicking on a grid column caption. It is now also possible to export this data.

ENUM datatype

The ENUM datatype was implemented in IBExpert version 2007.05.03. Generally this datatype is useful for input parameters when it is necessary to allow users to select a value from a given set of values.

Example

```
execute ibeblock (MonthName enum ('January', 'February', 'March',
                                  'April', 'May', 'June', 'July',
                                  'August', 'September', 'October',
                                  'November', 'December') default = 0)
as
begin
  ...
end;
```

For each input parameter of type ENUM IBExpert will create a combobox with the corresponding set of items. See Copy database object blocks to learn how this works.

Concatenating assignment operator - '.='

A new concatenating assignment operator - '.=' was introduced in IBExpert version 2007.05.03. This appends the argument on the right side to the argument on the left side.

Example

```
            sVal = 'abc';
sVal .= 'def'
```

Now `sVal` is equal to `'abcdef'`;

Procedural extensions of IBEBlock

CREATE CONNECTION

Creates a named connection to a database.

Syntax

```
CREATE CONNECTION connection DBNAME 'filespec'
   USER 'username' PASSWORD 'password'
   [CLIENTLIB 'libfile']
   [NAMES charset]
   [SQL_DIALECT dialect]
   [ROLE rolename]
```

Argument	Description
`connection`	Connection name.
`DBNAME 'filespec'`	Database file name; can include path specification and node.
`USER 'username'`	String that specifies a user name for use when attaching to the database. The server checks the user name against the security database (Server security `ISC4.GDB` / `SECURITY.FDB`). User names are case insensitive on the server.
`PASSWORD 'password'`	String, up to 8 characters in size, that specifies password for use when attaching to the database. The server checks the user name and password against the security database. Case sensitivity is retained for the comparison.
`CLIENTLIB 'libfile'`	Client library file name; default: `gds32.dll`.
`NAMES charset`	Name of a character set that identifies the active character set for a given connection; default: `NONE`.
`SQL_DIALECT dialect`	The SQL Dialect for database access, either 1, 2, or 3.
`ROLE rolename`	String, up to 31 characters in size, which specifies the role that the user adopts on connection to the database. The user must have previously been granted membership in the role to gain the privileges of that role. Regardless of role memberships granted, the user has the privileges of a role at connect time only if a `ROLE` clause is specified in the connection. The user cannot adopt more than one role per connection, and cannot switch roles except by reconnecting.

Example

```
execute IBEBlock
as
begin
  CREATE CONNECTION Con1 DBNAME 'localhost:c:\mydata\mydb.gdb'
  USER 'SYSDBA' PASSWORD 'masterkey'
  CLIENTLIB 'C:\Program Files\Firebird\Bin\fbclient.dll'
  SQL_DIALECT 3 NAMES WIN1251 ROLE ADMIN;

  USE Con1;

  ...

  CLOSE CONNECTION Con1;
end
```

USE connection

Makes an existing connection the active connection.

Syntax

```
USE connection;
```

Argument	Description
connection	Name of an existing connection created with the CREATE CONNECTION statement.

Example

```
execute IBEBlock
as
begin
  CREATE CONNECTION Con1 DBNAME 'localhost:c:\mydata\mydb.gdb'
  USER 'SYSDBA' PASSWORD 'masterkey'
  CLIENTLIB 'C:\Program Files\Firebird\Bin\fbclient.dll'
  SQL_DIALECT 3 NAMES WIN1251 ROLE ADMIN;

  USE Con1;

  ...

  CLOSE CONNECTION Con1;
end
```

CLOSE CONNECTION

Closes an existing connection.

Syntax

```
CLOSE CONNECTION connection;
```

Argument	Description
connection	Name of an existing connection opened with the CREATE CONNECTION statement.

Example

```
execute IBEBlock
as
begin
  CREATE CONNECTION Con1 DBNAME 'localhost:c:\mydata\mydb.gdb'
  USER 'SYSDBA' PASSWORD 'masterkey'
  SQL_DIALECT 3 NAMES WIN1251;

  USE Con1;

  ...

  CLOSE CONNECTION Con1;
end
```

See Also:
Joining tables from different databases

CREATE DATABASE

Syntax

```
CREATE DATABASE 'filespec' USER 'username' PASSWORD 'password'
   [CLIENTLIB 'libfile']
   [SQL_DIALECT dialect]
   [PAGE_SIZE int]
   [DEFAULT CHARACTER SET charset]
```

Argument	Description
`'filespec'`	A new database file specification; file naming conventions are platform-specific.
`USER 'username'`	Checks the username against valid user name and password combinations in the security database (Server security `ISC4.GDB` / `SECURITY.FDB`) on the server where the database will reside.
`PASSWORD 'password'`	Checks the password against valid user name and password combinations in the security database on the server where the database will reside; can be up to 8 characters.
`CLIENTLIB 'libfile'`	Client library file name; default: gds32.dll.
`SQL_DIALECT dialect`	The SQL Dialect for the new database, either 1, 2, or 3.
`PAGE_SIZE int`	Size, in bytes, for database pages; int can be 1024 (default), 2048, 4096, or 8192.
`DEFAULT CHARACTER SET charset`	Sets default character set for a database; charset is the name of a character set; if omitted, character set defaults to `NONE`.

Example

```
execute IBEBlock
as
begin
  CREATE DATABASE 'localhost:c:\db2.fdb'
  USER 'SYSDBA' PASSWORD 'masterkey'
  PAGE_SIZE 4096 SQL_DIALECT 3
  DEFAULT CHARACTER SET WIN1251
  CLIENTLIB 'C:\Program Files\Firebird\bin\fbclient.dll';

  CREATE CONNECTION Con1 'localhost:c:\db2.fdb'
  USER 'SYSDBA' PASSWORD 'masterkey'
  CLIENTLIB 'C:\Program Files\Firebird\Bin\fbclient.dll'
  SQL_DIALECT 3 NAMES WIN1251;

  USE Con1;

  ...

  CLOSE CONNECTION Con1;
end
```

See also:
Create Database

DROP DATABASE

Deletes specified database.

Syntax

```
DROP DATABASE 'filespec' USER 'username' PASSWORD 'password'
   [CLIENTLIB 'libfile'];
```

Argument	Description
`'filespec'`	A database file specification; file naming conventions are platform-specific.
`USER 'username'`	Checks the username against valid user name and password combinations in the security database (Server security `ISC4.GDB` / `SECURITY.FDB`) on the server where the database will reside.
`PASSWORD 'password'`	Checks the password against valid user name and password combinations in the security database on the server where the database will reside; can be up to 8 characters.
`CLIENTLIB 'libfile'`	Client library file name; default: `gds32.dll`.

Description

`DROP DATABASE` deletes specified database, including any associated secondary, shadow, and log files. Dropping a database deletes any data it contains.

A database can be dropped by its creator, the SYSDBA user, and any users with operating system root privileges.

Example

```
execute ibeblock
as
begin
  drop database 'localhost/3060:c:\db1.fdb' user 'SYSDBA' password 'masterkey'
  clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';
end
```

FOR ... DO loops

FOR ... DO loops were implemented in IBExpert version 2005.03.12.

Examples

```
EXECUTE IBEBLOCK
RETURNS (I INTEGER)
AS
BEGIN
  FOR I = 0 TO 100 DO
    SUSPEND;
END
```

It is possible to use the CONTINUE statement within FOR loop to proceed to the next iteration of FOR:

```
EXECUTE IBEBLOCK
RETURNS (I INTEGER)
AS
BEGIN
  FOR I = 0 TO 100 DO
  BEGIN
    IF (I < 20) THEN
      CONTINUE; -- SUSPEND will not be executed
    SUSPEND;
  END
END
```

FOREACH statement

The FOREACH statement was implemented in IBExpert version 2007.02.22. This statement simply offers a way to iterate arrays. The SKIP NULLS option was added in IBExpert version 2007.05.03.

Syntax

```
FOREACH (var1 AS var2 [KEY | INDEX var3] [SKIP NULLS]) DO
   <statements>
```

FOREACH loops over the array given by var1. On each loop, the value of the current element is assigned to var2. If the KEY (INDEX) var3 clause is specified, the current element's key will be assigned to the variable var3 on each loop.

Example

```
MyVar = ibec_Array('Some text', 23, NULL, 56.32);
foreach (MyVar as val key id) do
  if (val is not null) then
    ibec_ShowMessage('MyVar[' || id || '] value is: ' || val);
```

The code above is equal to following:

```
MyVar = ibec_Array('Some text', 23, NULL, 56.32);
for id = 0 to ibec_High(MyVar) do
begin
  val = MyVar[id];
  if (val is not null) then
    ibec_ShowMessage('MyVar[' || id || '] value is: ' || val);
end
```

This FOREACH statement with the SKIP NULLS option is equal to the following FOREACH statement without the SKIP NULLS option:

```
FOREACH (var1 AS var2 [KEY | INDEX var3]) DO
BEGIN
  IF (var2 IS NULL) THEN
    CONTINUE;
  <statements>     END
```

See also:
ibec_Array

SELECT ... AS DATASET

Syntax

```
<select_statement> AS DATASET dataset;
```

Argument	Description
`<select_statement>`	Regular SELECT statement.
`dataset`	Name of the dataset.

Example

```
execute ibeblock
returns (FieldName varchar(31), FieldType varchar(100))
as
begin
  select * from rdb$fields
  where (1 = 0)
  as dataset RdbFields;

  iCount = ibec_ds_FieldCount(RdbFields);
  i = 0;
  while (i < iCount) do
  begin
    FieldName = ibec_ds_FieldName(RdbFields, i);
    FieldType = ibec_ds_FieldTypeN(RdbFields, i);
    suspend;
    i = i + 1;
  end;

  close dataset RdbFields;
end
```

See also:
Dataset Functions
Recreating indices 2
Using SELECT statements

EXPORT AS ... INTO

`SELECT ... EXPORT AS ...` was implemented in IBExpert version 2005.03.12.

Examples of usage

1.
```
SELECT * FROM RDB$FIELDS
EXPORT AS HTML INTO 'E:\TestExport.html'
OPTIONS 'ColorShema=MSMoney; FontFace=Verdana';
```

Possible `ColorShemes` are `BW`, `Classic`, `ColorFull`, `Gray`, `MSMoney`, `Murky`, `Olive`, `Plain`, `Simple`.

2.
```
SELECT * FROM RDB$FIELDS
EXPORT AS XLS INTO 'E:\TestExport.xls'
OPTIONS '';
```

3.
```
SELECT * FROM RDB$FIELDS
EXPORT AS TXT INTO 'E:\TestExport.txt'
OPTIONS 'OmitCaptions';
```

4.
```
SELECT * FROM RDB$FIELDS
EXPORT AS CSV INTO 'E:\TestExport.txt'
OPTIONS 'OmitCaptions; Delimiter=";"';
```

5.
```
SELECT * FROM RDB$FIELDS
EXPORT AS XML INTO 'E:\TestExport.xml'
OPTIONS 'Encoding=windows-1251; MemoAsText; StringAsText';
```

New to IBExpert version 2005.12.04:
6.
```
SELECT * FROM RDB$FIELDS
EXPORT AS DBF INTO 'E:\TestExport.dbf'
OPTIONS 'ConvertToDOS; LongStringsToMemo; DateTimeAsDate';
```

See also:
Example: Export data into DBF

CLOSE DATASET

Closes an existing dataset.

Syntax

```
CLOSE DATASET dataset;
```

Argument	Description
dataset	Name of an existing dataset created with SELECT ... AS DATASET statement.

Example

```
execute ibeblock
returns (FieldName varchar(31), FieldType varchar(100))
as
begin
  select * from rdb$fields
  where (1 = 0)
  as dataset RdbFields;

  iCount = ibec_ds_FieldCount(RdbFields);
  i = 0;
  while (i < iCount) do
  begin
    FieldName = ibec_ds_FieldName(RdbFields, i);
    FieldType = ibec_ds_FieldTypeN(RdbFields, i);
    suspend;
    i = i + 1;
  end;

  close dataset RdbFields;
end
```

See also:
Recreating indices 2
SELECT ... AS DATASET

EXECUTE IBEBLOCK

The `EXECUTE IBEBLOCK` statement was implemented in IBExpert verison 2005.03.12. Using this statement you can call other IBEBlocks from the main block.

Examples of usage

1.
```
 EXECUTE IBEBLOCK
 AS
 BEGIN
   ...
   MyFunc = 'EXECUTE IBEBLOCK (
               IntVal INTEGER)
             RETURNS (
               Square INTEGER)
             AS
             BEGIN
               Square = IntVal * IntVal;
             END';
   EXECUTE IBEBLOCK MyFunc (2) RETURNING_VALUES :Square;
   ...
 END
```

2.
```
 EXECUTE IBEBLOCK
 AS
 BEGIN
  ...
   MyFunc = ibec_LoadFromFile('C:\MyBlocks\Square.ibeblock');
   EXECUTE IBEBLOCK MyFunc (2) RETURNING_VALUES :Square;
   ...
 END
```

EXECUTE STATEMENT

Executes specified SQL statement.

Syntax

```
EXECUTE STATEMENT 'statement'
   [INTO :var [, :var ...]]
   [VALUES :var];
```

Argument	Description
`'statement'`	Any valid DML or DDL statement except `CREATE/DROP DATABASE`. DML statements may contain parameters.
`INTO :var [, :var ...]`	Specifies a list of variables into which to retrieve values. Only singleton `SELECT` operators may be executed with this form of `EXECUTE STATEMENT`.
`VALUES :var`	Array of variants which values will be used to fill parameters if any exist in the statement.

Example

```
execute ibeblock
returns (TableName varchar(31))
as
begin
  TableID = 0;
  Stmt = 'select rdb$relation_name from rdb$relations where rdb$relation_id = :rel_id';
  while (TableID < 35) do
  begin
    execute statement :Stmt into :TableName values :TableId;
    suspend;
    TableID = TableID + 1;
  end
end
```

See also:
`EXECUTE STATEMENT ... AS DATASET`
Table Data Comparing
`FOR EXECUTE STATEMENT ... DO`?

INSERT INTO connection.table

Syntax

```
INSERT INTO connection.table [(col [, col ...])]
     {VALUES (<val> [, <val> ...]) | <select_expr>};
```

See also:
Example: Inserting files into a database

COMMIT

Makes a transaction's changes to the database permanent, and ends the transaction.

Syntax

```
COMMIT;
```

Example

```
execute IBEBlock
as
begin
  ...

  EXECUTE STATEMENT 'create table mytable (id integer, data varchar(50))';
  COMMIT;

  INSERT INTO MYTABLE (ID, DATA) VALUES (1, NULL);
  COMMIT;

  ...
end
```

See also:
ROLLBACK

ROLLBACK

Restores the database to its state prior to the start of the current transaction.

Syntax

```
ROLLBACK;
```

Description

ROLLBACK undoes changes made to a database by the current transaction, then ends the transaction.

See also:
COMMIT

EXECUTE STATEMENT ... AS DATASET

Implemented in IBExpert version 2006.08.12.

FOR EXECUTE STATEMENT ... DO

Example

```
execute ibeblock
returns (TableName varchar(31))
as
begin
  TableID = 0;
  Stmt = 'select rdb$relation_name from rdb$relations where rdb$relation_id = :rel_id';
  while (TableID < 35) do
  begin
    execute statement :Stmt into :TableName values :TableId;
    suspend;
    TableID = TableID + 1;
  end
end
```

TRY ... FINALLY

Syntax

```
TRY
  statementList1
FINALLY
  statementList2
END
```

where each statementList is a sequence of statements delimited by semicolons.

Description

The `TRY...FINALLY` statement executes the statements in `statementList1` (the `TRY` clause). If `statementList1` finishes without raising any exceptions, `statementList2` (the `FINALLY` clause) is executed. If an exception is raised during execution of `statementList1`, control is transferred to `statementList2`; once `statementList2` finishes executing, the exception is re-raised.

If a call to the `Exit` procedure causes the control to leave `statementList1`, `statementList2` is automatically executed. Thus the `FINALLY` clause is always executed, regardless of how the `TRY` clause terminates.

Example

```
execute ibeblock
as
begin
  i = 1;
  try
    i = i/0; <-- Here an will be exception raised...
  finally
    i = 2;   <-- ... but this statement will be executed anyway
  end
  i = 3;     <-- This statement will not be executed
end
```

See also:
EXCEPTION
TRY ... EXCEPT
Exception-handling Functions

TRY ... EXCEPT

Syntax

```
TRY
  statements
EXCEPT
  exceptionBlock
END
```

where `statements` is a sequence of statements (delimited by semicolons) and `exceptionBlock` is another sequence of statements.

Description

A `TRY...EXCEPT` statement executes the statements in the initial statements list. If no exceptions are raised, the exception block (`exceptionBlock`) is ignored and the control passes on to the next part of the IBEBlock.

If an exception is raised during execution of the initial statements list, the control passes to the first statement in the `exceptionBlock`. Here you can handle any exceptions which may occur using the following functions:

- `function ibec_err_Message()` - returns an exception message.
- `function ibec_err_SQLCode()` - returns the SQLCode of an exception if there was an SQL error.
- `function ibec_err_Name()` - returns an exception name (for exceptions raised with `EXCEPTION` statement; see below).

You can also re-raise an exception using the `RAISE` statement.

Example

```
execute ibeblock
as
begin
  try
    -- Attempt to insert into non-existent table
    insert into missing_table (f1) values (1);
    ibec_ShowMessage('There were no errors...');
  except
    ErrSQLCode = ibec_err_SQLCode();
    if (ErrSQLCode = -204) then
      ibec_ShowMessage(ibec_err_Message());
    else
      raise;
  end
end
```

See also:
`TRY ... FINALLY` `EXCEPTION`
Exception-handling Functions

EXCEPTION

The `EXCEPTION` statement is similar to Firebird dynamic exceptions.

Syntax

```
EXCEPTION <exception_name> [<exception_text>]
```

`<exception_name>` is the name of an exception which may be tested using the `ibec_err_Name` function.

Example

```
execute ibeblock (divisor double precision)
as
begin
  i = 1;
  try
    if ((divisor is null) or (divisor = 0)) then
      exception INVALID_DIVISOR 'The divisor is invalid: NULL or 0';
    i = i/divisor;
  except
    if (ibec_err_name() = 'INVALID_DIVISOR') then
      i = 0;
    else
      raise;
  end
end
```

See also:
`TRY ... FINALLY`
`TRY ... EXCEPT`
Exception-handling Functions

Default values and comments

Default values and comments for input/output parameters and variables were implemented in IBExpert version 2005.03.12.

Example

```
EXECUTE IBEBLOCK (
  CodeDir VARCHAR(1000) = 'C:\MyBlocks\' COMMENT 'Path to my IBEBlocks',
  SQLDialect INTEGER = 3 COMMENT 'Database SQL Dialect')
RETURNS (
  TotalTime DOUBLE PRECISION = 0 COMMENT 'Total time spent')
AS
DECLARE VARIABLE MyVar INTEGER = 0 COMMENT 'Just a comment'
BEGIN
 ...
END
```

- Comments for input parameters will be displayed in `Description` column of the `Request Input Parameters` form.
- Comments for output variables will be used as column captions of the result dataset.
- Comments for local variables are ignored.

IBEBlock Functions

For further functions not included in this section, please refer to User-Defined Functions and the Firebird documentation: Firebird 2 Cheat Sheet: Firebird built-in Functions.

- String-handling functions
- Mathematical functions
- File functions
- Database functions
- Dataset functions
- Managing Firebird and InterBase users
- Date and Time functions
- Windows Registry functions
- Functions to handle regular expressions
- Miscellaneous functions

String-handling functions

The following string-handling functions are available in IBEBlock:

Function	Description
`ibec_Copy`	Returns a substring of a string.
`ibec_Length`	Returns the number of characters in a string.
`ibec_Pos`	Returns the index value of the first character in a specified substring that occurs in a given string.
`ibec_Trim`	Trims leading and trailing spaces and control characters from a string.
`ibec_Format`	Returns a formatted string assembled from a format string and a list of arguments.
`ibec_InputQuery`	Displays an input dialog that enables the user to enter a string.
`ibec_Explode`	Returns an array of strings.

ibec_Copy

Returns a substring of a string.

Syntax

```
function ibec_Copy(S : string; Index, Count: Integer): string;
```

Description

`S` is an expression of a string. `Index` and `Count` are integer-type expressions. `ibec_Copy` returns a substring containing `Count` characters starting at `S[Index]`. If `Index` is larger than the length of `S`, `ibec_Copy` returns an empty string.

If `Count` specifies more characters than are available, only the characters from `S[Index]` to the end of `S` are returned.

Example

```
execute IBEBlock
returns (proc_name varchar(31), proc_src varchar(100))
as
begin
  for
    select rdb$procedure_name, rdb$procedure_source
    from rdb$procedures
    order by rdb$procedure_name
    into :proc_name, :proc_src
  do
  begin
    proc_src = ibec_Copy(proc_src, 1, 100);
    suspend;
  end
end
```

See also:
`ibec_Length`
`ibec_Pos`

ibec_Length

Returns the number of characters in a string.

Syntax

```
function ibec_Length(S : string): string;
```

Description

No additional description...

Example

```
execute IBEBlock
returns (iresult integer)
as
begin
  for select rdb$relation_name
      from rdb$relations
      into :sname
  do
  begin
    sname = ibec_Trim(sname);
    iresult = ibec_Length(sname);
    suspend;
  end
end
```

See also:
ibec_Copy
ibec_Pos

ibec_Pos

Returns the index value of the first character in a specified substring that occurs in a given string.

Syntax

```
function ibec_Pos(Substr: string; S : string): integer;
```

Description

No additional description...

Example

```
execute IBEBlock
returns (vcresult varchar(100))
as
begin
  for select rdb$relation_name
      from rdb$relations
      into :sname
  do
  begin
    sname = ibec_trim(sname);
    vcresult = '';
    if (ibec_Pos('RDB$', sname) = 1) then
      vcresult = sname || ' is a system table';
    else if (ibec_Pos('IBE$', sname) = 1) then
      vcresult = sname || ' is an IBExpert table';
    else
      vcresult = sname || ' is an user table';
    suspend;
  end
end
```

See also:
ibec_Copy
ibec_Length

ibec_Trim

Trims leading and trailing spaces and control characters from a string.

Syntax

```
function ibec_Trim(S : string): string;
```

Description

No additional description...

Example

```
execute IBEBlock
returns (proc_name varchar(31), proc_src varchar(100))
as
begin
  for
    select rdb$procedure_name, rdb$procedure_source
    from rdb$procedures
    order by rdb$procedure_name
    into :proc_name, :proc_src
  do
  begin
    proc_src = ibec_Trim(ibec_Copy(proc_src, 1, 100));
    suspend;
  end
end
```

ibec_Format

This function returns a formatted string assembled from a format string and a list of arguments.

Syntax

```
function ibec_Format(AFormat: string; Arg1 : variant; ...; ArgN : variant): string;
```

Description

`ibec_Format` function formats the series of arguments `Arg1...ArgN`. Formatting is controlled by the format string `AFormat`; the results are returned in the function result as a string.

Example

```
execute ibeblock
as
begin
  ...
  NumOfFiles = 10;
  Mes = ibec_Format('%d files were deleted', NumOfFiles);
  ibec_ShowMessage(Mes);
end

execute ibeblock
as
begin
  ...
  Mes = ibec_Format('There are now s', 1000, 'MYTABLE');
  ibec_ShowMessage(Mes);
end
```

ibec_InputQuery

The `ibec_InputQuery` function was implemented in IBExpert version 2006.12.11. This function displays an input dialog that enables the user to enter a string.

Syntax

```
function ibec_InputQuery(const ACaption, APrompt: string; var Value: string): Boolean;
```

Description

Call `ibec_InputQuery` to bring up an input dialog box ready for the user to enter a string in its edit box. The `ACaption` parameter is the caption of the dialog box, the `APrompt` parameter is the text that prompts the user to enter input in the edit box, and the `Value` parameter is the string that appears in the edit box when the dialog box first appears.

If the user enters a string in the edit box and selects `OK`, the `Value` parameter changes to the new value. `InputQuery` returns `True` if the user selects `OK`, and `False` if the user selects `Cancel` or presses the [Esc] key.

Example

```
execute ibeblock
as
begin
  ...
  Caption = '
  Mes = ibec_Format('There are now s', 1000, 'MYTABLE');
  ibec_ShowMessage(Mes);
end
```

ibec_Explode

`ibec_Explode` returns an array of strings.

Syntax

```
function ibec_Explode(Delimiter : string; Str : string) : array of string;
```

Description

`ibec_Explode` returns an array of strings, each of which is a substring of `Str` formed by splitting it on boundaries formed by the string `Delimiter`.

Example

```
execute ibeblock
as
begin
   Str = 'just a test';
   Delimiter = ' ';
   Words = ibec_Explode(Delimiter, Str);
end;
```

Mathematical functions

The following mathematical functions are available in IBEBlock:

Function	Description
`ibec_Div`	Returns the value of `x/y` rounded in the direction of zero to the nearest integer.
`ibec_Mod`	Returns the remainder obtained by dividing its operands.
`ibec_Power`	Raises the base to any power.

ibec_Div

The value of `x div y` is the value of `x/y` rounded in the direction of zero to the nearest integer.

Syntax

```
function ibec_div(Operand1, Operand2 : integer) : integer;
```

Description

No additional description...

Example

```
execute IBEBlock
returns (cout varchar(100))
as
begin
  i = 1;
  while (I < 50) do
  begin
    if ((i/2 - ibec_div(i, 2)) > 0) then
      cout = i || ' is odd number';
    else
      cout = i || ' is even number';
    suspend;
    i = i + 1;
  end
end
```

ibec_Mod

Returns the remainder obtained by dividing its operands.

Syntax

```
function ibec_mod(Operand1, Operand2 : integer) : integer;
```

Description

No additional decription...

Example

```
execute IBEBlock
returns (cout varchar(100))
as
begin
  i = 1;
  while (I < 50) do
  begin
    if (ibec_mod(i, 2) = 0) then
      cout = i || ' is even number';
    else
      cout = i || ' is odd number';
    suspend;
    i = i + 1;
  end
end
```

See also:
Data Comparer using cursors

ibec_Power

`ibec_Power` raises `Base` to any power.

Syntax

```
function ibec_Power(Base, Exponent : double precision) :
   double precision;
```

Description

For fractional exponents `Base` must be greater than 0.

The `ibec_Power` returns `NULL` if it is impossible to raise `Base` to specified power (for example, `ibec_Power(-4, 0.5)` will return `NULL`).

File functions

The following file-handling functions are available in IBEBlock:

Function	Description
ibec_DeleteFile	Erases the file from the disk.
ibec_FileExists	Tests if a specified file exists.
ibec_FileSize	Returns the size of the specified file.
ibec_GetFiles	Retrieves specified file or list of files.
ibec_LoadFromFile	Loads file data into variable.
ibec_SaveToFile	Saves value of variable into file.
ibec_CopyFile	Copies an existing file to a new one.
ibec_FileDateTime	Returns the TIMESTAMP of a specified file.

The following functions are intended for working with files in stream mode:

Function	Description
ibec_fs_CloseFile	Closes the file opened with the ibec_fs_OpenFile function.
ibec_fs_Eof	Tests whether the file position is at the end of a file.
ibec_fs_OpenFile	Opens a file for reading or writing.
ibec_fs_Position	Returns the current offset into the stream for reading and writing.
ibec_fs_Readln	Reads a line of text from a file.
ibec_fs_ReadString	Reads count bytes from the file stream.
ibec_ini_SetStrings	Sets the contents of the INI file from a variable.
ibec_ini_GetStrings	Saves the contents of the INI file to a variable.
ibec_fs_Seek	Resets the current position of the file stream.
ibec_fs_Size	Returns the length, in bytes, of the file stream.
ibec_fs_Writeln	
ibec_fs_WriteString	

The following functions were introduced to handle work with INI files:

Function	Description
ibec_ini_Open	Instantiates an INI file object.
ibec_ini_Close	Frees the memory associated with the INI file object.
ibec_ini_Clear	Erases all data from the INI file in the memory.
ibec_ini_UpdateFile	Flushes buffered INI file data to disk.
ibec_ini_EraseSection	Erases an entire section of an INI file.
ibec_ini_ReadString	Retrieves a string value from an INI file.
ibec_ini_WriteString	Writes a string value to an INI file.

Please note that all ibec_ini_xxx functions, except ibec_ini_ReadString and ibec_ini_Open, return NULL.

ibec_DeleteFile

Erases the file from the disk.

Syntax

```
function ibec_DeleteFile(FileName : string): boolean;
```

Description

The `ibec_DeleteFile` function erases the file named by `FileName` from the disk. If the file cannot be deleted or does not exist, the function returns `False`.

Example

```
execute IBEBlock
as
begin
  FileName = 'C:\mydata.txt';
  if (ibec_FileExists(FileName)) then
    ibec_DeleteFile(FileName);
end
```

ibec_FileExists

Tests if a specified file exists.

Syntax

```
function ibec_FileExists(FileName : string): boolean;
```

Description

`ibec_FileExists` returns `True` if the file specified by `FileName` exists. If the file does not exist, the function returns `False`.

Example

```
execute IBEBlock
as
begin
  FileName = 'C:\mydata.txt';
  if (ibec_FileExists(FileName)) then
    ibec_DeleteFile(FileName);
end
```

See also:
Data Comparer using cursors

ibec_FileSize

Returns the size of the specified file.

Syntax

```
function ibec_FileSize(FileName : string): variant;
```

Description

The `ibec_FileSize` function returns the size in bytes of the file specified by `FileName`. If the file does not exist, the function returns `NULL`.

Example

```
execute ibeblock
returns (fname varchar(100), isize integer)
as
begin
  options = __gfFullName;
  files_count = ibec_getfiles(files_list, 'E:\Projects_5\', '*.*', options);
  if (files_count > 0) then
  begin
    i = 0;
    while (i < ibec_high(files_list)) do
    begin
      fname = files_list[i];
      isize = ibec_filesize(fname);
      suspend;
      i = i + 1;
    end
  end
end
```

See also:
Example: Importing data from a file
Inserting file data into a database

ibec_GetFiles

Retrieves specified file or list of files.

Syntax

```
ibec_getfiles(files_list, 'path', 'file_name', _gfXXX + __gfXXX);
```

There are three `__gfXXX` constants:

`__gfRecursiveSearch`	The search will be performed recursively for each directory. For example, if `D:\` is specified as the initial path for the search, the function will search also in `D:\MyData`, in `D:\MyPhotos`, in `D:\MyPhotos\Last` etc. In this case the entire `D:` drive will be scanned.
`__gfFullName`	The file names in the result list will include the full path, otherwise only the file name (without the drive letter and directories) will be listed.
`__gfAppend`	This is useful when you perform several searches one by one with different conditions. If this option is specified the function will NOT clear the result list before performing a new search, new results will be added to the `files_list`. Otherwise the `files_list` variable will be erased before searching.

See also:
Inserting file data into a database

ibec_LoadFromFile

Loads file data into variable.

Syntax

```
function ibec_LoadFromFile(FileName : string): string;
```

Example

See Inserting file data into a database.

See also:
`ibec_SaveToFile`
Example: Importing data from a file

ibec_SaveToFile

Saves value of variable into file.

Syntax

```
function ibec_SaveToFile(FileName : string; Value : variant; Mode : integer): variant;
```

See also:
`ibec_LoadFromFile`

ibec_CopyFile

Copies an existing file to a new file.

Syntax

```
ibec_CopyFile(ExistingFileName, NewFileName : string;
                          FailIfExists : boolean) : boolean;
```

Description

The `ibec_CopyFile` function copies an existing file to a new file. If the `FailIfExists` parameter is `True` and the new file already exists, the function fails. If this parameter is `False` and the new file already exists, the function overwrites the existing file.

ibec_FileDateTime

Returns the `TIMESTAMP` of a specified file.

Syntax

```
function ibec_FileDateTime(FileName : string) : variant;
```

Returns the `TIMESTAMP` of a specified file. If the file doesn't exist `ibec_FileDateTime` returns `NULL`.

ibec_fs_CloseFile

Closes the file opened with the `ibec_fs_OpenFile` function.

Syntax

```
function ibec_fs_CloseFile(FileHandle : variant): variant
```

Description

The `ibec_fs_CloseFile` function closes the file opened with the `ibec_fs_OpenFile` function. This function always returns `0`.

Example

```
execute  IBEBlock
as
begin
  FileName = 'C:\mydata.txt';
  FH = ibec_fs_OpenFile(FileName, __fmCreate);
  if (not FH is NULL) then
  begin
    ibec_fs_Writeln(FH, 'just a test');
    ibec_fs_CloseFile(FH);
  end
end
```

ibec_fs_Eof

Tests whether the file position is at the end of a file.

Syntax

```
function ibec_fs_Eof(FileHandle : variant): boolean;
```

Description

The `ibec_fs_Eof` function tests whether the file position is at the end of a file. `ibec_fs_Eof` returns `True` if the current file position is beyond the last character of the file or if the file is empty; otherwise, `ibec_fs_Eof` returns `False`.

Example

```
execute IBEBlock
returns (vcout varchar(1000))
as
begin
  FileName = 'C:\mydata.csv';
  FH = ibec_fs_OpenFile(FileName, __fmOpenRead);
  if (not FH is NULL) then
  begin
    while (not ibec_fs_Eof(FH)) do
    begin
      vcout = ibec_fs_Readln(FH);
      suspend;
    end
    ibec_fs_CloseFile(FH);
  end
end
```

See also:
Example: Importing data from a CSV file

ibec_fs_OpenFile

Opens a file for reading or writing.

Syntax

```
function ibec_fs_OpenFile(FileName : string; Mode : integer): variant;
```

Description

The `ibec_fs_OpenFile` function opens file specified by `FileName` for reading or writing.

The `Mode` parameter indicates how the file is to be opened. The `Mode` parameter consists of an `open mode` and a `share mode` stored together. The `open mode` must be one of the following values:

Value	Meaning
`__fmCreate`	Create a file with the given name. If a file with the given name exists, open the file in write mode.
`__fmOpenRead`	Open the file for reading only.
`__fmOpenWrite`	Open the file for writing only. Writing to the file completely replaces the current contents.
`__fmOpenReadWrite`	Open the file to modify the current contents rather than replace them.

The `share mode` must be one of the following values:

Value	Meaning
`__fmShareCompat`	Sharing is compatible with the way FCBs are opened.
`__fmShareExclusive`	Other applications can not open the file for any reason.
`__fmShareDenyWrite`	Other applications can open the file for reading but not for writing.
`__fmShareDenyRead`	Other applications can open the file for writing but not for reading.
`__fmShareDenyNone`	No attempt is made to prevent other applications from reading from or writing to the file.

If the file cannot be opened, `ibec_fs_OpenFile` returns `NULL`. Otherwise it returns the handle for the file just opened.

To close the file opened with `ibec_fs_OpenFile` use the [`@ibec_fs_CloseFile@`] function.

Example

```
execute  IBEBlock
as
begin
  FileName = 'C:\mydata.txt';
  FH = ibec_fs_OpenFile(FileName, __fmCreate);
  if (not FH is NULL) then
  begin
    ibec_fs_Writeln(FH, 'just a test');
    ibec_fs_CloseFile(FH);
  end
end
```

See also:
Creating an UPDATE script with domain descriptions
Example: Importing data from a CSV file

ibec_fs_Position

Returns the current offset into the stream for reading and writing.

Syntax

```
function ibec_fs_Position(FileHandle : variant) : integer;
```

Description

Use `ibec_fs_Position` to obtain the current position of the stream. This is the number of bytes from the beginning of the streamed data.

Example

```
execute IBEBlock
returns (vcout varchar(1000))
as
begin
  FileName = 'C:\mydata.csv';
  FH = ibec_fs_OpenFile(FileName, __fmOpenRead);
  if (not FH is NULL) then
  begin
    while (ibec_fs_Position(FH) < ibec_fs_Size(FH)) do
    begin
      vcout = ibec_fs_Readln(FH);
      suspend;
    end
    ibec_fs_CloseFile(FH);
  end
end
```

See also:
`ibec_fs_Seek`

ibec_fs_Readln

Reads a line of text from a file.

Syntax

```
function ibec_fs_Readln(FileHandle : variant) : string;
```

Description

The `ibec_fs_Readln` function reads a line of text and then skips to the next line of the file.

Example

```
execute IBEBlock
returns (vcout varchar(1000))
as
begin
  FileName = 'C:\mydata.csv';
  FH = ibec_fs_OpenFile(FileName, __fmOpenRead);
  if (not FH is NULL) then
  begin
    while (not ibec_fs_Eof(FH)) do
    begin
      vcout = ibec_fs_Readln(FH);
      suspend;
    end
    ibec_fs_CloseFile(FH);
  end
end
```

See also:
`ibec_fs_Writeln`
`ibec_fs_WriteString`
Example: Importing data from a CSV file

ibec_fs_ReadString

Reads `Count` bytes from the file stream created with `ibec_fs_OpenFile`.

Syntax

```
function ibec_fs_ReadString(FileHandle : variant; Count : integer) :
string;
```

Description

Use `ibec_fs_ReadString` to read `Count` bytes from the file stream created with `ibec_fs_OpenFile` into a variable in cases where the number of bytes is known and fixed.

Example

```
execute ibeblock
as
begin
  fs = ibec_fs_OpenFile('C:\MyData.dat', __fmOpenRead);
  if (fs is not null) then
  begin
    ibec_fs_Seek(fs, -100, __soFromEnd);
    MyStr = ibec_fs_ReadString(fs, 100);
    ibec_fs_CloseFile(fs);
  end
end
```

See also:
`ibec_fs_WriteString`

ibec_ini_SetStrings

`ibec_ini_SetStrings` sets the contents of the `INI` file from a variable.

ibec_ini_GetStrings

`ibec_ini_GetStrings` saves the contents of the `INI` file to a variable.

[[#t]

ibec_fs_Seek

Resets the current position of the file stream.

Syntax

```
function ibec_fs_Seek(FileHandle : variant; Offset: integer; Origin: integer): integer;
```

Description

Use `ibec_fs_Seek` to move the current position within the file by the indicated offset. `ibec_fs_Seek` allows you to read from or write to a particular location within the file.

The `Origin` parameter indicates how the `Offset` parameter should be interpreted. `Origin` should be one of the following values:

Value	Meaning
`__soFromBeginning`	`Offset` is from the beginning of the resource. `ibec_fs_Seek` moves to the position `Offset`. `Offset` must be >= 0.
`__soFromCurrent`	`Offset` is from the current position in the resource. `ibec_fs_Seek` moves to `Position + Offset`.
`__soFromEnd`	`Offset` is from the end of the resource. `Offset` must be `<= 0` to indicate a number of bytes before the end of the file.

`ibec_fs_Seek` returns the new current position in the file.

See also:
`ibec_fs_Position` `ibec_fs_Size`

ibec_fs_Size

Returns the length, in bytes, of the file stream.

Syntax

```
function ibec_fs_Size(FileHandle : variant) : integer;
```

Description

The `ibec_fs_Size` returns the length, in bytes, of the file identified by the `FileHandle`.

Example

```
execute IBEBlock
returns (vcout varchar(1000))
as
begin
  FileName = 'C:\mydata.csv';
  FH = ibec_fs_OpenFile(FileName, __fmOpenRead);
  if (not FH is NULL) then
  begin
    while (ibec_fs_Position(FH) < ibec_fs_Size(FH)) do
    begin
      vcout = ibec_fs_Readln(FH);
      suspend;
    end
    ibec_fs_CloseFile(FH);
  end
end
```

See also:
`ibec_fs_Position` `ibec_fs_Seek`

ibec_fs_SetSize

ibec_fs_Writeln

See also:
Example: Importing data from a CSV file

ibec_fs_WriteString

ibec_ini_Open

`ibec_ini_Open` instantiates an `INI` file object.

Syntax

```
function ibec_ini_Open(FileName : string) : variant;
```

The `FileName` is the name of the `INI` file which will be used.

Description

`ibec_ini_Open` loads a copy of the `INI` file into the memory if the specified file exists. `ibec_ini_Open` returns the handle of the `INI` file object if successful, otherwise it returns `NULL`.

Please note: all `ibec_ini_xxx` functions, except `ibec_ini_ReadString` and `ibec_ini_Open`, return `NULL`.

ibec_ini_Close

`ibec_ini_Close` frees the memory associated with the `INI` file object.

Syntax

```
function ibec_ini_Close(IniFile : variant) : variant;
```

Description

No updates are made of the associated file on disk, you must use `ibec_ini_UpdateFile` to flush buffered `INI` file data to disk.

ibec_ini_Clear

Erases all data from the `INI` file in the memory.

Syntax

```
function ibec_ini_Clear(IniFile : variant) : variant;
```

Description

Call `ibec_ini_Clear` to erase all data from the `INI` file that is currently buffered in the memory. All sections, keys, and values are erased. No exception is generated when using `Clear` and the data has not been saved to the `INI` file with the `ibec_ini_UpdateFile` function.

ibec_ini_UpdateFile

`ibec_ini_UpdateFile` flushes buffered `INI` file data to disk.

Syntax

```
function ibec_ini_UpdateFile(IniFile : variant) : variant;
```

Description

Call `ibec_ini_UpdateFile` to copy `INI` file data stored in the memory to the copy of the `INI` file on disk. `ibec_ini_UpdateFile` overwrites all data contained in the disk copy of the `INI` file with the `INI` file data stored in the memory. If the file does not already exist, it is created. If the new file already exists, it is overwritten.

ibec_ini_EraseSection

Erases an entire section of an `INI` file.

Syntax

```
function ibec_ini_EraseSection(IniFile : variant; Section : string) : variant;
```

Description

Call `ibec_ini_EraseSection` to remove a section, all its keys, and their data values from an `INI` file. `Section` identifies the `INI` file section to remove. If a section cannot be removed, an exception is raised. `ibec_ini_EraseSection` only affects the in-memory copy of the `INI` file, not the copy on disk.

ibec_ini_ReadString

Retrieves a string value from an `INI` file.

Syntax

```
function ibec_ini_ReadString(IniFile : variant; Section, Ident, Default : string) : string;
```

Call `ibec_ini_ReadString` to read a string value from an `INI` file.

Parameters

`Section`	identifies the section in the file that contains the desired key.
`Ident`	is the name of the key from which to retrieve the value.
`Default`	is the string value to return if the `Section` does not exist or the key doesn't exist or the data value for the key is not assigned.

ibec_ini_WriteString

Writes a string value to an `INI` file.

Syntax

```
function ibec_ini_WriteString(IniFile : variant; Section, Ident, Value : string) : variant;
```

Description

Call `ibec_ini_WriteString` to write a string value to an `INI` file.

Parameters

`Section`	identifies the section in the file that contains the key to which to write to.
`Ident`	is the name of the key for which to set a value.
`Value`	is the string value to write.

Please note that attempting to write a data value to a non-existent section or attempting to write data to a non-existent key are not errors. In these cases, `ibec_ini_WriteString` creates the section and key and sets its initial value to `Value`.

Database functions

The following database-handling functions are available in IBEBlock:

Function	Description
ibec_CreateConnection	Creates an active database connection.
ibec_UseConnection	Uses an active database connection.
ibec_CloseConnection	Closes an active database connection.
ibec_RecompileTrigger	Recompiles triggers.
ibec_RecompileProcedure	Recompiles stored procedures.
ibec_CompareTables	Compares the data of specified tables and creates a script of all discrepancies.
ibec_CompareMetadata	Compares the metadata of specified databases and creates a script of all discrepancies.
ibec_ExtractMetadata	Extracts metadata (and data if specified) of a database into a script.
Specifying WHERE clauses in ibec_ExtractMetadata	Allows specification of WHERE clauses for each data table.
ibec_BackupDatabase	Starts the backup process using the server Services Manager.
ibec_RestoreDatabase	Starts the restore process using the server Services Manager.
ibec_GetConnectionProp	Returns the server version of the active connection.
ibec_GetCurrentDir	Returns the fully qualified name of the current directory.
ibec_GetRunDir	Returns the path of the currently executing program (IBExpert.exe of IBEScript.exe.
ibec_GetUserDBConnection	Returns the pointer to the User Database if it is used.
ibec_ibe_GetActiveDatabaseID	Returns the unique identifier of the active (currently used) database within IBExpert.
ibec_ibe_GetDatabaseProp	Returns value of specifed database property.

ibec_CreateConnection

The ibec_CreateConnection creates an active database connection.

See also:
Example: ODBC Access

ibec_UseConnection

See also:
Example: ODBC Access

ibec_CloseConnection

See also:
Example: ODBC Access

ibec_RecompileTrigger

Recompiles triggers.

Syntax

```
function ibec_RecompileTrigger(Connection : variant; TriggerName : string) : string;
```

Description

This function recompiles (alters using current trigger source) a specified trigger and returns an empty string if no error occurs or an error message otherwise. Instead of a trigger name you can specify an empty string to recompile ALL database triggers.

Examples of usage

1. Recompile a single trigger using the current connection:

```
execute ibeblock
returns (ErrMessage varchar(1000))
as
begin
   db = ibec_GetDefaultConnection();
   ErrMessage = ibec_RecompileTrigger(db, 'MYTABLE_TRG_BI');
   if (ErrMessage <> '') then
     suspend;
end
```

2. Recompile ALL database triggers using the current connection:

```
execute ibeblock
returns (ErrMessage varchar(10000))
as
begin
   ErrMessage = ibec_RecompileTrigger(0, '');
   if (ErrMessage <> '') then
     suspend;
end
```

See also:
Recompile all Stored Procedures and Triggers

ibec_RecompileProcedure

Recompiles stored procedures.

Syntax

```
function ibec_RecompileProcedure(Connection : variant; ProcedureName : string) : string;
```

Description

This function recompiles (alters using current procedure source) a specified stored procedure and returns an empty string if no error occurs or an error message otherwise. Instead of a procedure name you can specify an empty string to recompile ALL database stored procedures.

Examples of usage

1. Recompile a single stored procedure using the current connection:

```
execute ibeblock
returns (ErrMessage varchar(1000))
as
begin
   db = ibec_GetDefaultConnection();
   ErrMessage = ibec_RecompileProcedure(db, 'MY_PROC');
   if (ErrMessage <> '') then
     suspend;
end
```

2. Recompile ALL database procedures using the current connection:

```
execute ibeblock
returns (ErrMessage varchar(10000))
as
begin
   ErrMessage = ibec_RecompileProcedure(0, '');
   if (ErrMessage <> '') then
     suspend;
end
```

See also:
Recompile all Stored Procedures and Triggers

ibec_CompareTables

Compares the data of specified tables and creates a script of all discrepancies.

Syntax

```
function ibec_CompareTables(MasterDB : variant; SubscriberDB : variant;
   MasterTable : string; SubscriberTable :string;
   ScriptFile : string; Options : string;
   CallbackProc : variant) : variant;
```

Description

This function compares the data of two tables and creates a discrepancy script. Both tables must have a primary key.

Since IBExpert version 2006.08.12 it is possible to include millseconds into time/timestamp values when comparing table data. Use the `IncludeMilliseconds` or `IncludeMsecs` option for this.

Parameters

`MasterDB`	A handle to the reference database, maybe `0` or `NULL` if the current connection is used as a reference connection.
`SubscriberDB`	A handle to the comparative database, maybe `0` or `NULL` if the current connection is used as a comparative connection.
`MasterTable`, `SubscriberTable`	Names of the reference and comparative tables.
`ScriptFile`	Name of the script file which will contain the discrepancy script.
`Options`	List of options, delimited with a semicolon; possible options are:
`OmitDeletes`	Missing records will not be checked by the data comparison. You can also use `ProcessDeletes=0`.
`OmitInserts`	New records will not be checked by the data comparison. You can also use `ProcessInserts=0`.
`OmitUpdates`	Modified records will not be checked by the data comparison. You can also use `ProcessDeletes=0`.
`UpdateAllColumns`	If this option is specified `UPDATE` statements will include non-modifed columns too.
`AppendMode`	If this option is specified and the file `ScriptFile` already exists the resulting script will be appended to the `ScriptFile`. Otherwise a new file will be created.
`CallbackProc`	A callback IBEBlock which will be executed for each record processed whilst comparing data. The callback IBEBlock must have at least one input parameter, which will be used to pass a number of processed records within it.

IBExpert version 2008.08.08 introduced the ability to compare more than one table in a single operation. Simply specify the list of necessary tables, delimited with a comma or semicolon, as `MasterTable` and `SubscriberTable`. For example:

```
ibec_CompareTables@@(DB1, DB2,'TABLE1, TABLE2, "Table3"',
                     'TABLE1, TABLE2, "Table3"',
                     'D:\Diff.sql', 'UpdateOrInsert', cbb);'
```

The `UpdateOrInsert` option (and `UseUpdateOrInsert`) is now also valid. This allows you to generate `UPDATE OR INSERT` statements instead of `UPDATE/INSERT` for Firebird 2.1 databases (see example above).

Example of usage

```
execute ibeblock
returns (
  TotalTime double precision = 0 comment 'Time spent (seconds)')
as
begin
  create connection MasterDB dbname 'localhost:c:\MasterDB.fdb'
  password 'masterkey' user 'SYSDBA'
  clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

  create connection SubscriberDB dbname 'localhost:c:\SubscriberDB.fdb'
  password 'masterkey' user 'SYSDBA'
  sql_dialect 3
  clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

  cbb = 'execute ibeblock (
      RecsProcessed variant)
    as
    begin
      if (ibec_mod(RecsProcessed, 100) = 0) then
      ibec_progress(Records compared:  || RecsProcessed);
    end';

ibec_CompareTables(MasterDB, SubscriberDB, 'IBE$$TEST_DATA', 'IBE$$TEST_DATA',

'E:\CompRes.sql','OmitUpdates', cbb);
ibec_CompareTables(MasterDB, SubscriberDB, 'IBE$$TEST_DATA', 'IBE$$TEST_DATA',

'E:\CompRes.sql','AppendMode; OmitDeletes; OmitInserts; UpdateAllColumns', cbb);

  close connection MasterDB;
  close connection SubscriberDB;
```

```
  EndTime = ibec_gettickcount();
  TotalTime =  (EndTime - StartTime) / 1000;
  suspend;
end
```

See also:
Table Data Comparer
Table Data Comparing

ibec_CompareMetadata

Compares the metadata of specified databases and creates a script of all discrepancies.

Syntax

```
function ibec_CompareMetadata(MasterDB : variant; SubscriberDB :variant;
   ScriptFile : string; Options : string;
   CallbackProc : variant) : string;
```

Description

This function compares the metadata of two databases (or scripts) and creates a discrepancy script.

Parameters

MasterDB	Reference database or script file.
SubscriberDB	Comparative database or script file.
ScriptFile	Name of the difference script file.
Options	List of options, delimited with semicolon; possible options are:
OmitDomains	(Domains=0) don't compare domains.
OmitTables	(Tables=0) don't compare tables.
OmitViews	(Views=0) don't compare views.
OmitTriggers	(Triggers=0) don't compare triggers.
OmitProcedures	(Procedures=0) don't compare procedures.
OmitGenerators	(Generators=0 don't compare generators.
OmitExceptions	(Exceptions=0) don't compare exceptions.
OmitUDFs	(UDFs=0) don't compare UDFs.
OmitRoles	(Roles=0) don't compare roles.
OmitIndices	(Indices=0) don't compare indices.
OmitGrants	(Grants=0) don't compare privileges.
OmitDescriptions	(Descriprions=0) don't compare object descriptions.
OmitPrimaryKeys	(PrimaryKeys=0) don't compare primary keys.
OmitForeignKeys	(ForeignKeys=0) don't compare foreign keys.
OmitUniques	(Uniques=0) don't compare unique constraints.
OmitChecks	(Checks=0) don't compare check constraints.
ServerVersion	New to IBExpert version 2005.12.04. Possible values are: IB4? - for InterBase 4.?, IB5? - for InterBase 5.?, IB6? - for InterBase 6.?, IB7? - for InterBase 7.?, FB1? - for Firebird 1.?, FB15 - for Firebird 1.5, FB2? - for Firebird 2.?, YA1? - for Yaffil 1.?. If the ServerVersion is not specified, FB15 will be used.
CallbackProc	A callback IBEBlock which will be executed for each record processed whilst comparing data. The callback IBEBlock must have at least one input parameter, which will be used to pass a number of processed records within it.

Examples of usage

1. Comparing databases:

```
   execute ibeblock
   as
   begin
      create connection MasterDB dbname 'localhost:c:\MasterDB.fdb'
      password 'masterkey' user 'SYSDBA'
      clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

      create connection SubscriberDB dbname 'localhost:c:\SubscriberDB.fdb'
      password 'masterkey' user 'SYSDBA'
      sql_dialect 3
      clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';


      cbb = 'execute ibeblock (LogMessage variant)
        as
        begin
          ibec_progress(LogMessage);
        end';

 ibec_CompareMetadata(MasterDB, SubscriberDB, 'E:\CompRes.sql', 'OmitDescriptions;
 OmitGrants', cbb);

      close connection MasterDB;
      close connection SubscriberDB;
   end
```

2. Comparing scripts:

```
  execute ibeblock
  as
  begin
     cbb = 'execute ibeblock (
          LogMessage variant)
        as
        begin
          ibec_progress(LogMessage);
        end';

ibec_CompareMetadata('c:\myscripts\master.sql','c:\myscripts\subscriber.sql', 'E:\CompRes.sql','', cbb);
   end
```

3. Using the `ServerVersion` parameter (IBExpert version 2005.12.04):

```
ibec_CompareMetadata(MasterDB,
                     SubscriberDB,
                     'E:\CompRes.sql',
                     'OmitDescriptions; OmitGrants; ServerVersion=FB1?',
                     cbb);
```

See also:
Comparing databases using IBEBlock
Comparing scripts with IBEBlock
Extract metadata using IBEBlock

ibec_ExtractMetadata

Extracts metadata (and data if specified) of a database into a script.

Syntax

```
function ibec_ExtractMetadata(Connection : variant; ScriptFile :string;
Options : string; CallbackProc : variant): string;
```

Description

This function extracts metadata/data of a specified database into a script.

Parameters

Connection	Active database connection.
ScriptFile	Name of the resulting script file or directory name if the VCSFiles option is used.
Options	List of options delimited with semicolon; possible options are:
VCSFiles	Each database object definition will be extracted into a separate file.
SeparateFiles	Extracts metadata (and data if specified) into a set of files: two files with metadata (_ibe$start_.sql and _ibe$finish_.sql), files containing table data (one or more files for each database table) and a runme.sql file, that consists of a number of INPUT <file_name> statements in the correct order.
GenerateCreate	Determines whether a CREATE DATABASE statement should be included at the beginning of the generated script.
GenerateConnect	Determines whether a CONNECT statement should be included at the beginning of the generated script.
IncludePassword	Determines whether the password should be included into the CREATE DATABASE or the CONNECT statement in the resulting SQL script.
SuppressComments	Use to suppress comments in the resulting script.
IncludeCharset	Introduced in IBExpert version 2006.01.29. This option forces IBExpert/IBEScript to include the CHARACTER SET clause into the definition of all CHAR/VARCHAR/domains/columns/parameters, even if their CHARSET is equal to the default CHARSET of the database.
SeparateComputedBy	Specifies whether computed fields should be extracted separately.
SetGenerators	Use to set generator values.
ExtractDescriptions	Determines whether database object descriptions should be included into the generated script.
DescriptionsAsUpdate	Determines whether the raw UPDATE statement will be used for object descriptions instead of the IBExpert-specific DESCRIBE statement.
ExtractPrivileges	Use to extract privileges.
OnlySelectedPrivileges	If used only privileges of the selected objects will be included into the resulting script. Otherwise ALL privileges will be extracted.
UseReinsert	Determines whether the IBExpert REINSERT command should be used to insert multiple data records.
ExtractBLOBs	Determines whether blob values should be extracted.
ExcludeIBE	Use to omit database objects with the prefix IBE$.
ExcludeTMP	Use to omit database objects with the prefix TMP$ (InterBase 7.x).
DecodeDomains	Determines whether domain definitions will be extracted as comments to the corresponding table fields.
CommitAfter=X	This option defines the number of records before inserting the COMMIT statement into the script. The default value is 500, i.e. 500 insert commands are performed and then committed.
MaxFileSize=X	Defines the maximum file size of script files (in megabytes). The default value is 0, this means that there will be no file splitting.
DateFormat=<format>	Specifies the format of date values and date part of timestamp values.
Domains=<objects_list>	Specifies list of domains to be extracted. Items should be separated with comma. If this option is not defined *all* domains will be extracted.
Tables=<objects_list>	Specifies list of tables to be extracted. Items should be separated with comma. If this option is not defined *all* tables will be extracted.
Views=<objects_list>	Specifies list of views to be extracted. Items should be separated with comma. If this option is not defined *all* views will be extracted.
Triggers=<objects_list>	Specifies list of triggers to be extracted. Items should be separated with comma. If this option is not defined *all* triggers will be extracted.
Procedures=<objects_list>	Specifies list of procedures to be extracted. Items should be separated with comma. If this option is not defined *all* procedures will be extracted.
Generators=<objects_list>	Specifies list of generators to be extracted. Items should be separated with comma. If this option is not defined *all* generators will be extracted.
Exceptions=<objects_list>	Specifies list of exceptions to be extracted. Items should be separated with comma. If this option is not defined *all* exceptions will be extracted.
UDFs=<objects_list>	Specifies list of UDFs to be extracted. Items should be separated with comma. If this option is not defined *all* UDFs will be extracted.
Roles=<objects_list>	Specifies list of roles to be extracted. Items should be separated with comma. If this option is not defined *all* roles will be extracted.

DataTables=<objects_list>	Specifies the list of tables from which data should be extracted. If this option is not defined NO data will be extracted. You can use the ALL keyword as a list of objects to specify that all objects of that type must be extracted. You can use the NONE keyword as a list of objects to omit all objects of that type.
CallbackProc	A callback IBEBlock which will be executed for each record processed whilst comparing data. The callback IBEBlock must have at least one input parameter, which will be used to pass a number of processed records within it.
UseComment	New to IBExpert version 2005.09.25 for support of the Firebird 2 COMMENT ON statement.
UseSequence	New to IBExpert version 2007.12.01 for support of the Firebird 2.x CREATE / ALTER SEQUENCE.

Examples of usage

1. Extracting domain definitions in VCS-files:

```
   execute ibeblock
   as
   begin
      cbb = 'execute ibeblock (LogLine variant)
        as
        begin
          ibec_progress(LogLine);
        end';

      db = ibec_GetDefaultConnection();
 ibec_ExtractMetadata(db, 'E:\Domains\', 'Domains=ALL; Tables=NONE; Views=NONE;
 Triggers=NONE; Procedures=NONE; Generators=NONE;
 Exceptions=NONE; UDFs=NONE; Roles=NONE;
  VCSFiles', cbb);
    end;
```

2. Complete metadata extract:

```
   execute ibeblock
   as
   begin
      cbb = 'execute ibeblock (LogLine variant)
        as
        begin
          ibec_progress(LogLine);
        end';

   db = ibec_GetDefaultConnection();
 ibec_ExtractMetadata(db, 'E:\meta.sql', 'GenerateCreate; ExtractPrivileges; ExtractDescriptions',
 cbb);
   end;
```

3. Extracting data from specified tables:

```
   execute ibeblock
   as
   begin
      cbb = 'execute ibeblock (LogLine variant)
        as
        begin
          ibec_progress(LogLine);
        end';

      db = ibec_GetDefaultConnection();
 ibec_ExtractMetadata(db, 'E:\data.sql', 'Domains=NONE; Tables=NONE; Views=NONE; Triggers=NONE;

 Procedures=NONE; Generators=NONE;
 Exceptions=NONE; UDFs=NONE; Roles=NONE;DataTables=IBE$$TEST_DATA, MY_TABLE;
 ExtractBLOBs;UseReinsert; CommitAfter=1000', cbb);
    end;
```

4. Using the IncludeCharset parameter:

```
 ibec_ExtractMetadata(db, 'E:\meta.sql', 'GenerateCreate;IncludeCharset;ExtractPrivileges; ExtractDescriptions',cbb);
```

See also:
Extract Metadata
Extract metadata using IBEBlock
Specifying WHERE clauses in ibec_ExtractMetadata
ibec_CompareMetadata

Specifying WHERE clauses in ibec_ExtractMetadata

Since IBExpert version 2007.07.18 `ibec_ExtractMetadata` allows specification of `WHERE` clauses for each data table. To specify these clauses you should create variable with a list of `WHERE`'s in the form `<table_name>=<where_clause>`:

```
WhereClauses[0] = 'HELP_ITEMS=where item_id > 1000';
WhereClauses[1] = 'GOODS=where id < 500000';
WhereClauses[2] = 'DT_TRANSFER=where transfer_id in (4, 6, 7)';
```

and indicate the variable name in the `WhereVar` option of the `Options` parameter of the function:

```
 WhereVar=WhereClauses;
```

Example

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (
           LogLine variant)
         as
         begin
           ibec_progress(LogLine);
         end';
     WhereClauses[0] = 'HELP_ITEMS=where item_id > 1000';
  WhereClauses[1] = 'GOODS=where id < 500000';
  WhereClauses[2] = 'DT_TRANSFER=where transfer_id in (4, 6, 7)';
     DB = ibec_CreateConnection(__ctInterBase,

'DBName="LOCALHOST/3060:D:\FB2_DATA\FORMTEST.FDB";
                                ClientLib=C:\Program Files\Firebird\bin\fbclient.dll;
                                User=SYSDBA; Password=masterkey; Names=NONE; SqlDialect=3');
  try
    ibec_ExtractMetadata(DB, 'D:\myscript.sql',
                             'GenerateCreate;
                              IncludePassword;
                              SetGenerators;
                              ExtractDescriptions;
                              UseComment;
                              MaxFileSize=500;
                              DecodeDomains;
                              ExtractBLOBs;
                              TrimStrings;
                              DateFormat=YYYY-MM-DD;
                              Domains=NONE;
                              Tables=NONE;
                              Views=NONE;
                              Procedures=NONE;
                              Triggers=NONE;
                              Generators=NONE;
                              Exceptions=NONE;
                              UDFs=NONE;
                              Roles=NONE;
                              DataTables=HELP_ITEMS,GOODS,DT_TRANSFER;
                              WhereVar=WhereClauses', cbb);
```

finally:

```
    ibec_CloseConnection(DB);
  end;
end;
```

See also:
`ibec_ExtractMetadata`

ibec_BackupDatabase

Syntax

```
function ibec_BackupDatabase(DatabaseToBackup : string;
         BackupFiles :string; Options : string;
         CallbackBlock : string) : variant;
```

Description

The `ibec_BackupDatabase` starts the backup process using the server *Services Manager*. It returns `NULL` if the backup process is successful, otherwise it returns an error message.

Options

`DatabaseToBackup`	Full connection string to the database including server name or IP address if the database is located on a remote server (for example, `123.123.123.123:D:\DATA\MyDB.fdb`).
`BackupFiles`	List of backup files delimited with semicolon. Each list item should be formatted as `<file_name>=<file_size>`.
`<file_size>`	Specifies the length of the result backup file in bytes (no suffix), kilobytes (`K`), megabytes (`M`) or gigabytes (`G`). *IMPORTANT:* All backup files will be created on the server side because of the use of the Services Manager!
`Options`	A list of backup options delimited with semicolon. Possible options are:
`USER=<user_name>`	User name
`PASSWORD=<password>` or `PAS=<password>`	Password.
`CLIENTLIB=<client_lib_name>`	Name of clientlib dll; `gds32.dll` will be used if not specified.
`IGNORE` (or `IG`)	Ignore bad checksums.
`LIMBO` (or `L`)	Ignore transactions in limbo.
`METADATA` (or `META_DATA`, or `M`)	Backup metadata only.
`GARBAGECOLLECT` (or `GARBAGE_COLLECT`, or `G`)	Inhibit garbage collection.
`OLDDESCRIPTIONS` (or `OLD_DESCRIPTIONS`, or `OL`)	Save old style metadata descriptions.
`NONTRANSPORTABLE` (or `NON_TRANSPORTABLE`, or `NT`)	Non-transportable backup file format.
`CONVERT` (or `CO`)	Backup external files as tables.
`LOGFILE=<log_file_name>`	Name of output log file.
`CallbackBlock`	A callback IBEBlock which will be executed for each output line. The callback IBEBlock must have at least one input parameter, which will be used to pass an output line within it. If there is no callback block use `NULL` or an empty string as a value of this.

Example 1

Backup a database to a single backup file with no output (silent mode):

```
  execute ibeblock
  as
  begin
    res = ibec_BackupDatabase('LOCALHOST:D:\FB2_DATA\TESTDB.FDB',
                              'E:\TESTDB.FBK',
'ClientLib=C:\Program Files\Firebird\Bin\fbclient.dll;
                                 Password=masterkey; User=SYSDBA; G;',
                              null);
    if (res is null) then
      ibec_ShowMessage('Backup completed successfully.);
    else
      ibec_ShowMessage(res);
  end
```

Example 2

Backup a database to multiple backup files with full output:

```
  execute ibeblock
  as
  begin
    cbb = 'execute ibeblock (LogStr variant)
           as
           begin
             ibec_Progress(LogStr);
           end';

    res = ibec_BackupDatabase('LOCALHOST:D:\FB2_DATA\TESTDB.FDB',
'E:\TESTDB_1.FBK=200M; E:\TESTDB_2.FBK=200M; E:\TESTDB_3.FBK=200M',
'ClientLib=C:\Program Files\Firebird\Bin\fbclient.dll;
Password=masterkey; User=SYSDBA; IGNORE; L; LogFile=E:\Backup.log',
```

```
                                cbb);
      if (res is null) then
        ibec_ShowMessage('Backup completed successfully.);
      else
        ibec_ShowMessage(res);
    end
```

See also:
ibec_RestoreDatabase

ibec_RestoreDatabase

Syntax

```
function ibec_RestoreDatabase(BackupFiles : string; RestoreTo : string;
                              Options : string; CallbackBlock : string) : variant;
```

Description

The `ibec_RestoreDatabase` starts the restore process using the server *Services Manager*. It returns `NULL` if the restore process succeeded, otherwise it returns an error message.

Options

`BackupFiles`	List of backup files delimited with semicolon.
`RestoreTo`	List of database files delimited with semicolon. Each list item (in case of restore to multiple files) should be in format `<db_file_name>=<file_size_in_pages>`.
`<db_file_name>`	Full connection string to the database including server name or IP address if the database is located on a remote server (for example, `123.123.123.123:D:\DATA\MyDB.fdb`).
`<file_size_in_pages>`	Size of the database file in pages (!).
`Options`	List of restore options delimited with semicolon. Possible options are:
`USER=<user_name>`	User name.
`PASSWORD=<password>` or `PAS=<password>`	Password.
`CLIENTLIB=<client_lib_name>`	Name of clientlib dll; `gds32.dll` will be used if not specified.
`PAGESIZE=<page_size>` or `PAGE_SIZE=<page_size>`	Page size of the restored database.
`PAGEBUFFERS=<buffers>` or `BUFFERS=<buffers>` or `BU=<buffers>`	Overrides page buffers default.
`INACTIVE` (or `DEACTIVATEINDEXES`, or `I`)	Deactivate indexes during restore.
`KILL` (or `NOSHADOWS`, or `K`)	Restore without creating shadows.
`NO_VALIDITY` (or `NOVALIDITY`, or `N`)	Do not restore database validity conditions.
`ONE_AT_A_TIME` (or `ONEATATIME`, or `O`)	Restore one table at a time (commit after each table).
`REPLACE_DATABASE` (or `REPLACEDATABASE`, or `REP`)	Replace database from backup file.
`CREATE_DATABASE` (or `CREATEDATABASE`, or `C`)	Create database from backup file.
`USE_ALL_SPACE` (or `USEALLSPACE`, or `USE`)	Do not reserve space for record versions.
`META_DATA` (or `METADATA`, or `M`)	Restore metadata only.
`LOGFILE=<log_file_name>`	Name of output log file.
`CallbackBlock`	Callback IBEBlock which will be executed for each output line. The callback IBEBlock must have at least one input parameter, which will be used to pass an output line within it. If there is no callback block use `NULL` or an empty string as a value of this parameter.

Example 1

Restore database from single backup file with no output (silent mode):

```
  execute ibeblock
  as
  begin
    res = ibec_RestoreDatabase('E:\TESTDB.FBK',
                               'LOCALHOST:E:\TESTDB.FBK',
'ClientLib=C:\Program Files\Firebird\Bin\fbclient.dll;
Password=masterkey; User=SYSDBA;OneAtATime; PageSize=8192; C',
                               null);
    if (res is null) then
      ibec_ShowMessage('Restore completed successfully.);
    else
      ibec_ShowMessage(res);
  end
```

Example 2

Restore database from multiple backup files to single database file with full output:

```
  execute ibeblock
  as
  begin
    cbb = 'execute ibeblock (LogStr variant)
           as
           begin
             ibec_Progress(LogStr);
```

```
            end';

 res = ibec_RestoreDatabase('E:\TESTDB_1.FBK; E:\TESTDB_2.FBK; E:\TESTDB_3.FBK',
                                 'LOCALHOST:E:\TESTDB.FBK',
 'ClientLib=C:\Program Files\Firebird\Bin\fbclient.dll;
 Password=masterkey; User=SYSDBA; C; REP; O; LogFile=E:\Restore.log',
                                 cbb);
      if (res is null) then
        ibec_ShowMessage('Restore completed successfully.);
      else
        ibec_ShowMessage(res);
    end
```

Example 3

Restore database from multiple backup files to multiple database files with full output:

```
    execute ibeblock
    as
    begin
      cbb = 'execute ibeblock (LogStr variant)
             as
             begin
               ibec_Progress(LogStr);
             end';

 res = ibec_RestoreDatabase('E:\TESTDB_1.FBK; E:\TESTDB_2.FBK; E:\TESTDB_3.FBK',
                                 'LOCALHOST:E:\TESTDB1.FBK=20000;
                                     LOCALHOST:E:\TESTDB2.FBK=20000;
                                     LOCALHOST:E:\TESTDB3.FBK',
 'ClientLib=C:\Program Files\Firebird\Bin\fbclient.dll;
 Password=masterkey; User=SYSDBA; C; REP; O; BU=3000;
                                     LogFile=E:\Restore.log',
                                 cbb);
      if (res is null) then
        ibec_ShowMessage('Restore completed successfully.);
      else
        ibec_ShowMessage(res);
    end
```

See also:
`ibec_BackupDatabase`

ibec_GetConnectionProp

The `ibec_GetConnectionProp` function was implemented in IBExpert version 2006.10.14, and offers the additional possibility to get the server version of the active connection.

Example

```
 SrvVerStr = ibec_GetConnectionProp(Conn, 'ServerVersion');
```

ibec_GetCurrentDir

The `ibec_GetCurrentDir` function was implemented in IBExpert version 2006.10.14. This function returns the fully qualified name of the current directory.

Example

```
 CurrDir = ibec_GetCurrentDir();
```

ibec_GetRunDir

The `ibec_GetRunDir` function was implemented in IBExpert version 2008.02.19. This function returns the path of the currently executing program (`IBExpert.exe` of `IBEScript.exe`.

Syntax

```
 function ibec_GetRunDir : string;
```

ibec_GetUserDBConnection

The `ibec_GetUserDBConnection` function was implemented in IBExpert version 2008.02.19. It returns the pointer to the User Database (found in the IBExpert Options menu under Environment Options / User Database, if it is used. Otherwise this function returns `NULL`.

Syntax

```
function ibec_GetUserDBConnection : variant;
```

Example

```
    execute ibeblock
  as
  begin
    CRLF = ibec_CRLF();
    sTab = ibec_Chr(9);
    sLine = '==============================================';
         UserDB = ibec_GetUserDBConnection();
    if (UserDB is not null) then
    begin
      sMes = '';
           sHost = ibec_GetConnectionProp(UserDB, 'HostName');
      sFile = ibec_GetConnectionProp(UserDB, 'FileName');
      sServerVersion = ibec_GetConnectionProp(UserDB, 'ServerVersion');
      sDBSqlDialect = ibec_GetConnectionProp(UserDB, 'DBSqlDialect');
      sClientLib = ibec_GetConnectionProp(UserDB, 'ClientLib');
      sUser = ibec_GetConnectionProp(UserDB, 'UserName');
      sPass = ibec_GetConnectionProp(UserDB, 'Password');
      sNames = ibec_GetConnectionProp(UserDB, 'lc_ctype');
           iPageSize = ibec_GetConnectionProp(UserDB, 'PageSize');
      iSweep = ibec_GetConnectionProp(UserDB, 'SweepInterval');
      iODSMinorVersion = ibec_GetConnectionProp(UserDB, 'ODSMinorVersion');
      iODSMajorVersion = ibec_GetConnectionProp(UserDB, 'ODSMajorVersion');
           sMes = 'User Database properties' + CRLF + sLine + CRLF;
      sMes .= 'Database host: ';
      if (sHost = '') then
        sMes .= sTab + '(local)';
      else
        sMes .= sTab + sHost;
           sMes .= CRLF +
              'Database file: ' + sTab + sFile + CRLF +
              'Server version: ' + sTab + sServerVersion + CRLF +
              'Client library: ' + sTab + sClientLib + CRLF + CRLF +
                   'Page size, bytes: ' + sTab + ibec_Cast(iPageSize, __typeString) + CRLF +
              'Sweep interval: ' + sTab + sTab + ibec_Cast(iSweep, __typeString) + CRLF +
              'ODS version: ' + sTab + sTab + ibec_Cast(iODSMajorVersion, __typeString) + '.' +
                   ibec_Cast(iODSMinorVersion, __typeString) + CRLF + CRLF +
                   'Connection username: ' + sTab + sUser + CRLF +
              'Connection password: ' + sTab + sPass + CRLF +
              'Connection charset: ' + sTab + sNames + CRLF;
                ibec_UseConnection(UserDB);
           sMes .= CRLF + CRLF + 'User Database tables' + CRLF + sLine + CRLF;
      for select rdb$relation_name
          from rdb$relations
          where (rdb$system_flag is null) or (rdb$system_flag = 0)
          order by rdb$relation_name
          into :RelName
      do
      begin
        RelName = ibec_Trim(RelName);
        sMes .= RelName + CRLF;
      end
      commit;
           ibec_ShowMessage(sMes);
    end
  end
```

ibec_ibe_GetActiveDatabaseID

The `ibec_ibe_GetActiveDatabaseID` function was implemented in IBExpert version 2008.02.19. It returns the unique identifier of the active (currently used) database within IBExpert. If there is no active database `ibec_ibe_GetActiveDatabaseID` returns `-1`.

Syntax

```
function ibec_ibe_GetActiveDatabaseID : integer;
```

ibec_ibe_GetDatabaseProp

The `ibec_ibe_GetDatabaseProp` function was implemented in IBExpert version 2008.02.19. It returns the value of specifed database property.

Syntax

```
function ibec_ibe_GetDatabaseProp(DatabaseID : integer; PropertyName : string) : variant;
```

Following properties are available:

ALIAS	Alias of the registered database.
CLIENTLIB	Name of the client library file specified in the Database Registration Info.
SERVERNAME or HOSTNAME	Server name.
FILENAME or DBNAME	Database file name.
PASSWORD	Password specified in the database regstration info.
USERNAME or USER_NAME or USER	User name.
ROLENAME or ROLE_NAME or ROLE	Role name.
NAMES or LC_CTYPE or CHARSET	Connection charset.
CONNECTIONSTRING or CONNECTION_STRING	Connection string.
ACTIVE or CONNECTED	Returns TRUE if the database is active and FALSE if it is not.

Example

```
    execute ibeblock as
begin
  CRLF = ibec_CRLF();
  ActiveDB = ibec_ibe_GetActiveDatabaseID();
  if (ActiveDB is not null) then
  begin
    if (ActiveDB = -1) then
      Exit;
    sAlias = ibec_ibe_GetDatabaseProp(ActiveDB, 'Alias');
    sClientLib = ibec_ibe_GetDatabaseProp(ActiveDB, 'ClientLib');
    sHost = ibec_ibe_GetDatabaseProp(ActiveDB, 'HostName');
    sFileName = ibec_ibe_GetDatabaseProp(ActiveDB, 'FileName');
    sPassword = ibec_ibe_GetDatabaseProp(ActiveDB, 'Password');
    sUser = ibec_ibe_GetDatabaseProp(ActiveDB, 'User');
    sRole = ibec_ibe_GetDatabaseProp(ActiveDB, 'Role');
    sCharset = ibec_ibe_GetDatabaseProp(ActiveDB, 'Names');
    sConnectionStr = ibec_ibe_GetDatabaseProp(ActiveDB, 'ConnectionString');
    bActive = ibec_ibe_GetDatabaseProp(ActiveDB, 'Connected');
         s = 'Database alias: ' + sAlias + CRLF +
        'Client library: ' + sClientLib + CRLF +
        'Server name: ' + sHost + CRLF +
        'Database file name: ' + sFileName + CRLF +
        'User name: ' + sUser + CRLF +
        'Password: ' + sPassword + CRLF +
        'Role: ' + sRole + CRLF +
        'Charset: ' + sCharset + CRLF +
        'Connection string: ' + sConnectionStr;
         if (bActive) then
      s .= CRLF + CRLF + 'Database is active.';
         ibec_ShowMessage(s);
  end
end
```

ibec_SetDatabaseProp

Dataset functions

The following dataset-handling functions are available in IBEBlock:

Function	Description
ibec_CopyData	Returns number of records copied from SrcConnection to DestConnection.
ibec_Array	Returns a one-dimensional 0-based array of values.
ibec_ds_Append	Adds a new, empty record to the end of the dataset.
ibec_ds_Cancel	Cancels modifications to the active record if those changes are not yet posted.
ibec_ds_Delete	Deletes the active record and positions the cursor on the next record.
ibec_ds_Edit	Enables editing of data in the dataset.
ibec_ds_Eof	Indicates whether or not a cursor is positioned at the last record in a dataset.
ibec_ds_Bof	Indicates whether or not a cursor is positioned at the first record in a dataset.
ibec_ds_FieldCount	Returns the number of fields associated with the dataset.
ibec_ds_FieldName	Returns the name of specified field.
ibec_ds_FieldType	
ibec_ds_FieldTypeN	Returns the native type of specified field.
ibec_ds_First	Positions the cursor on the first record in the dataset.
ibec_ds_GetField	Returns value of specified field.
ibec_ds_Insert	
ibec_ds_Last	Positions the cursor on the last record in the dataset.
ibec_ds_Locate	Locates single or multiple specified search values in a dataset.
ibec_ds_Next	Positions the cursor on the next record in the dataset.
ibec_ds_Post	
ibec_ds_Prior	Positions the cursor on the previous record in the dataset.
ibec_ds_SetField	
ibec_ds_Sort	Sorts datasets according to the SortFields specification.

ibec_CopyData

This function was implemented in IBExpert version 2006.08.12. It is intended for the quick copying of data from one connection (ODBC or Firebird/InterBase) to another (Firebird/InterBase only).

Syntax

```
function ibec_CopyData(SrcConnection : variant;
                       DestConnection : variant;
                       DestTableName : string;
                       SelectStatement : string;
                       Options : string;
                       CallbackBlock : variant) : integer;
```

Description

The `ibec_CopyData` function returns the number of records copied from `SrcConnection` to `DestConnection`.

Example

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (RecNo integer)
         as
         begin
           if (ibec_mod(RecNo, 100) = 0) then
             ibec_Progress(RecNo ||  records copied...);
         end';

  OdbcCon = ibec_CreateConnection(__ctODBC, 'DBQ=C:\IBE Demo\demo.mdb; DRIVER=Microsoft Access Driver

(*.mdb)');

  DB = ibec_CreateConnection(__ctInterBase,
                             'DBName="localhost:D:\FB2_DATA\IBEHELP.FBA";
                              ClientLib=C:\Program Files\Firebird\bin\fbclient.dll;
                              user=SYSDBA; password=masterkey; names=WIN1251; sqldialect=3');
  try
    use DB;
    if (exists(select * from rdb$relations where rdb$relation_name = 'IBEC_COPYDATA'))

then
    begin
      execute statement 'drop table IBEC_COPYDATA';
      commit;
    end;

    Country = 'US';

    RecCount = ibec_CopyData(OdbcCon, DB, 'IBEC_COPYDATA',
                             'SELECT * FROM CUSTOMER WHERE COUNTRY < :Country',
                             'CommitAfter=100; EmptyTable; CreateTable; DontQuoteIdents',
                             cbb);

    if (RecCount is not null) then
      ibec_ShowMessage(RecCount || ' records copied successfully.');


  finally
    ibec_CloseConnection(DB);
    ibec_CloseConnection(OdbcCon);
  end;
end
```

ibec_Array

The `ibec_Array` function was implemented in IBExpert version 2007.02.22. This function returns a one-dimensional 0-based array of values.

Syntax

```
function ibec_Array(val1 [, val2, ..., valN) : variant;
```

Example

```
MyVar = ibec_Array('Some text', 23, NULL, 56.32);
```

The code above is equal to following:

```
MyVar[0] = 'Some text';
MyVar[1] = 23;
MyVar[2] = NULL;
MyVar[4] = 56.32
```

And since IBExpert version 2007.12.01 it is also possible to pass arrays into IBEBlocks:

Example

```
execute ibeblock
as
begin
  MyBlock = 'execute ibeblock (inparam variant)
             as
             begin
               ibec_ShowMessage(inparam[0] || inparam[1] || inparam[2]);
             end';       MyVar[0] = 'Hello';
  MyVar[1] = ', ';
  MyVar[2] = 'World!';
  execute ibeblock MyBlock(MyVar);
end
```

See also:
FOREACH statement

ibec_ds_Append

Adds a new, empty record to the end of the dataset.

Syntax

```
function ibec_ds_Append(Dataset : variant) : variant;
```

Description

Call `ibec_ds_Append` to:

- Open a new, empty record at the end of the dataset.
- Set the active record to the new record.
- After a call to `ibec_ds_Append`, you can enter data in the fields of the record, and can then post those changes to the dataset using `ibec_ds_Post`.

ibec_ds_Cancel

Cancels modifications to the active record if those changes are not yet posted.

Syntax

```
function ibec_ds_Cancel(Dataset : variant) : variant;
```

Description

Call `ibec_ds_Cancel` to undo modifications made to one or more fields belonging to the active record. As long as those changes are not already posted to the dataset, `ibec_ds_Cancel` returns the record to its previous state, and sets the dataset state to `__dsBrowse`.

ibec_ds_Close

ibec_ds_Delete

Deletes the active record and positions the cursor on the next record.

Syntax

```
function ibec_ds_Delete(Dataset : variant) : variant;
```

Description

Call `ibec_ds_Delete` to remove the active record from the database. If the dataset is inactive, `ibec_ds_Delete` raises an exception. Otherwise `ibec_ds_Delete`:

- Verifies that the dataset is not empty (and raises an exception if it is).
- Deletes the record.
- Frees the buffers allocated for the record.
- Puts the dataset into `__dsBrowse` mode :
- Resynchronizes the dataset to position the cursor on the next undeleted record.

ibec_ds_Edit

Enables editing of data in the dataset.

Syntax

```
procedure ibec_ds_Edit(Dataset : variant) : variant;
```

Description

Call `ibec_ds_Edit` to permit editing of the active record in a dataset. `ibec_ds_Edit` determines the current state of the dataset. If the dataset is empty, `ibec_ds_Edit` calls `ibec_ds_Insert`.

ibec_ds_Eof

Indicates whether or not a cursor is positioned at the last record in a dataset.

Syntax

```
function ibec_ds_Eof(Dataset : variant) : boolean;
```

Description

Call `ibec_ds_Eof` to determine if the cursor is positioned at the last record in a dataset. If `ibec_ds_Eof` returns `True`, the cursor is unequivocally on the last row in the dataset. Otherwise this function returns `False`.

Example

```
execute ibeblock
as
begin
  select * from RDB$FIELDS as dataset MyDataset;

  while (not ibec_ds_Eof(MyDataset)) do
  begin
    ...
    ibec_ds_Next(MyDataset);
  end

  ...

  close dataset MyDataset;
end
```

See also:
`SELECT ... AS DATASET`
`ibec_ds_Bof`
`ibec_ds_First`
`ibec_ds_Last`
`ibec_ds_Next`
`ibec_ds_Prior`

ibec_ds_Export

ibec_ds_Bof

Indicates whether or not a cursor is positioned at the first record in a dataset.

Syntax

```
function ibec_ds_Bof(Dataset : variant) : boolean;
```

DescriptiAon

Call `ibec_ds_Bof` to determine if the cursor is positioned at the first record in a dataset. If `bec_ds_Bof` returns `True`, the cursor is unequivocally on the first row in the dataset. Otherwise this function returns `False`.

Example

```
execute ibeblock
as
begin
  select * from RDB$FIELDS as dataset MyDataset;

  ibec_ds_Last(MyDataset);
  while (not ibec_ds_Bof(MyDataset)) do
  begin
    ...
    ibec_ds_Prior(MyDataset);
  end

  ...

  close dataset MyDataset;
end
```

See also:
`SELECT ... AS DATASET`
`ibec_ds_Eof`
`ibec_ds_First`
`ibec_ds_Last`
`ibec_ds_Next`
`ibec_ds_Prior`

ibec_ds_FieldCount

Returns the number of fields associated with the dataset.

Syntax

```
function ibec_ds_FieldCount(Dataset : variant) : integer;
```

Description

Call `ibec_ds_FieldCount` to determine the number of fields associated with the dataset.

ibec_ds_FieldName

Returns the name of specified field.

Syntax

```
function ibec_ds_FieldName(Dataset : variant; FieldIndex : integer) : variant;
```

Example

```
execute ibeblock
returns (FieldName varchar(31), FieldType varchar(100))
as
begin
  select * from rdb$fields
  where (1 = 0)
  as dataset RdbFields;

  iCount = ibec_ds_FieldCount(RdbFields);
  i = 0;
  while (i < iCount) do
  begin
    FieldName = ibec_ds_FieldName(RdbFields, i);
    FieldType = ibec_ds_FieldTypeN(RdbFields, i);
    suspend;
    i = i + 1;
  end;

  close dataset RdbFields;
end
```

See also:
ibec_ds_FieldType
ibec_ds_FieldTypeN

ibec_ds_FieldType

See also:
ibec_ds_FieldName
ibec_ds_FieldTypeN

ibec_ds_FieldTypeN

Returns the native type of specified field.

Syntax

```
function ibec_ds_FieldTypeN(Dataset : variant; Field : variant) : variant;
```

Example

```
execute ibeblock
returns (FieldName varchar(31), FieldType varchar(100))
as
begin
  select * from rdb$fields
  where (1 = 0)
  as dataset RdbFields;

  iCount = ibec_ds_FieldCount(RdbFields);
  i = 0;
  while (i < iCount) do
  begin
    FieldName = ibec_ds_FieldName(RdbFields, i);
    FieldType = ibec_ds_FieldTypeN(RdbFields, i);
    suspend;
    i = i + 1;
  end;

  close dataset RdbFields;
end
```

See also:
ibec_ds_FieldName
ibec_ds_FieldType

ibec_ds_First

Positions the cursor on the first record in the dataset.

Syntax

```
function ibec_ds_First(Dataset : variant) : variant;
```

Description

Call `ibec_ds_First` to position the cursor on the first record in the dataset and make it the active record.

See also:
`SELECT ... AS DATASET`
`ibec_ds_Bof`
`ibec_ds_Last`
`ibec_ds_Next`
`ibec_ds_Prior`

ibec_ds_GetField

Returns value of specified field.

Syntax

```
function ibec_ds_GetField(Dataset : variant; Field : variant) : variant;
```

See also:
Example: Recreating indices 2

ibec_ds_Insert

ibec_ds_Last

Positions the cursor on the last record in the dataset.

Syntax

```
function ibec_ds_Last(Dataset : variant) : variant;
```

Description

Call `ibec_ds_Last` to position the cursor on the last record in the dataset and make it the active record.

Example

```
execute ibeblock
as
begin
  select * from RDB$FIELDS as dataset MyDataset;

  ibec_ds_Last(MyDataset);
  while (not ibec_ds_Bof(MyDataset)) do
  begin
    ...
    ibec_ds_Prior(MyDataset);
  end

  ...

  close dataset MyDataset;
end
```

See also:
SELECT ... AS DATASET
ibec_ds_Bof
ibec_ds_First
ibec_ds_Next
ibec_ds_Prior

ibec_ds_Locate

Locates single or multiple specified search values in a dataset.

Syntax

```
function ibec_ds_Locate(Dataset : variant; KeyFields : string;
                        KeyValues : array of variant; Options : integer) :  boolean;
```

ibec_ds_Locate	searches Dataset for a specified record and makes that record the active record.
KeyFields	is a string containing a semicolon-delimited list of field names in which to search.
KeyValues	is a variant array containing the values to match in the key fields.

Description

ibec_ds_Locate locates single or multiple specified search values in a dataset. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the ibec_Array function.

Examples

```
ibec_ds_Locate('Company;Contact;Phone', ibec_Array('Sight Diver', 'P', '408-431-1000'), __loPartialKey);
```

or

```
Keys[0] = 'Sight Diver';
Keys[1] = 'P';
Keys[2] = '408-431-1000';
ibec_ds_Locate('Company;Contact;Phone', Keys, __loPartialKey);
```

Options is a set of flags that optionally specifies additional search latitude when searching on string fields. If Options contains the __loCaseInsensitive flag, then ibec_ds_Locate ignores case when matching fields. If Options contains the __loPartialKey flag, then ibec_ds_Locate allows partial-string matching on strings in KeyValues. If Options is 0 or NULL or if the KeyFields property does not include any string fields, Options is ignored.

This function returns True if a record is found that matches the specified criteria and the cursor repositioned to that record. Otherwise it returns False.

Example

```
execute ibeblock
returns (FieldName varchar(100))
as
begin
  select * from rdb$relation_fields
  as dataset ds;
  try
    ibec_ds_Sort(ds, 'RDB$RELATION_NAME, RDB$FIELD_POSITION');
    res = ibec_ds_Locate(ds, 'RDB$RELATION_NAME', 'RDB$FIELDS', __loPartialKey);
    while (res) do
    begin
      FieldName = ibec_ds_GetField(ds, 'RDB$FIELD_NAME');
      FieldName = ibec_Trim(FieldName);
      suspend;
      ibec_ds_Next(ds);
      res = not ibec_ds_EOF(ds);
      if (res) then
      begin
        RelName = ibec_Trim(ibec_ds_GetField(ds, 'RDB$RELATION_NAME'));
        res = RelName = 'RDB$FIELDS';
      end;
    end;
  finally
    ibec_ds_Close(ds);
  end;
end
```

See also:
ibec_ds_Sort

ibec_ds_Next

Positions the cursor on the next record in the dataset.

Syntax

```
function ibec_ds_Next(Dataset : variant) : variant;
```

Description

Call `ibec_ds_Next` to position the cursor on the next record in the dataset and make it the active record.

Example

```
execute ibeblock
as
begin
  select * from RDB$FIELDS as dataset MyDataset;

  while (not ibec_ds_Eof(MyDataset)) do
  begin
    ...
    ibec_ds_Next(MyDataset);
  end

  ...

  close dataset MyDataset;
end
```

See also:
`SELECT ... AS DATASET`
`ibec_ds_Bof`
`ibec_ds_First`
`ibec_ds_Last`
`ibec_ds_Prior`

ibec_ds_Post

ibec_ds_Prior

Positions the cursor on the previous record in the dataset.

Syntax

```
function ibec_ds_Prior(Dataset : variant) : variant;
```

Description

Call `ibec_ds_Prior` to position the cursor on the previous record in the dataset and make it the active record.

Example

```
execute ibeblock
as
begin
  select * from RDB$FIELDS as dataset MyDataset;

  ibec_ds_Last(MyDataset);
  while (not ibec_ds_Bof(MyDataset)) do
  begin
    ...
    ibec_ds_Prior(MyDataset);
  end

  ...

  close dataset MyDataset;
end
```

See also:
`SELECT ... AS DATASET`
`ibec_ds_Bof`
`ibec_ds_First`
`ibec_ds_Last`
`ibec_ds_Next`

ibec_ds_SetField

ibec_ds_Sort

Sorts datasets according to the `SortFields` specification.

Syntax

```
function ibec_ds_Sort(Dataset : variant; SortFields : string) : variant;
```

Description

`ibec_ds_Sort` function sorts the specified `Dataset` according to the `SortFields` specification.

Example

```
execute ibeblock
as
begin
  select * from rdb$relation_fields
  as dataset ds;
  try
    ibec_ds_Sort(ds, 'RDB$RELATION_NAME ASC, RDB$FIELD_POSITION ASC');
    ibec_ds_Sort(ds, 'RDB$RELATION_NAME, RDB$FIELD_POSITION');
    ibec_ds_Sort(ds, '1, 2 DESC');       finally
    ibec_ds_Close(ds);
  end;
end;
```

See also:
ibec_ds_Locate

Managing Firebird/InterBase users

The following functions have been added to manage Firebird/InterBase users:

`ibec_CreateUser`	Creates a user.
`ibec_AlterUser`	Alters a user.
`ibec_RecreateUser`	Recreates a user.
`ibec_DropUser`	Deletes a user.
`ibec_GetUsers`	Retrieves a list of users from the server using the IBExpert *Services Manager*.
`ibec_GetUserProp`	

These functions use the Firebird/InterBase *Services Manager*, therefore they will not work with servers that do not support the Services Manager API.

ibec_CreateUser

Syntax

```
ibec_CreateUser(ConnectOptions, UserData : string) : variant;
```

All functions return `NULL` if there were no errors, otherwise they return an error message text.

`ConnectOptions` is a list of parameters to connect to the *Services Manager* delimited by semicolons. Possible options are:

`Server=<server_name>`	The name of the server. Also you can use `ServerName=<server_name>` to specify the server name.
`Protocol=<protocol>`	The network protocol with which to connect to the server. Possible values are `'Local'`, `'TCP'`, `'SPX'` and `'NamedPipe'`.
`User=<user_name>`	The user name.
`Password=<password>`	The password.
`ClientLib=<client_lib_name>`	The name of client library dll, by default `GDS32.DLL`.

Example

```
ibec_DropUser('Server=localhost; User=SYSDBA; Password=masterkey;
                  Protocol=TCP; ClientLib=gds32.dll', 'VASYA');
```

If the server name is not specified the connection will be established with the local server using the local protocol. TCP/IP will be used when the server name is specified but the protocol is not specified.

`UserData` is a list of user properties, delimited by semicolons. Possible properties are:

`UserName=<user_name>`	User name to create or modify; maximum 31 characters.
`Password=<password>`	Password for the user; maximum 31 characters, only the first 8 characters are significant.
`FirstName=<first_name>`	Optional first name of the person using this user name.
`MiddleName=<middle_name>`	Optional middle name of the person using this user name.
`LastName=<last_name>`	Optional last name of the person using this user name.
`UserID=<user_id>`	Optional userID number, defined in `/etc/passwd`, to assign to the user; reserved for future implementation.
`GroupID=<group_id>`	Optional groupID number, defined in `/etc/group`, to assign to the user; reserved for future implementation.

Example

```
ibec_CreateUser('Server=localhost; User=SYSDBA; Password=masterkey;
                   Protocol=TCP',
                 'UserName=BILL_GATES; Password=microsoft; FirstName=BILL;
                   LastName=GATES');
```

ibec_AlterUser

Syntax

```
ibec_AlterUser(ConnectOptions, UserData : string) : variant;
```

All functions return `NULL` if there were no errors, otherwise they return an error message text.

Please refer to `ibec_CreateUser` for the parameter lists for `ConnectOptions` and `UserData` options, and examples.

If the server name is not specified the connection will be established with the local server using the local protocol. TCP/IP will be used when the server name is specified but the protocol is not specified.

ibec_RecreateUser

The `ibec_RecreateUser` function first tests whether a specified user exists or not. In the case of the specified user existing, it deletes his login record and recreates it again using the properties specified. Otherwise it just creates a new login record.

Syntax

```
ibec_RecreateUser(ConnectOptions, UserData : string) : variant;
```

All functions return `NULL` if there were no errors, otherwise they return an error message text.

Please refer to `ibec_CreateUser` for the parameter lists for `ConnectOptions` and `UserData` options, and examples.

If the server name is not specified the connection will be established with the local server using the local protocol. TCP/IP will be used when the server name is specified but the protocol is not specified.

ibec_DropUser

Syntax

```
ibec_DropUser(ConnectOptions, UserName : string) : variant;
```

All functions return `NULL` if there were no errors, otherwise they return an error message text.

Please refer to `ibec_CreateUser` for the parameter lists for `ConnectOptions` and `UserData` options, and examples.

If the server name is not specified the connection will be established with the local server using the local protocol. TCP/IP will be used when the server name is specified but the protocol is not specified.

ibec_GetUsers

The `ibec_GetUsers` function was implemented in IBExpert version 2007.05.03. This function retrieves a list of users from the server using the IBExpert *Services Manager*.

Syntax

```
function ibec_GetUsers(ConnectOptions : string; UserNames : variant [; FullData : variant]) : variant;
```

The `ibec_GetUsers` returns `NULL` if no error occured, otherwise it returns an error message.

Example

The `UserNames` parameter: the following example returns a list of users registered on the server:

```
execute ibeblock
returns (UserName varchar(100),
         FirstName varchar(100),
         MiddleName varchar(100),
         LastName varchar(100))
as
begin
  res = ibec_GetUsers('Server=localhost/3065; User=SYSDBA; Password=masterkey;
                      ClientLib=C:\Program Files\Firebird\Bin\fbclient.dll',
                      UserNames, FullData);
  foreach (UserNames as UserName key UserIdx) do
  begin
    s = FullData[UserIdx];
    ini = ibec_ini_Open('');
    try
      ibec_ini_SetStrings(ini, s);

      FirstName = ibec_ini_ReadString(ini, 'UserData', 'FirstName', '');
      MiddleName = ibec_ini_ReadString(ini, 'UserData', 'MiddleName', '');
      LastName = ibec_ini_ReadString(ini, 'UserData', 'LastName', '');
    finally
      ibec_ini_Close(ini);
    end;
    suspend;
  end
end
```

ibec_GetUserProp

Date / Time functions

The following date/time functions are available in IBEBlock:

Function	Description
`ibec_Date`	Returns the current date (without the time part).
`ibec_Now`	Returns the current timestamp.
`ibec_Time`	Returns the current time.
`ibec_DayOfWeek`	Returns the day of the week as an integer.

See also:
ibec_FileDateTime

ibec_Date

Syntax

```
ibec_Date : Date;
```

`ibec_Date` returns the current date (without the time part).

ibec_Now

Syntax

```
ibec_Now : TimeStamp;
```

`ibec_Now` returns the current timestamp.

ibec_Time

Syntax

```
ibec_Time : Time;
```

`ibec_Time` returns the current time.

ibec_DayOfWeek

Syntax

```
ibec_DayOfWeek(Date : TimeStamp) : integer;
```

`ibec_DayOfWeek` returns the day of the week as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

Windows Registry functions

The following functions are available in IBEBlock to handle work with the Windows Registry:

Function	Description
ibec_reg_Open	Instantiates a registry object.
ibec_reg_Close	Closes the current key and frees the resources allocated for a registry object when no longer needed.
ibec_reg_OpenKey	Makes the specified key the current key.
ibec_reg_CloseKey	Writes the current key to the registry and closes the key.
ibec_reg_DeleteKey	Removes a specified key and its associated data from the registry.
ibec_reg_CreateKey	Creates a new key in the registry.

The following functions are intended for reading and writing data from/to the Windows Registry:

ibec_reg_WriteString
ibec_reg_ReadString
ibec_reg_WriteBool
ibec_reg_ReadBool
ibec_reg_WriteDate
ibec_reg_ReadDate
ibec_reg_WriteDateTime
ibec_reg_ReadDateTime
ibec_reg_WriteTime
ibec_reg_ReadTime
ibec_reg_WriteInteger
ibec_reg_ReadInteger
ibec_reg_WriteFloat
ibec_reg_ReadFloat

ibec_reg_Open

`ibec_reg_Open` instantiates a registry object.

Syntax

```
function ibec_reg_Open(RootKey : HKEY; Access : LongWord) : variant;
```

Parameters

`RootKey` determines the hierarchy of sub-keys an application can access. Possible values are `__HKEY_CLASSES_ROOT`, `__HKEY_CURRENT_USER`, `__HKEY_LOCAL_MACHINE`, `__HKEY_USERS` and `__HKEY_CURRENT_CONFIG`.

Access determines the level of security access to use when opening keys; it is currently ignored when `KEY_ALL_ACCESS` is used.

See also:
Example of ibec_reg_xxx functions: daily backup User Database
IBExpert After Start Script

ibec_reg_Close

Syntax

```
function ibec_reg_Close(Registry : variant) : variant;
```

Description

`ibec_reg_Close` closes the current key and frees the resources allocated for a registry object when it is no longer needed.

See also:
Example of ibec_reg_xxx functions: daily backup User Database
IBExpert After Start Script

ibec_reg_OpenKey

Call `ibec_reg_OpenKey` to make a specified key the current key.

Syntax

```
function ibec_reg_OpenKey(Registry : variant; Key: String; CanCreate: Boolean) : boolean;
```

Description

`Key` is the name of the key to open. `CanCreate` specifies whether to create the specified key if it does not exist. If `CanCreate` is `True`, the key is created if necessary. `ibec_reg_OpenKey` returns `True` if the key is successfully opened or created.

See also:
Example of ibec_reg_xxx functions: daily backup User Database
IBExpert After Start Script

ibec_reg_CloseKey

Syntax

```
function ibec_reg_CloseKey(Registry : variant) : variant;
```

Description

Call `ibec_reg_CloseKey` to write the current key to the registry and close the key.

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_DeleteKey

Removes a specified key and its associated data from the registry.

Syntax

```
function ibec_reg_DeleteKey(Registry : variant; Key: String) : boolean;
```

Description

Call `ibec_reg_DeleteKey` to remove a specified key and its associated data, if any, from the registry. `ibec_reg_DeleteKey` returns `True` if key deletion is successful. On error, `ibec_reg_DeleteKey` returns `False`.

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_CreateKey

Creates a new key in the registry.

Syntax

```
function ibec_reg_CreateKey(Registry : variant; Key: String) : boolean;
```

Description

Use `ibec_reg_CreateKey` to add a new key to the registry.

`Key` is the name of the key to create. `Key` can be an absolute or relative name. An absolute key begins with a backslash (\) and is a sub-key of the root key. A relative key is a sub-key of the current key.

`ibec_reg_CreateKey` returns `True` if the key creation is successful. On error, an exception is raised. Attempting to create a key that already exists has no effect.

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_WriteString

Writes strings to the Windows Registry.

Syntax

```
function ibec_reg_WriteString(Registry : variant; Name, Value: string) : variant;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_ReadString

Reads strings from the Windows Registry.

Syntax

```
ibec_reg_ReadString(Registry : variant; Key: String) : string;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database
IBExpert After Start Script

ibec_reg_WriteBool

Writes data to the Windows Registry.

Syntax

```
ibec_reg_WriteBool(Registry : variant; Name: String; Value: boolean) : variant;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database
Boolean datatype

ibec_reg_ReadBool

Reads data from the Windows Registry.

Syntax

```
ibec_reg_ReadBool(Registry : variant; Key: String) : boolean;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database
Boolean datatype

ibec_reg_WriteDate

Writes the date to the Windows Registry.

Syntax

```
ibec_reg_WriteDate(Registry : variant; Name: String; Value: date) : variant;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_ReadDate

Reads the date from the Windows Registry.

Syntax

```
ibec_reg_ReadDate(Registry : variant; Key: String) : date;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_WriteDateTime

Writes date and time to the Windows Registry.

Syntax

```
ibec_reg_WriteDateTime(Registry : variant; Name: String; Value: timestamp) : variant;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_ReadDateTime

Reads date and time from the Windows Registry.

Syntax

```
ibec_reg_ReadDateTime(Registry : variant; Key: String) : timestamp;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_WriteTime

Writes the time to the Windows Registry.

Syntax

```
ibec_reg_WriteTime(Registry : variant; Name: String; Value: time) : variant;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_ReadTime

Reads the time from the Windows Registry.

Syntax

```
ibec_reg_ReadTime(Registry : variant; Key: String) : time;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_WriteInteger

Writes data to the Windows Registry.

Syntax

```
ibec_reg_WriteInteger(Registry : variant; Name: String; Value: integer) : variant;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_ReadInteger

Reads data from the Windows Registry.

Syntax

```
ibec_reg_ReadInteger(Registry : variant; Key: String) : integer;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_WriteFloat

Writes data to the Windows Registry.

Syntax

```
ibec_reg_WriteFloat(Registry : variant; Name: String; Value: double precision) : variant;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

ibec_reg_ReadFloat

Reads data from the Windows Registry.

Syntax

```
ibec_reg_ReadFloat(Registry : variant; Key: String) : double precision;
```

See also:
Example of ibec_reg_xxx functions: daily backup User Database

Functions to handle regular expressions

The following functions are available in IBEBlock to handle work with regular expressions:

Function	Description
ibec_re_Create	
ibec_re_Free	
ibec_re_Exec	
ibec_re_ExecNext	
ibec_re_Match	

ibec_re_SetExpression	
ibec_re_Replace	
ibec_preg_Match	Searches `Subject` for a match to the regular expression given in `Pattern`.
ibec_preg_Replace	Searches `Subject` for matches to `Pattern` and replaces them with `Replacement`.

ibec_re_Create

Syntax

```
function ibec_re_Create(Expression : string) : variant;
```

See also:
Example: Retrieve all valid e-mail addresses from an input text

ibec_re_Free

Syntax

```
function ibec_re_Free(RegExp : variant) : variant;
```

See also:
Example: Retrieve all valid e-mail addresses from an input text

ibec_re_Exec

Syntax

```
function ibec_re_Exec(RegExp : variant; InputString : string) : boolean;
```

See also:
Example: Retrieve all valid e-mail addresses from an input text

ibec_re_ExecNext

Syntax

```
function ibec_re_ExecNext(RegExp : variant) : boolean;
```

See also:
Example: Retrieve all valid e-mail addresses from an input text

ibec_re_Match

Syntax

```
function ibec_re_Match(RegExp : variant; Index : integer) : string;
```

See also:
Example: Retrieve all valid e-mail addresses from an input text

ibec_re_SetExpression

Syntax

```
function ibec_re_SetExpression(RegExp : variant; Expression : string) : boolean;
```

ibec_re_Replace

Syntax

```
function ibec_re_Replace(RegExp : variant; InputStr : string; ReplaceStr : string) : string;
```

ibec_preg_Match

Syntax

```
function ibec_preg_Match(Pattern : string; Subject : string [; Matches : array of variant) : boolean;
```

Description

The `ibec_preg_Match` function searches `Subject` for a match to the regular expression given in `Pattern`.

It returns `TRUE` if a match for `Pattern` was found in the `Subject` string, or `FALSE` if no match was found or an error occurred.

If `Matches` is specified, then it is filled with the results of the search.

Example

The following example returns a list of all e-mail addresses used in a text file:

```
execute ibeblock     returns (
  Email varchar(200))
as
begin
  s = ibec_LoadFromFile('C:\SomeData.txt');
  sPattern = '([_a-zA-Z\d\-\.]+@[_a-zA-Z\d\-]+(\.[_a-zA-Z\d\-]+)+)';
  ibec_preg_match(sPattern, s, aEmails);
  foreach (aEmails as Email skip nulls) do
    suspend;
end
```

To learn more about the syntax of regular expressions available in IBExpert, please refer to Regular Expressions explained.

ibec_preg_Replace

Syntax

```
function ibec_preg_Replace(Pattern : string; Replacement : string; Subject : string) : string;
```

The `ibec_preg_Replace` function searches `Subject` for matches to `Pattern` and replaces them with `Replacement`. If matches are found, the new `Subject` will be returned, otherwise `Subject` will be returned unchanged.

Example

The following example removes all IB comments (/*...*/) from a text:

```
   execute ibeblock
as
begin
  s = ibec_LoadFromFile('C:\SomeScript.sql');
  sPattern = '/\*/*([^/]*\*+)+/';
  s = ibec_preg_replace(sPattern, '', s);
  ibec_SaveToFile('C:\ScriptNoComments.sql', s, __stfOverwrite);
end
```

To learn more about the syntax of regular expressions available in IBExpert, please refer to Regular Expressions explained.

Functions for working with POP3 servers

The following functions are implemented to work with pop3-servers.

ibec_pop3_OpenSession	Creates and initializes an internal object which is used to work with the pop3 protocol.
ibec_pop3_CloseSession	Destroys a POP3 object created with the ibec_pop3_OpenSession function.
ibec_pop3_Connect	Tries to establish a connection to the POP3 server.
ibec_pop3_User	Passes the user name specified for the POP3Session to the server.
ibec_pop3_Pass	Performs the POP3 PASS command specified for the POP3Session, passing to the server.
ibec_pop3_ConnectAndAuth	Performs the connection and POP3 USER and PASS commands one by one.
ibec_pop3_List	Performs the POP3 LIST command, retrieving a string with numbers and sizes (in bytes) of all of the messages available on a POP3 server.
ibec_pop3_List	Performs the POP3 UIDL command, retrieving a string with numbers and unique identifiers of all of the messages available on a POP3 server.
ibec_pop3_List	Performs the POP3 RETR command, retrieving a string with the entire text (including header) of the message specified with MessageNumber.
ibec_pop3_Dele	This marks the message specified with MessageNumber as deleted.
ibec_pop3_Quit	This deletes all messages marked as deleted and disconnects from the POP3 mail server.
ibec_pop3_GetProperty	Returns a value of the specified property.
ibec_pop3_SetProperty	Sets a value of the specified property.

ibec_pop3_OpenSession

Description

`ibec_pop3_OpenSession` creates and initializes an internal object which is used to work with the POP3 protocol.

Syntax

```
function ibec_pop3_OpenSession(Params : string) : variant;
```

The following parameters are available:

`Host=<string>`	POP3 server name.
`UserName=<string>`	User name.
`Password=<string>`	Password.
`Port=<string>`	POP3 port number. Default value is 25.

`ibec_pop3_OpenSession` returns a handle of a POP3 object.

See also:
Example of working with POP3 servers

ibec_pop3_CloseSession

Description

`ibec_pop3_CloseSession` destroys a POP3 object created with the `ibec_pop3_OpenSession` function.

Syntax

```
function ibec_pop3_CloseSession(POP3Session : variant) : variant;
```

See also:
Example of working with POP3 servers

ibec_pop3_Connect

Description

`ibec_pop3_Connect` function tries to establish a connection to the POP3 server. It returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_Connect(POP3Session : variant) : variant;
```

See also:
Example of working with POP3 servers

ibec_pop3_User

Description

`ibec_pop3_User` performs the POP3 `USER` command, passing user name, specified for the `POP3Session`, to the server. It returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_User(POP3Session : variant) : variant;
```

See also:
Example of working with POP3 servers

ibec_pop3_Pass

Description

`ibec_pop3_Pass` performs the POP3 `PASS` command specified for the `POP3Session`, passing to the server. It returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_Pass(POP3Session : variant) : variant;
```

See also:
Example of working with POP3 servers

ibec_pop3_ConnectAndAuth

Description

`ibec_pop3_ConnectAndAuth` performs the connection and POP3 `USER` and `PASS` commands one by one. It returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_ConnectAndAuth(POP3Session : variant) : variant;
```

See also:
Example of working with POP3 servers

ibec_pop3_List

Description

`ibec_pop3_List` performs the POP3 `LIST` command, retrieving a string with numbers and sizes (in bytes) of all of the messages available on a POP3 server. It returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_List(POP3Session : variant) : variant;
```

You can get a list of messages using `ibec_pop3_GetProperty` function.

See also:
Example of working with POP3 servers

ibec_pop3_Uidl

Description

`ibec_pop3_Uidl` performs POP3 `UIDL` command, retrieving a string with numbers and unique identifiers of all of the messages available on a POP3 server. It returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_Uidl(POP3Session : variant) : variant;
```

You can get a list of unique identifiers using `ibec_pop3_GetProperty` function.

See also:
Example of working with POP3 servers

ibec_pop3_Retr

Description

`ibec_pop3_Retr` performs the POP3 `RETR` command, retrieving a string with the entire text (including header) of the message specified with `MessageNumber`. `ibec_pop3_Retr` returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_Retr(POP3Session : variant; MessageNumber : integer) : variant;
```

After successfull execution you can get the message data using `ibec_pop3_GetProperty` function.

See also:
Example of working with POP3 servers

ibec_pop3_Dele

Description

`ibec_pop3_Dele` performs POP3 `DELE` command. This marks the message specified with `MessageNumber` as deleted. `ibec_pop3_Dele` returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_Dele(POP3Session : variant; MessageNumber : integer) : variant;
```

See also:
Example of working with POP3 servers

ibec_pop3_Quit

Description

`ibec_pop3_Quit` performs POP3 `QUIT` command. This deletes all messages marked as deleted and disconnects from the POP3 mail server. It returns `TRUE` if succeeded, otherwise it returns `FALSE`.

Syntax

```
function ibec_pop3_Quit(POP3Session : variant) : variant;
```

See also:
Example of working with POP3 servers

ibec_pop3_GetProperty

Description

`ibec_pop3_GetProperty` returns a value of the specified property.

Syntax

```
function ibec_pop3_GetProperty(POP3Session : variant; PropertyName : string) : variant;
```

The following properties are supported:

`Host`	POP3 server name
`UserName`	User name.
`Password`	Password.
`Port`	POP3 server port number.
`MsgData`	Text of message retrieved with `ibec_pop3_Retr` function.
`MessageData`	Same as `MsgData`.
`Uidl`	List of unique identifiers retrieved with `ibec_pop3_Uidl` function.
`List`	List of numbers and sizes of messages retrieved with `ibec_pop3_List` function.
`LastResponse`	A text string of last server response.

See also:
Example of working with POP3 servers

ibec_pop3_SetProperty

Description

`ibec_pop3_SetProperty` sets a value of the specified property.

Syntax

```
function ibec_pop3_SetProperty(POP3Session : variant; PropertyName : string; Value : variant) : variant;
```

The following properties are supported:

`Host`	POP3 server name.
`UserName`	User name.
`Password`	Password.
`Port`	POP3 server port number.

See also:
Example of working with POP3 servers

Exception-handling functions

Exception-handling functions are used with the `TRY...EXCEPT` statement or the `RAISE` statement.

If an exception is raised during execution of the initial statements list, the control passes to the first statement in the `exceptionBlock`. Here you can handle any exceptions which may occur using the following functions:

Function	Description
`function ibec_err_Message()`	Returns an exception message.
`function ibec_err_SQLCode()`	Returns the SQLCode of an exception if there was an SQL error.
`function ibec_err_Name()`	Returns an exception name.

See also:
`EXCEPTION`
`TRY ... FINALLY`
`TRY ... EXCEPT`

function ibec_err_Message()

Returns an exception message.

Examples can be found at the links below.

See also:
`EXCEPTION`
`TRY ... FINALLY`
`TRY ... EXCEPT`

function ibec_err_SQLCode()

Returns the SQLCode of an exception if there was an SQL error.

Examples can be found at the links below.

See also:
`EXCEPTION`
`TRY ... FINALLY`
`TRY ... EXCEPT`

function ibec_err_Name()

Returns an exception name.

Examples can be found at the links below.

See also:
`EXCEPTION`
`TRY ... FINALLY`
`TRY ... EXCEPT`

Cursor functions

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_CloseCursor

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_Eof

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_Fetch

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_FieldCount

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_FieldName

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_FieldValue

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_Next

... coming soon.

See also:
Data Comparer using cursors

ibec_cr_OpenCursor

... coming soon.

See also:
Data Comparer using cursors

User Form functions

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_CloseForm

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_CreateForm

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_ExecScript

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_FreeForm

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_GetElementAttribute

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_GetElementAttributeDef

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_GetFormData

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_SetElementAttribute

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_SetFormData

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

ibec_uf_ShowForm

... coming soon.

See also:
TableDDL.ibeblock
RunMe.ibeblock

Miscellaneous functions

The following miscellaneous functions are available in IBEBlock:

Function	Description
ibec_BuildCube	Builds an OLAP cube using a specified SELECT statement.
ibec_Chr	Returns the character for a specified ASCII value.
ibec_CmpRecords	Compares two arrays of variants (records).
ibec_CmpVals	Compares two values.
ibec_CompressFile	Allows you to create archives of files and extract them using the ibec_DecompressFile function.
ibec_CompressVar	Compresses VALUE using the LZ77 algorithm.
ibec_CreateModelScript	Creates an SQL script from specified database model file.
ibec_CreateReport	Prepares a report from a specified source (FastReport) and returns prepared report data.
ibec_DecompressFile	Allows you to extract files from archives from files compressed using the ibec_CompressFile function.
ibec_DecompressVar	Decompresses VALUE preciously compressed with ibec_CompressVar.
ibec_DisableFeature	Use this feature to disable all IBExpert menu items
ibec_EnableFeature	Use this feature to blend in only those menu items which you wish the user to see.
ibec_EncodeDate and ibec_DecodeDate	New to IBExpert version 2005.09.25. These functions are similar to the Delphi EncodeDate and DecodeDate functions.
ibec_Exec	Runs a specified application.
ibec_ExecSQLScript	Executes an SQL script from a variable or a file.
ibec_ExportReport	Exports a prepared report, created with the ibec_CreateReport function, into a specified format.
ibec_FormatIdent	Creates a string representation of a GUID.
ibec_FreeGlobalVar	Removes a specified variable from a list of global variables, and frees memory associated with the variable.
ibec_GetGlobalVar	Returns the value of a specified global variable.
ibec_GetIBEVersion	Returns a string representation of the IBExpert/IBEScript version.
ibec_GetTickCount	Retrieves the number of milliseconds that have elapsed since Windows was started.
ibec_GetViewRecreateScript	Creates a *Recreate* script for a specified view(s) and returns it as a result.
ibec_GUID	Creates a string representation of a GUID, a unique 128-bit integer used for CLSIDs and interface identifiers.
ibec_High	Returns the highest value within the range of the index type of the array.
ibec_IIF	Tests a condition and returns Value1 if the condition is True and Value2 if the condition is False.
ibec_IntToHex	Returns the hex representation of an integer.
ibec_MessageDlg	Displays a message dialog box in the center of the screen.
ibec_Ord	Returns the ordinal value of the specified character.
ibec_ParseCSVLine	
ibec_Progress	Displays a progress message.
ibec_Random	Generates random numbers within a specified range.
ibec_Random2	Generates random numbers within a specified range.
ibec_RandomChar	Generates random char within a specified range.
ibec_RandomString	Returns a random string.
ibec_RandomVal	
ibec_SetGlobalVar	Allows you to create/modify a global variable.
ibec_SetLength	Sets the length of a dynamic-array variable.
ibec_ShiftRecord	
ibec_smtp_SendMail	Sends an email using the SMTP protocol.
ibec_WaitForEvent	Monitors events sent by the POST_EVENT command.

ibec_BuildCube

See also:
Building an OLAP cube

ibec_Chr

Returns the character for a specified ASCII value.

Syntax

```
function ibec_Chr(X : integer): string;
```

Description

`ibec_Chr` returns the character with the ordinal value(ASCII value) of the byte-type expression, x.

Example

```
execute IBEBlock
returns (cout varchar(1))
as
begin
  i = 0;
  while (i < 256) do
  begin
    cout = ibec_Chr(i);
    i = i + 1;
    suspend;
  end
end
```

See also:
`ibec_Ord`

ibec_CmpRecords

Compares two arrays of variants (records).

Syntax

```
function ibec_CmpRecords(Record1, Record2 : array of variants): variant;
```

Example

```
execute ibeblock
returns (iresult integer)
as
begin
  Val1[0] = 1; Val1[1] = 'ABC'; Val1[2] = 25.67;
  Val2[0] = 1; Val2[1] = 'ABC'; Val2[2] = 25.67;
  iresult = ibec_CmpRecords(Val1, Val2); /* iresult = 0 */
  suspend;

  Val2[2] = 15.43;
  iresult = ibec_CmpRecords(Val1, Val2); /* iresult = 2 */
  suspend;

  Val2[3] = 0;
  iresult = ibec_CmpRecords(Val1, Val2); /* iresult = NULL */
  suspend;
end
```

See also:
ibec_CmpVals

ibec_CmpVals

Compares two values.

Syntax

```
function ibec_CmpVals(Value1, Value2 : variant): variant;
```

Description

The `ibec_CmpVals` compares `Value1` and `Value2` and returns `0` if they are equal.

If `Value1` is greater than `Value2`, `ibec_CmpVals` returns `1`.

If `Value1` is less than `Value2`, `ibec_CmpVals` returns `-1`.

If it is impossible to compare values the function returns `NULL`.

Example

```
execute IBEBlock
returns (iresult integer)
as
begin
  iresult = ibec_CmpVals(25, '25');
  suspend; /* Values are equal, iresult = 0 */

  iresult = ibec_CmpVals('25', 40);
  suspend; /* 25 is less then 40, iresult = -1 */

  iresult = ibec_CmpVals('ABC', 'abc');
  suspend; /* 'ABC' is less then 'abc', iresult = -1 */

  iresult = ibec_CmpVals(NULL, '25');
  suspend; /* NULL is less than any other value, iresult = -1 */

  iresult = ibec_CmpVals('25', NULL);
  suspend; /* Any value is greater than NULL, iresult = 1 */

  iresult = ibec_CmpVals(NULL, NULL);
  suspend; /* NULL is equal to NULL!!!, iresult = 0 */

  iresult = ibec_CmpVals('ABC', 25);
  suspend; /* Impossible to compare, iresult = NULL */

  iresult = ibec_CmpVals('24.56', 24.56);
  suspend; /* Values are equal, iresult = 0 */
end
```

ibec_CompressFile

This function allows you to create archives of files and extract them using the `ibec_DecompressFile` function. Archives currently supported by the `ibec_CompressFile` function include the following formats:

`ZIP`, `BZIP`, `GZIP`, `JAR`, `LHA`, `CAB`, `TAR`, `BlackHole`.

Syntax

```
function ibec_CompressFile(FileSpec : string; ExcludeFileSpec : string; ArcType : integer;
        ArcName : string; Options : string; CallbackBlock : string) : variant;
```

The `ibec_CompressFile` currently returns `NULL`.

Parameters

`FileSpec`	A filter to retrieve specific file(s) or a range of files. Wildcard characters (asterisk (`*`)and question mark (`?`)) are supported. It can include directory names. Items within `FileSpec` must be delimited with commas.
`ExcludeFileSpec`	Defines specific file names or a range of file names (using wildcards) to exclude from being compressed. This parameter has precedence over the `FileSpec` param. For example, if `FileSpec` contains a file named `file.txt`, and `ExcludeFileSpec` contains a wild card such as `*.txt`, the value of `ExcludeFileSpec` overrides the value of `FileSpec` and the file will not be compressed. Items within `ExcludeFileSpec` must be delimited with commas.
`ArcType`	Type of archive. Possible values are: `__atBlackHole`, `__atBZip`, `__atCab`, `__atGZip`, `__atJar`, `__atLha`, `__atZip`.
`ArcName`	Defines the file name of the archive to be created or an existing archive to which files are to be added.
`Options`	List of additional options, must be separated with semicolon. Possible options are: `CompressMethod`, `DeflateType`, `Password`, `StoredDirNames`, `StoreEmptySubdirs`, `StoreFilesOfType`, `Action`, `PartSize`, `DateAttribute`, `RecurseDirs`. See detailed description of each option below.
`CallbackBlock`	A call-back [IBEBlock EXECUTE IBEBLOCK |IBEBlock]] which will be executed for some events during the compression process. The call-back IBEBlock must have at least one input parameter, which will be used to pass an array of event values. If there is no call-back block use `NULL` or an empty string as a value of this parameter.

Description of possible options

```
CompressMethod= Store | Deflate | Fuse | Frozen5 | Frozen6 | MsZip | LZX | Quantum |
                Tarred | TarGZip | TarBZip | BZip2
```

The default method (if the `CompressMethod` option is omitted) for each archive type is:

```
__atBlackHole: Fuse
__atZip: Deflate
__atLha: Frozen6
__atCab: MsZip
__atGZip: Deflate
__atTar: Tarred
```

The following is the listing of the value of `CompressMethod` for each archive type:

```
__atZip:  [Store, Deflate]
__atBlackHole:   [Store, Fuse]
__atLha:   [Store, Frozen5, Frozen6]
__atCab:  [Store, MsZip, Lzx, Quantum]
__atGZip:  [Deflate ]
__atTar:  [Tarred, TarGZip, TarBZip]
```

```
DeflateType= Store | Fast | Normal | Best
```

This defines the setting for archive types which use the `Deflate` compression method. The default setting is `NORMAL`. Different settings either increase compression speed but reduce compression ratios, or increase ratios but decrease speed.

`Password= <password>` : Use the Password option to add encrypted files to a `ZIP` and `lackHole` archives or extract encrypted files from ones. If the value of this property is not blank, the value will be used as the password for encryption/decryption.

```
StoredDirNames= None | Absolute | AbsoluteNoDrv | AbsoluteNoRoot | Relative | RelativeStoreStart | ExplorerAuto
```

Use this option to set how directories are to be stored in an archive. The default setting is `AbsoluteNoDrv`.

Examples

```
USING UNC-Pathnames
FileSpec = '//Server/Group11/Emp4129/*.txt';
StoredDirNames   Saved in archive as:
--------------------------------------------------
```

```
                None                        proj1.txt
                Absolute                    //Server/Group11/Emp4129/proj1.txt
                AbsoluteNoDrv               /Emp4129/proj1.txt
                Relative                    proj1.txt (subdirs = dir/*.txt)
                RelativeStoreStart          Emp4129/proj1.txt (subdirs =  mp4129/dir/proj1.txt)
                ExplorerAuto                proj1.txt (subdirs = dir/*.txt)
                Using local drives
                FileSpec := 'f:\ZipTV\Project1\proj1.exe';
                StoredDirNames:   Saved in archive as:
                --------------------------------------------------
                None                        proj1.txt
                Absolute                    f:\ZipTV\Project1\proj1.txt
                AbsoluteNoDrv               \ZipTV\Project1\proj1.txt
                Relative                    proj1.txt (subdirs = dir\proj1.txt)
                RelativeStoreStart          project1\proj1.txt (subdirs = Project1\dir\*.txt)
                ExplorerAuto                proj1.txt (subdirs = dir\proj1.txt)
StoreEmptySubDirs= TRUE | FALSE
```

When the value of this option is `True`, empty sub-directories names are stored to the archive. The default setting is `True`.

```
StoreFilesOfType=<list_of_file_extensions>
```

This property contains a listing of file extensions delimited with commas or spaces. Any file whose extension is contained within this list will not be compressed, but stored within the archive during compression.

The default value is `.LZH`, `.PAK`, `.PK3`, `.PK_`, `.RAR`, `.TAR`, `.TGZ`, `.UUE`, `.UU`, `.WAR`, `.XXE`, `.Z`, `.ZIP`, `.ZOO`.

Note: The extension separator (dot) character for each extension in the list is mandatory!

```
Action= Add | Move | Delete | Read
```

This option defines what action the `ibec_CompressFile` function is to perform.

Possible values are:

`Add`	Adds files to an archive if they are found not to already exist in the archive. If they do already exist, a comparison of files date stamp with the date stored in the archive is made. If the date stamps do not match, the file is recompressed, otherwise it is skipped. If the desired `CompressMethod` of compression is different than the method previously used to compress the file, then the file is recompressed, otherwise it is skipped. If the archive doesn't already exist, it is created and all files matching `FileSpec` are compressed and added to the archive.
`Move`	Follows the same convention as `Add`, but deletes all files on disk that were added to the archive. Files are deleted only after a successful `ADD`.
`Delete`	Deletes all files matching `FileSpec` from an existing archive.
`Read`	Reserved for future use. The default value for this option is `Add`.

`PartSize=<int_value>[KB|K|MB|M|B]` - Use this option to specify the file-size of the output volumes for a multi-volume ZIP archive. For example, `PartSize=100MB`. `PartSize` value must be at least 65 KB.

`DateAttribute= FileDate | SysDate | MaxFileDate` - Use this option to define a file's date when extracted to disk. Use this property to define a file's date to be stored into an archive. Possible values:

- `FileDate` - set the extracted file's date using the date stored in the archive; store the date using the disk file's date being compressed.
- `SysDate` - set the extracted file's date using the systems date/time; store the date using the current system's date.
- `MaxFileDate` - set the extracted file's date using the date from the newest file in the archive; store the date using the newest file's date matching `FileSpec`.

`RecurseDirs - TRUE | FALSE` - Use this option to recurse sub-directories for files matching the `FileSpec` parameter. The default value is `False`.

`OverwriteMode - Skip | Overwrite` - Use the `OverwriteMode` property to either skip or overwrite files that already exist on disk. The default value is `Skip`.

`ConfirmOverwrites - TRUE | FALSE` - The `ConfirmOverwrites` option is directly related to the `OverwriteMode` option. The default value is `False`. This option is reserved for future use, please don't change it yet!

`RestoreFileAttributes - TRUE | FALSE` - When `True`, this sets an extracted file's attribute to the setting stored in the compressed header for that file. The default value is `True`.

`UseStoredDirs - TRUE | FALSE` - When set to `False`, this uses the current directory to extract files into, if the `TargetDir` property is blank. When set to `True`, the default is the current directory information existing in regard to the internal compressed file. The default value is `False`.

Example

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (Vals variant)
         as
         begin
           EventName = Vals[EVENT];
           Action = Vals[ACTION];
           File name = Vals[FILE NAME];
           if (Action = COMPRESS) then
```

```
                sPref = Adding ;
              else
                sPref = Extracting ;
              if (EventName = FILEBEGIN) then
                ibec_Progress(sPref + File name + ...);
              else if (EventName = PROGRESS) then
              begin
                iBytes = Vals[BYFILE];
                if (ibec_Mod(iBytes, 5) = 0) then
                  ibec_Progress(sPref + File name + ...  + ibec_Cast(iBytes, __typeString) +

              %);
              end;
            end';
             ibec_DecodeDate(ibec_Now(), iYear, iMonth, iDay);
        ArcName = 'E:\IBE_' + ibec_Cast(iYear, __typeString) + '_' + ibec_Cast(iMonth, __typeString)

 +
                '_' + ibec_Cast(iDay, __typeString) + '.zip';
             if (ibec_FileExists(ArcName)) then
        begin
          ibec_ShowMessage('Nothing to do.');
          Exit;
        end;


             -- Compressing
        CompressOptions = 'CompressMethod=Deflate; RecurseDirs=Yes; DeflateType=Best;

 StoredDirs=AbsoluteNoRoot';
        FileSpec = 'D:\MyProjects\IBExpert\*.*, D:\MyProjects\IBEScript\*.*, D:\MyComponents\*.*';
        ExcludeSpec = '*.dcu, *.~*, *.bak';
        MyVar = ibec_CompressFile(FileSpec, ExcludeSpec, __atZip, ArcName, CompressOptions, cbb);
      end;
```

See also:
ibec_DecompressFile

ibec_CompressVar

`ibec_CompressVar` compresses `Value` using the `LZ77` algorithm.

Syntax

```
function ibec_CompressVar(Value : variant; Options : string) : string;
```

Description

`ibec_CompressVar` compresses Value using the `LZ77` algorithm and returns the string that represents a compressed content of `Value`.

Parameters

The `Options` parameter is reserved for future use.

Example

```
execute ibeblock
as
begin
  -- Compressing
  MyVar = ibec_LoadFromFile('D:\Script.sql');
  MyVar = ibec_CompressVar(MyVar, '');
  -- Decompressing
  MyVar = ibec_DeCompressVar(MyVar, '');
  ibec_SaveToFile('D:\Script.copy.sql', MyVar, __stfOverwrite);
end
```

See also:
`ibec_DecompressVar`

ibec_CreateModelScript

Creates an SQL script from a specified *Database Model* file.

Syntax

```
function ibec_CreateModelScript(ModelFileName : string; ScriptFileName : string; Options : cardinal):
```

integer;

Example

```
execute ibeblock
as
begin
  ibec_create_model_script('C:\npfe_1.grc', 'C:\npfe_1.sql', __msoDontQuoteIdents +

__msoIncludeDescriptions);
end
```

See also:
Example: Creating a script from a Database Designer model file

ibec_CreateReport

Prepares a report from a specified source (FastReport) and returns prepared report data.

Syntax

```
function ibec_CreateReport(ReportSource : string; Params : array of variant; Options : string) : variant;
```

Description

`ibec_CreateReport` prepares a report from a specified source (FastReport) and returns prepared report data. For preparing the intial report please refer to the IBExpert Report Manager.

This feature can be used for executing reports created with the IBExpert Report Manager in command-line mode, for example with batch files. The monthly sales report, invoices or other such reports can be designed in the Report Manager and executed with simple SQL statements. The result can then be saved in the database as a pdf file or other formats and sent by e-mail, exporting using `ibec_ExportReport`.

Example

```
execute ibeblock
as
begin
  Params['HeaderMemo'] = '';
  Params['MEMO2'] = 2;

  select ibe$report_source from ibe$reports
  where ibe$report_id = 4
  into :RepSrc;

  Report = ibec_CreateReport(RepSrc, Params, null);
  ibec_SaveToFile('D:\reptest.fp3', Report, 0);
end
```

See also:
Report Manager
`ibec_ExportReport`

ibec_DecompressFile

Description

This function allows you to extract files from archives from files compressed using the ibec_CompressFile function.

Archives currently supported by ibec_DecompressFile function include the following formats:

ZIP, ZIP SFX, ZOO, ZOO SFX, RAR, ARJ, ARJ SFX, ARC, ARC SFX, ACE, CAB, HA, JAR (JavaSoft java format), LHA, LHA SFX, LZH, LZH SFX, PAK, PAK SFX, TAR, GZIP, Z, BH, BH SFX.

Syntax

```
function ibec_DecompressFile(ArcName : string; FileSpec : string; ExcludeFileSpec : string;
                             TargetDir : string; Options : string; CallbackBlock : string) : variant;
```

ibec_DecompressFile returns the number of extracted files if there were no errors. Otherwise it returns NULL.

Parameters

ArcName	Defines the file name of the archive from which to extract files.
FileSpec	See description of corresponding parameter for ibec_CompressFile.
ExcludeFileSpec	See description of corresponding parameter for ibec_CompressFile.
TargetDir	Defines the directory in which the files from an archive are to be extracted. If this parameter does not contain a blank string, then the UseStoredDirs option is automatically set to False. To extract files into original directories, this property must be blank and the UseStoredDirs option set to True.
Options	List of additional options, which must be separated with semicolon. Possible options are: Password, UseStoredDirs, DateAttribute, RecurseDirs, onfirmOverwrites, OverwriteMode, RestoreFileAttributes. See detailed description of each option below.
CallbackBlock	A call-back IBEBlock which will be executed for some events during the decompression process. The call-back IBEBlock must have at least one input parameter, which will be used to pass array of event values. If there is no call-back block use NULL or an empty string as a value of this parameter.

Description of possible options

Password= <password>	Use the password option to add encrypted files to a ZIP and BlackHole archives or extract encrypted files from ones. If the value of this property is not blank, the value will be used as the password for encryption/decryption.
DateAttribute= FileDate | SysDate | MaxFileDate	Use this option to define a file's date when extracted to disk. Use this property to define a file's date to be stored into an archive. Possible values: * FileDate - set the extracted file's date using the date stored in the archive; store the date using the disk file's date being compressed. * SysDate - set the extracted file's date using the systems date/time; store the date using the current system's date. * MaxFileDate - set the extracted file's date using the date from the newest file in the archive; store the date using the newest file's date matching FileSpec.
RecurseDirs	TRUE | FALSE - Use this option to recurse sub-directories for files matching the FileSpec parameter. The default value is False.
OverwriteMode	Skip | Overwrite - Use the OverwriteMode property to either skip or overwrite files that already exist on disk. The default value is Skip.
ConfirmOverwrites	TRUE | FALSE - The ConfirmOverwrites option is directly related to the OverwriteMode option. The default value is False. This option is reserved for future use, please don't change it yet!
RestoreFileAttributes	TRUE | FALSE - When True, this sets an extracted file's attribute to the setting stored in the compressed header for that file. The default value is True.
UseStoredDirs	TRUE | FALSE - When set to False, this uses the current directory to extract files into, if the TargetDir property is blank. When set to True, the default is the current directory information existing in regard to the internal compressed file. The default value is False.

Example

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (Vals variant)
         as
         begin
           EventName = Vals[EVENT];
           Action = Vals[ACTION];
           File name = Vals[FILE NAME];
           if (Action = COMPRESS) then
             sPref = Adding ;
           else
             sPref = Extracting ;
           if (EventName = FILEBEGIN) then
             ibec_Progress(sPref + File name + ...);
           else if (EventName = PROGRESS) then
```

```
          begin
            iBytes = Vals[BYFILE];
            if (ibec_Mod(iBytes, 5) = 0) then
              ibec_Progress(sPref + File name + ...  + ibec_Cast(iBytes, __typeString) +

          %);
          end;
        end';
       ibec_DecodeDate(ibec_Now(), iYear, iMonth, iDay);
  ArcName = 'E:\IBE_' + ibec_Cast(iYear, __typeString) + '_' + ibec_Cast(iMonth, __typeString)

+
            '_' + ibec_Cast(iDay, __typeString) + '.zip';
       if (ibec_FileExists(ArcName)) then
  begin
    ibec_ShowMessage('Nothing to do.');
    Exit;
  end;


  -- Decompressing

  FileSpec = '*.*';
  ibec_ForceDirectories('E:\TestDecompress\');
  MyVar = ibec_DecompressFile(ArcName, FileSpec, , 'E:\TestDecompress\', , cbb);
end;
```

See also:
ibec_CompressFile

ibec_DecompressVar

Syntax

```
function ibec_DecompressVar(Value : variant; Options : string) : string;
```

Description

`ibec_DecompressVar` performs decompresion of `Value` (previously compressed with the `ibec_CompressVar` function) and returns the string that repreent the decompressed content of `Value`.

Parameters

The `Options` parameter is reserved for future use.

Example

```
execute ibeblock
as
begin
  -- Compressing
  MyVar = ibec_LoadFromFile('D:\Script.sql');
  MyVar = ibec_CompressVar(MyVar, '');
  -- Decompressing
  MyVar = ibec_DeCompressVar(MyVar, '');
  ibec_SaveToFile('D:\Script.copy.sql', MyVar, __stfOverwrite);
end
```

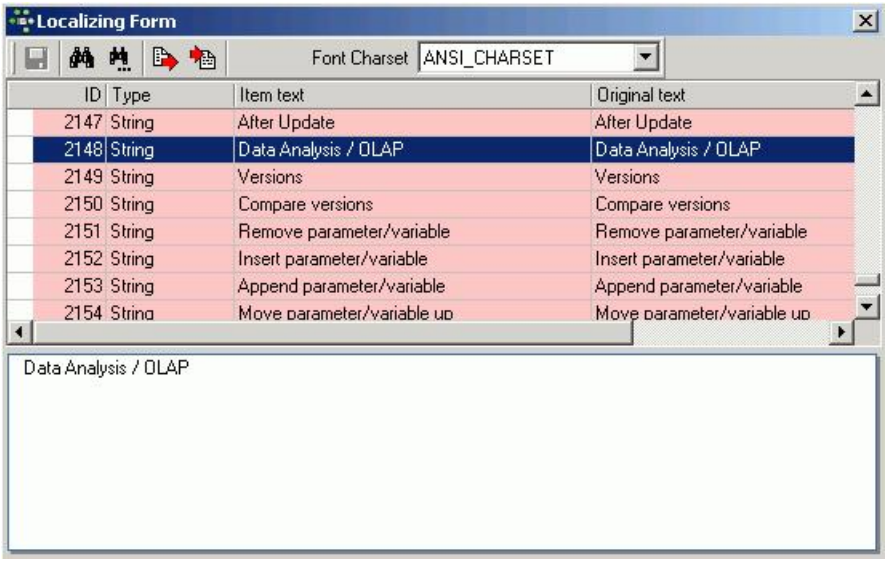
See also:
`ibec_CompressVar`

ibec_DisableFeature

Using this feature it is possible to disable all menu items, and then, using `ibec_EnableFeature`, to blend only those in which you wish the user to see. A particularly useful security feature!

```
execute ibeblock
as
begin
 ibec_DisableFeature(0);      --disable all
 ibec_EnableFeature(1003);  --enable Tools menu
 ibec_EnableFeature(2148);  --enable menuitem tools-data analysis
end
```

The example above enables only the IBExpert Tools menu item, Data Analysis. The numbers quoted directly after the IBEBlock keyword can be found in the IBExpert Tools menu, Localize IBExpert.



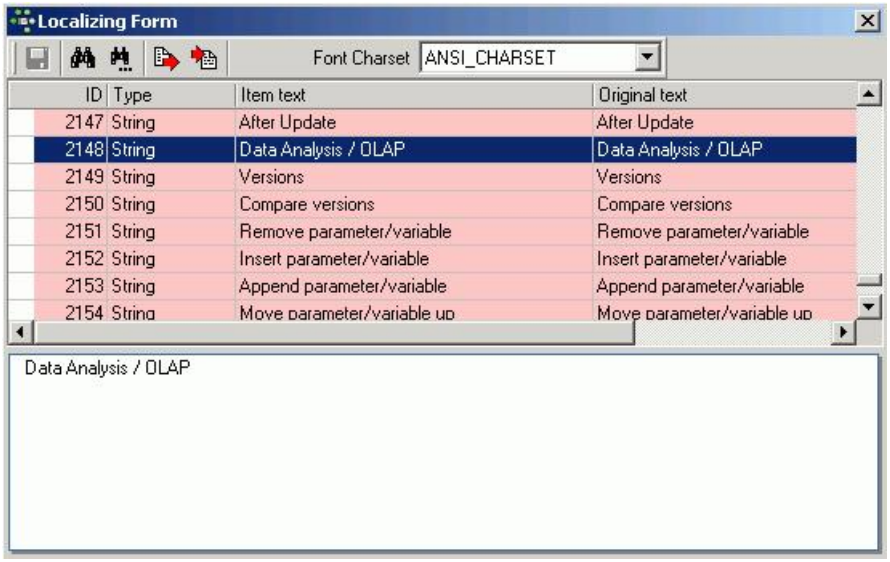
See also:
Example: Disable and enable IBExpert features

ibec_EnableFeature

Using this feature it is possible, after disabling all IBExpert menu items using `ibec_DisableFeature`, to blend in only those menu items which you wish the user to see. A particularly useful security feature!

```
execute ibeblock
as
begin
 ibec_DisableFeature(0);      --disable all
 ibec_EnableFeature(1003);  --enable Tools menu
 ibec_EnableFeature(2148);  --enable menuitem tools-data analysis
end
```

The example above enables only the IBExpert Tools menu item, Data Analysis. The numbers quoted directly after the IBEBlock keyword can be found in the IBExpert Tools menu, Localize IBExpert.



See also:
Example: Disable and enable IBExpert features

ibec_EncodeDate and ibec_DecodeDate

The functions, `ibec_EncodeDate` and `ibec_DecodeDate` are new to IBExpert version 2005.09.25. These functions are similar to the Delphi `EncodeDate` and `DecodeDate` functions.

ibec_Exec

Syntax

```
function ibec_Exec(CommandLine : string; Options : string;
CallbackBlock : string) : variant;
```

Description

The `ibec_Exec` function runs the specified application.

Parameters

`CommandLine`	The command line (filename plus optional parameters) for the application to be executed.
`Options`	String containing additional options delimited with semicolon; possible options are:
`OutFile=<file_name>`	Name of the file where the output of the application will be stored.
`ConvertToANSI`	If specified, the output will be translated from the OEM-defined character set into an ANSI string.
`CallbackBlock`	A callback IBEBlock which will be executed for each output line. The callback IBEBlock must have at least one input parameter, which will be used to pass an output line within it. If there is no callback block use `NULL` or an empty string as a value of this parameter.

Example

The following example uses the `ibec_Exec` function to restore a database from a backup copy using `GBAK.EXE`:

```
execute ibeblock
as
begin

  cbb = 'execute ibeblock (LogStr variant)
         as
         begin
           ibec_Progress(LogStr);
         end';

  res = ibec_Exec('C:\Program Files\Firebird\Bin\gbak.exe
                     -r -v -rep -user SYSDBA -pas masterkey
                     E:\test_db.fbk E:\test_db.fdb',
                  'OutFile=E:\Restore.log; ConvertToANSI',  cbb);

  if (res = 0) then
    ibec_ShowMessage('Restore process completed successfully');
  else
    ibec_ShowMessage('Restore process failed with exit code = '||res);
end
```

ibec_ExecSQLScript

Executes an SQL script from a variable or a file.

Syntax

```
function ibec_ExecSQLScript(Connection : variant; SQLScript : string; Options : string; ProgressBlock : variant) : variant;
```

SQLScript	script text or name of script file.
Options	additional options. There are two additional options currently available: ServerVersion and StopOnError.
ProgressBlock	an IBEBlock which will be executed for every progress message generated during script execution.

Description

ibec_ExecSQLScript executes an SQL script from a variable or a file.

Connection is an active connection created with the ibec_CreateConnection function which will be used while executing a script. If Connection is not specified (NULL) the script must contain the CREATE DATABASE or the CONNECT statement, otherwise an exception will be raised.

ibec_ExecSQLScript returns NULL if there were no errors while executing a script. Otherwise it returns an error(s) message.

Example

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (BlockData variant)
         as
         begin
           sMessage = BlockData;
           if (sMessage is not null) then
             ibec_Progress('SQL Script: ' + sMessage);
         end';

  db = ibec_CreateConnection(__ctFirebird, ...);
  try
    Scr = 'INSERT INTO MYTABLE (ID, DATA) VALUES (1, 'Bla-bla'); ' + 'INSERT INTO MYTABLE (ID, DATA) VALUES
   (2, 'Bla-bla'); '  + 'COMMIT;';
    ibec_ExecSQLScript(db, Scr, 'ServerVersion=FB21; StopOnError=FALSE', cbb); ...
    ibec_ExecSQLScript(db, 'D:\Scripts\CheckData.sql', 'ServerVersion=FB21', null); finally
    ibec_CloseConnection(db);
  end
end
```

ibec_ExportReport

Syntax

```
function ibec_ExportReport(PreparedReport : variant; FileName : string; ExportType : integer; Options : string) : boolean;
```

Description

`ibec_ExportReport` exports report, created with the IBExpert Report Manager and prepared using the `ibec_CreateReport` function, into a specified format.

The following export types are supported as value of the `ExportType` parameter:

```
__erPDF  (= 0)
__erTXT  (= 1)
__erCSV  (= 2)
__erHTML (= 3)
__erXLS  (= 4)
__erXML_XLS (= 5)
__erRTF  (= 6)
__erBMP  (= 7)
__erJPEG (= 8)
__erTIFF (= 9)
__erGIF  (= 10)
```

Options

The following additional export options are supported:

`Background=TRUE|FALSE`	Export of graphic image assigned to a page into result file. It considerably increases output file size. Applicable for PDF, HTML, XLS, XML export types. Default value is `FALSE`.
`Compressed=TRUE|FALSE`	Output file compressing. It reduces file size but increases export time. Applicable for PDF export. Default value is `TRUE`.
`EmbeddedFonts=TRUE|FALSE`	Applicable for PDF export type. All fonts used in report will be contained in the PDF output file for correct file displaying on computers where these fonts may be absent. Output file size increases considerably. Default value is `FALSE`.
`PrintOptimized=TRUE|FALSE`	Applicable for PDF export type. Output of graphic images in high resolution for further correct printing. This option enabling is necessary only when the document contains graphics and its printing is necessary. It considerably increases output file size. Default value is `FALSE`.
`EmptyLines=TRUE|FALSE`	Export of empty lines, applicable for `TXT` export. Default value is `FALSE`.
`Frames=TRUE|FALSE`	Export of text objects frames, applicable for `TXT` export. Default value is `FALSE`.
`OEMCodePage=TRUE|FALSE`	Resulting file OEM coding selecting. Applicable for `TXT` and `CSV` exports. Default value is `FALSE`.
`PageBreaks=TRUE|FALSE`	Export of page breaks to resulting file. Applicable for `TXT` export type. Default value is `TRUE`.
`Separator=<string>`	Values separator. Default value is semicolon (`;`). To avoid incorrect parsing of the options string double quote a separator value: Separator=","
`ExportStyles=TRUE|FALSE`	Transferring of text objects design styles. Disabling increases exporting but worsens document appearance. Applicable for HTML, XLS and XML documents. Default value is `TRUE`.
`ExportPictures=TRUE|FALSE`	Includes graphic images exporting possibility. Applicable for HTML, XLS and RTF documents. Default value is `TRUE`.
`Navigator=TRUE|FALSE`	Includes special navigator for fast navigation between pages. Applicable for HTML pages. Default value is `FALSE`.
`Multipage=TRUE|FALSE`	Every page of the report will be written to a separate file. Applicable for HTML documents. Default value is `FALSE`.
`AsText=TRUE|FALSE`	Applicable for XLS export type. All objects are transferred into table/diagram as text ones. This option may be useful when transferring numeric fields with complicated formatting. Default value is `FALSE`.
`MergeCells=TRUE|FALSE`	Applicable for XLS export type. Cells integration in resulting table/diagram for achieving maximum correspondence to the original. Disabling increases exporting but reduces document appearance. Default value is `TRUE`.
`Wysiwyg=TRUE|FALSE`	Full compliance to report appearance. Applicable for XML, XLS and RTF documents.
`CropImages=TRUE|FALSE`	After exporting blank area cropping will be performed along edges. Applicable for `BMP`, `JPEG`, `TIFF` and `GIF` export types. Default value is `FALSE`.
`Monochrome=TRUE|FALSE`	Monochrome picture creating. Applicable for `BMP`, `JPEG`, `TIFF` and `GIF` export types. Default value is `FALSE`.
`JPEGQuality=<integer>`	`JPEG` file compression ratio. Applicable for `JPEG` files. Default value is `90`.
`Quality=<integer>`	Same as `JPEG` quality.

Example

```
execute ibeblock
as
begin
  Params['HeaderMemo'] = '';
  Params['MEMO2'] = 2;

  SELECT IBE$REPORT_SOURCE FROM ibe$reports
  where ibe$report_id = 4
  into :RepSrc;

  Report = ibec_CreateReport(RepSrc, Params, null);
```

```
  ibec_SaveToFile('D:\reptest.fp3', Report, 0);
          Res = ibec_ExportReport(Report, 'D:\reptest.pdf', __erPDF, 'EmbeddedFonts=TRUE');
  Res = ibec_ExportReport(Report, 'D:\reptest.jpg', __erJPEG, 'CropImages; Quality=90');
end
```

See also:
Report Manager
`ibec_CreateReport`

ibec_FormatIdent

... coming soon.

See also:
Example: Recreating indices 1

ibec_FreeGlobalVar

Description

This function removes a specified variable from a list of global variables, and frees memory associated with the variable. If an empty string is specified as `VarName` all global variables will be destroyed. This function returns a number of destroyed global variables.

Syntax

```
function ibec_FreeGlobalVar
   (VarName : string) : variant;
```

(:keywords ibec_GetGlobalVar

ibec_GetGlobalVar

`ibec_GetGlobalVar` returns the value of a specified global variable. If the variable does not exist, this function returns the value passed in `DefaultValue`.

Syntax

```
function ibec_GetGlobalVar
   (VarName : string; DefaultValue : variant) :variant;
```

Example

The following example illustrates the use of this function, together with `ibec_SetGlobalVar`, described within an SQL script:

```
CONNECT ...;

execute ibeblock
as
begin
  select myfield from mytable
  where something = 25
  into :MyVar;
  ibec_SetGlobalVar('MyGlobalVar', MyVar);
end;

...

execute ibeblock
as
begin
  MyVar = ibec_GetGlobalVar('MyGlobalVar', null);
  if (MyVar = 1) then
    insert into mytable ...;
  else if (MyVar = 2) then
    update mytable set ...;
end;
```

ibec_GetIBEVersion

`ibec_GetIBEVersion` function was implemented in IBExpert version 2007.07.18. This function returns a string representation of the IBExpert/IBEScript version.

Syntax

```
function ibec_GetIBEVersion() : string;
```

ibec_GetTickCount

Retrieves the number of milliseconds that have elapsed since Windows was started.

Syntax

```
function ibec_GetTickCount : integer;
```

Example

```
execute IBEBlock
returns (cout varchar(100))
as
begin
  Time1 = ibec_GetTickCount();

  select * from rdb$fields as dataset ds;
  close dataset ds;

  Time2 = ibec_GetTickCount();
  cout = 'Time elapsed: ' || ((Time2 - Time1) / 1000) || ' seconds';
  suspend;
end
```

See also:
IBEBLOCK and Test Data Generator

ibec_ibec_GetViewRecreateScript

Creates a *Recreate* script for a specified view(s) and returns it as a result.

Syntax

```
function ibec_GetViewRecreateScript(Connection : variant; ViewName : string;
Options : string; ProgressBlock : variant) : string;
```

Connection	An active connection created with the ibec_CreateConnection function.
ViewName	List of names of view(s), delimited with semicolon or comma, for which a *Recreate* script will be created.
Options	List of options delimited with semicolon; possible options are:
GenerateCreate	Determines whether a CREATE DATABASE statement should be included at the beginning of the generated script.
GenerateConnect	Determines whether a CONNECT statement should be included at the beginning of the generated script.
IncludePassword	Determines whether the password should be included into the CREATE DATABASE or the CONNECT statement in the resulting SQL script.
SupressComments	Use to supress comments in the resulting script.
ExtractDescriptions	Determines whether database objects' descriptions should be included in the generated script. By default this option is enabled.
DescriptionsAsUpdate	Determines whether the raw UPDATE statement should be used for object descriptions instead of the IBExpert specific DESCRIBE statement.
UseComment	Generates the COMMENT ON statement for object descriptions (Firebird 2.x).
DontUseSetTerm	Don't use SET TERM statements, all statements will be separated by semicolon only.
UseCreateOrAlter	Generates CREATE OR ALTER instead of CREATE/ALTER where possible.
ProgressBlock	An IBEBlock which will be executed for every progress message generated during script execution. May be NULL or empty.

Description

ibec_GetViewRecreateScript creates a *Recreate* script for a specified view(s) and returns it as a result.

Use the IBExpert DB Explorer context-sensitive menu item, *Apply Block to selected objects ...* to recreate selected views based on IBEBlock and the ibec_GetViewRecreateScript function.

Example

```
execute ibeblock
as
begin
  cbb = 'execute ibeblock (MsgData variant)
         as
         begin
           ibec_Progress(MsgData);
         end';
  ...
  RecreateScript = ibec_GetViewRecreateScript(mydb, 'VIEW_A; VIEW_B; VIEW_C',
      'GenerateConnect; IncludePassword; UseCreateOrAlter', cbb);
  Res = ibec_ExecSQLScript(null, RecreateScript, 'ServerVersion=FB21', cbb);
end
```

ibec_GUID

This function creates a string representation of a GUID, a unique 128-bit integer used for CLSIDs and interface identifiers.

ibec_High

Returns the highest value within the range of the index type of the array.

Syntax

```
function ibec_High(AArray : array of variants): integer;
```

Example

```
execute IBEBlock
returns (iresult integer)
as
begin
  vals = 0;
  iresult = ibec_High(vals);
  suspend; /* iresult = 0 */

  vals[1] = 12;
  iresult = ibec_High(vals);
  suspend; /* iresult = 1 */

  vals[10] = 'ibexpert';
  iresult = ibec_High(vals);
  suspend; /* iresult = 10 */

  ibec_SetLength(vals, 5);
  iresult = ibec_High(vals);
  suspend; /* iresult = 4 */

  ibec_SetLength(vals, 500);
  iresult = ibec_High(vals);
  suspend; /* iresult = 499 */

  ibec_SetLength(vals, 0);
  iresult = ibec_High(vals);
  suspend; /* iresult = 0 */
end
```

See also:
`ibec_SetLength`
Data Comparer using cursors

ibec_IIF

Tests a condition and returns `Value1` if the `Condition` is `True` and `Value2` if the `Condition` is `False`.

Syntax

```
function ibec_IIF(Condition : boolean; Value1, Value2 : variant): variant;
```

Description

Tests a condition and returns `Value1` if the `Condition` is `True` and `Value2` if the `Condition` is `False`.

Example

```
execute IBEBlock
returns (cout varchar(100))
as
begin
  i = 1;
  while (I < 50) do
  begin
    cout = ibec_IIF((ibec_mod(i, 2) = 0), i || ' is even number', i || ' is odd number');
    suspend;
    i = i + 1;
  end
end
```

See also:
`IIF`
Firebird 2.0.4. Release Notes: `IIF` expression syntax added

ibec_IntToHex

Returns the hex representation of an integer.

Syntax

```
function ibec_IntToHex(Value: Integer; Digits: Integer): string;
```

Description

`ibec_IntToHex` converts a number into a string containing the number's hexadecimal (base 16) representation. `Value` is the number to convert. `Digits` indicates the minimum number of hexadecimal digits to return.

Example

```
execute ibeblock
returns (iint integer, shex varchar(5))
as
begin
  iint = 0;
  while (iint < 1000) do
  begin
    shex = '$' || ibec_IntToHex(iint, 4);
    iint = iint + 1;
    suspend;
  end
end
```

See also:
Creating an UPDATE script with domain descriptions

ibec_MessageDlg

The `ibec_MessageDlg` function was implemented in IBExpert version 2006.12.11. This function displays a message dialog box in the center of the screen.

Syntax

```
function ibec_MessageDlg(Msg: string; DlgType: integer; Buttons: integer): integer;
```

Description

Call `ibec_MessageDlg` to bring up a message box and obtain the user's response. The message box displays the value of the `Msg` parameter. Use the `DlgType` parameter to indicate the purpose of the dialog. Possible values of the `DlgType` parameter are:

`__mtWarning = 0`	A message box containing a yellow exclamation point symbol.
`__mtError = 1`	A message box containing a red stop sign.
`__mtInformation = 2`	A message box containing a blue `i`.
`__mtConfirmation = 3`	A message box containing a green question mark (`?`).
`__mtCustom = 4`	A message box containing no bitmap.

Use the `Buttons` parameter to indicate which buttons should appear in the message box. The following values and combinations can be used for the `Buttons` parameters:

`__mbYes = 1`	A button with *Yes* on its face.
`__mbNo = 2`	A button the text *No* on its face.
`__mbOK = 4`	A button the text *OK* on its face.
`__mbCancel = 8`	A button with the text *Cancel* on its face.
`__mbAbort = 16`	A button with the text *Abort* on its face.
`__mbRetry = 32`	A button with the text *Retry* on its face.
`__mbIgnore = 64`	A button the text *Ignore* on its face.
`__mbAll = 128`	A button with the text *All* on its face.
`__mbNoToAll = 256`	A button with the text *No to All* on its face.
`__mbYesToAll = 512`	A button with the text *Yes to All* on its face.
`__mbHelp = 1024`	A button with the text *Help* on its face.

`ibec_MessageDlg` returns the value of the button the user selected. These are the possible return values:

```
__mrNone
__mrOk
__mrCancel
__mrAbort
__mrRetry
__mrIgnore
__mrYes
__mrNo
__mrAll
__mrNoToAll
__mrYesToAll
```

ibec_Ord

Returns the ordinal value of the specified character.

Syntax

```
function ibec_Ord(Chr : char): integer;
```

Description

The `ibec_Ord` function returns the ordinal value of the specified character. If `Chr` is an empty string or `NULL`, then result is `0`.

Example

```
execute IBEBlock
returns (cout varchar(1))
as
begin
  i = 0;
  while (i < 256) do
  begin
    cout = ibec_Chr(i);
    i = i + 1;
    suspend;
  end
end
```

See also:
`ibec_Chr`

ibec_ParseCSVLine

Syntax

```
function ibec_fs_ParseCSVLine(DestValues : array of variants; CSVLine : string; QuoteChar : char; Delimiter : string; Options :
cardinal): integer;
```

See also:
Importing data from a CSV file

ibec_Progress

Displays a progress message.

Syntax

```
function ibec_Progress(Mes : string): string;
```

Description

Call `ibec_Progress` function to display a message. The `Msg` parameter is the message string that appears in the upper status panel of the SQL Editor or Script Editor. If you're executing an IBEBlock using the IBEScript tool the message will appear on the screen and will be included into log file .

Example

```
execute IBEBlock
returns (table_name varchar(31), irecords integer)
as
begin
  for select rdb$relation_name
      from rdb$relations
      order by rdb$relation_name
      into :table_name
  do
  begin
    ibec_Progress('Counting records of ' || ibec_Trim(table_name));
    execute statement 'select count(*) from ' || ibec_Trim(table_name) into :irecords;
    suspend;
  end
end
```

See also:
Comparing databases using IBEBlock
Comparing scripts with IBEBlock

ibec_Random

Generates random numbers within a specified range.

Syntax

```
function ibec_Random(Range : integer): integer;
```

Description

`ibec_Random` returns a random number within the range `0 <= X < Range`. If `Range=0`, the result is a real-type random number within the range `0 <= X < 1`.

Example

```
execute IBEBlock
returns (iout integer, dpout double precision)
as
begin
  i = 0;
  while (i < 100) do
  begin
    iout = ibec_Random(100);
    dpout = ibec_Random(0);
    i = i + 1;
    suspend;
  end
end
```

See also:
`ibec_Random2`
`ibec_RandomChar`
`ibec_RandomString`
`ibec_RandomVal`
Data Comparer using cursors
`IBEBLOCK` and Test Data Generator

ibec_Random2

Generates random numbers within a specified range.

Syntax

```
function ibec_Random2(MinValue, MaxValue : integer): integer;
```

Description

`ibec_Random2` returns a random number within the range `MinValue <= X <= MaxValue`.

Example

```
execute IBEBlock
returns (iout integer)
as
begin
  i = 0;
  while (i < 100) do
  begin
    iout = ibec_Random2(50, 100);
    i = i + 1;
    suspend;
  end
end
```

See also:
`ibec_Random`
`ibec_RandomChar`
`ibec_RandomString`
`ibec_RandomVal`
Data Comparer using cursors
`IBEBLOCK` and Test Data Generator

ibec_RandomChar

Generates random char within a specified range.

Syntax

```
function ibec_RandomChar(MinOrdValue, MaxOrdValue : integer): string;
```

Description

`ibec_RandomChar` returns a random char within the range `MinOrdValue <= X <= MaxOrdValue`.

Example

```
execute IBEBlock
returns (cout varchar(1))
as
begin
  i = 0;
  while (i < 100) do
  begin
    cout = ibec_RandomChar(1, 255);
    i = i + 1;
    suspend;
  end
end
```

See also:
`ibec_Random`
`ibec_Random2`
`ibec_RandomString`
`ibec_RandomVal`
Data Comparer using cursors
`IBEBLOCK` and Test Data Generator

ibec_RandomString

Returns a random string.

Syntax

```
function ibec_RandomString(MinLen, MaxLen, MinOrdValue, MaxOrdValue : integer): string;
```

See also:
`ibec_Random`
`ibec_Random2`
`ibec_RandomChar`
`ibec_RandomVal`
Data Comparer using cursors
`IBEBLOCK` and Test Data Generator

ibec_RandomVal

See also:
`ibec_Random`
`ibec_Random2`
`ibec_RandomChar`
`ibec_RandomString`
Data Comparer using cursors
`IBEBLOCK` and Test Data Generator

ibec_SetGlobalVar

`ibec_SetGlobalVar` allows you to create/modify a global variable. This function always returns `0`.

Syntax

```
function ibec_SetGlobarVar
   (VarName : string; VarValue : variant):variant;
```

Description

If you're using the `ibec_SetGlobalVar` function within scripts executed with IBEScript, it is not necessary to free global variables - they will be destroyed automatically after the script has finished.

If you're using the `ibec_SetGlobalVar` function within IBExpert (SQL Editor or the Script Executive), any global variables created will continue to exist until you close IBExpert. So if necessary, you should free them manually using the `ibec_FreeGlobalVar` function.

Example

The following example illustrates the use of this function, together with `ibec_GetGlobalVar`, described within an SQL script:

```
CONNECT ...;

execute ibeblock
as
begin
  select myfield from mytable
  where something = 25
  into :MyVar;
  ibec_SetGlobalVar('MyGlobalVar', MyVar);
end;

...

execute ibeblock
as
begin
  MyVar = ibec_GetGlobalVar('MyGlobalVar', null);
  if (MyVar = 1) then
    insert into mytable ...;
  else if (MyVar = 2) then
    update mytable set ...;
end;
```

ibec_SetLength

Sets the length of a dynamic-array variable.

Syntax

```
function ibec_SetLength(AArray : array of variants; NewLength : integer): integer;
```

Description

`AArray` is a dynamic-array variable.

`ibec_SetLength` reallocates the array referenced by `AArray` to the given length. Existing elements in the array are preserved, the content of newly allocated elements is `NULL`. `ibec_SetLength` returns the number of array elements.

Example

```
execute IBEBlock
returns (iresult integer)
as
begin
  vals = 0;
  iresult = ibec_SetLength(vals, 10);
  suspend; /* iresult = 10 */

  iresult = ibec_SetLength(vals, -1);
  suspend; /* illegal NewLength, iresult = 10 */

  iresult = ibec_SetLength(vals, '25');
  suspend; /* iresult = 25 */

  iresult = ibec_SetLength(vals, NULL);
  suspend; /* illegal NewLength, iresult = 25 */
end
```

See also:
`ibec_High`

ibec_ShiftRecord

Syntax

```
function ibec_ShiftRecord(AArray : array of variants; Shift : integer): integer;
```

ibec_smtp_SendMail

This function sends an email using SMTP protocol.

Syntax

```
function ibec_smtp_SendMail(SMTPHost : string; SMTPPort : string; UserName : string;
                            Password : string; From : string; To : string; CC : string; BCC : string;
                            Subject : string; Message : string; AttachedFiles : string;
                            AdditionalHeaders : string; Options : string; CallbackBlock : string)

: variant;
```

A detailed description of this function will be available later.

Example

```
execute ibeblock
as
begin
  CRLF = ibec_CRLF();
  cbb = 'execute ibeblock (Vals variant)
         as
         begin
           sPref = ;
           sEvent = Vals[EVENT];
           if ((sEvent = COMMAND) or (sEvent = HEADER)) then
             sPref = ==> ;
           else if (sEvent = RESPONSE) then
             sPref = <== ;
           sMes = sPref + Vals[TEXT];
           ibec_Progress(sMes);
           LogFile = ibec_GetGlobalVar(LogFileH, null);
           if (LogFile is not NULL) then
             ibec_fs_Writeln(LogFile, sMes);
        end';
  sMessage = 'Just a test' + CRLF +
             'This message was sent by ibec_smtp_SendMail function';
  sAttachments = 'D:\smtpsendmail.ibeblock' + CRLF +
                 'D:\script.sql';
  sAddHeaders  = 'IBE-Type: IBEBlock' + CRLF +
                 'IBE-Comment: Just a comment';
  LogFile = ibec_fs_OpenFile('D:\smtp.log', __fmCreate);
  try
    if (LogFile is not null) then
      ibec_SetGlobalVar('LogFileH', LogFile);
    ibec_smtp_SendMail('mail.myserver.com',
                       'smtp',
                       'Bill',
                       'windows_must_die!',
                       '"Bill Gates" <Bill@microsoft.com>',
                       'all@world.com',
                       '',
                       '',
                       'Test message from IBEBlock ibec_smtp_SendMail function',
                       :sMessage,
                       :sAttachments,
                       :sAddHeaders,
                       'Encoding=windows-1251; Confirm; Priority=Highest',
                       cbb);
  finally
    ibec_fs_CloseFile(LogFile);
  end;
end
```

ibec_WaitForEvent

The `ibec_WaitForEvent` function can be used to monitor events sent by the `POST_EVENT` command. It returns the event name if an event is fired or `NULL` if timeout is expired.

Syntax

```
ibec_WaitForEvent(Connection : variant; EventName : string; Timeout : cardinal) : variant;#
```

`Timeout` should be specified in milliseconds. `Timeout = 0` means infinitely waiting for event!

IBEBlock Examples

This section includes a few examples illustrating the usage of `EXECUTE IBEBLOCK` (please refer to the individual subjects for details).

All scripts, demos etc. can be downloaded from http://www.ibexpert.com/download/other_files/ (save `BlockScriptSamples.zip` to the hard drive and extract).

Automatic script execution

It is possible to execute any script automatically, simply by placing the script in a file, `ibexpert.usr`, in the main IBExpert directory.

Since IBExpert version 2006.08.12 it is also possible to execute a script automatically immediately after IBExpert starts. Please refer to IBExpert After Start Script for further information and an example.

Try it!

ODBC Access

1. Download `IBEBlockScriptSamples.zip` from http://www.ibexpert.com/download/other_files/
2. Copy `Demo.mdb` and `ODBCAcc.ibeblock` (found in the `Blocks/ODBC Access` directory) into a separate directory
3. Copy `ODBCAcc.ibeblock` (copy of script below) into the SQL Editor.
4. You can find the correct connection string for the ODBC driver you are using here: http://www.connectionstrings.com/
5. Modify the path to `Demo.mdb`.
6. Press [F9] to execute the block.

```
execute ibeblock
returns (CustNo integer, Company varchar(100), Addr1 varchar(100))
    as
    begin
    InCust = 3000;
    OdbcCon = ibec_CreateConnection(__ctODBC, 'DBQ=D:\Delphi5\CMP\mODBC\DB\demo.mdb;DRIVER=Microsoft Access Driver (*.mdb)');
    ibec_UseConnection(OdbcCon);

    execute statement 'select Company from customer where CustNo = 4312' into :MyCust;

    for select CustNo, Company, Addr1 from customer
       where CustNo > :InCust
       order by company
       into :CustNo, :Company, :Addr1
    do
    begin
       suspend;
    end
    ibec_CloseConnection(OdbcCon);
 end
```

Extract metadata using IBEBlock

1. Download `IBEBlockScriptSamples.zip` from http://www.ibexpert.com/download/other_files/, and copy all IBEBlocks found in the `Extract Metadata` directory into a separate directory.
2. Load the `RunMe.ibeblock` into the SQL Editor.
3. Replace the default values of `CodeDir` and `ScriptFile` input parameters with your own.
4. Press [F9] to execute the block.

Note: this is just an example, therefore only generators, domains and procedures will be extracted into the script.

The individual sample scripts:

- `DomExtract.ibeblock`
- `FldType.ibeblock`
- `GensExtract.ibeblock`
- `SPExtract.ibeblock`
- `RunMe.ibeblock`

can be viewed in the following sections.

See also:
Extract Metadata
`ibec_ExtractMetadata`

DomExtract.ibeblock

```
execute ibeblock (
   CodeDir varchar(1000) = 'E:\IBEBlocks\' comment 'Path to necessary IBEBlocks',
   FileStrm variant)
as
begin
   FldTypeFunc = ibec_LoadFromFile(CodeDir || 'FldType.ibeblock');

   if (FileStrm is not null) then
      FS = FileStrm;
   else
      FS = ibec_fs_OpenFile('E:\BlockScript.sql', __fmCreate);

for select f.rdb$field_name, -- 0
           f.rdb$validation_source, -- 1
           f.rdb$computed_source, -- 2
           f.rdb$default_source, -- 3
           f.rdb$field_length, -- 4
           f.rdb$field_scale, -- 5
           f.rdb$field_type, -- 6
           f.rdb$field_sub_type, -- 7
           f.rdb$description, -- 8
           f.rdb$segment_length, -- 9
           f.rdb$dimensions, -- 10
           f.rdb$null_flag, -- 11
           f.rdb$character_length, -- 12
           f.rdb$collation_id, -- 13
           f.rdb$character_set_id, -- 14
           f.rdb$field_precision, -- 15
           ch.rdb$character_set_name, -- 16
           co.rdb$collation_name -- 17
      from rdb$fields fleft join rdb$character_sets ch on (f.rdb$character_set_id =  ch.rdb$character_set_id)
      left join rdb$collations co on ((f.rdb$collation_id = co.rdb$collation_id) and
           (f.rdb$character_set_id = co.rdb$character_set_id))
      where not (f.rdb$field_name starting with 'RDB$')
      order by rdb$field_name
      into :DomProps
  do
  begin
     DomName = DomProps[0];
         execute ibeblock FldTypeFunc(DomProps[6], DomProps[7], DomProps[4], DomProps[5], DomProps[9],
                                      DomProps[12], DomProps[15], 3)
         returning_values :FieldType;
     DomType = FieldType;

     -- Character Set
     if ((DomProps[6] in (14, 37, 261)) and (DomProps[16] is not null)) then
        DomType = DomType || ' CHARACTER SET ' || ibec_trim(DomProps[16]) || ibec_Chr(13) || ibec_Chr (10);

     -- Default Value
     if ((DomProps[3] is not null) and (DomProps[3] <> '')) then
        DomType = DomType || ibec_trim(DomProps[3]) || ibec_Chr(13) || ibec_Chr(10);

     -- NOT NULL flag
     if (DomProps[11] is not null) then
        DomType = DomType || 'NOT NULL' || ibec_Chr(13) || ibec_Chr(10);

     -- Check source
     if ((DomProps[1] is not null) and (DomProps[1] <> '')) then
        DomType = DomType || ibec_trim(DomProps[1]) || ibec_Chr(13) || ibec_Chr(10);

     -- Collate
     if ((DomProps[17] is not null) and (DomProps[17] <> '')) then
        DomType = DomType || 'COLLATE ' || ibec_trim(DomProps[17]) || ibec_Chr(13) || ibec_Chr(10);

     DomType = ibec_Chr(13) || ibec_Chr(10) || ibec_Trim(DomType) || ';';
     ibec_progress('Writing domain ' || DomName);
     ibec_fs_Writeln(FS, 'CREATE DOMAIN ' || ibec_Trim(DomProps[0]) || DomType);
     ibec_fs_Writeln(FS, '');
  end

  if (FileStrm is null) then
     ibec_fs_CloseFile(FS);
end
```

FldType.ibeblock

```
execute ibeblock (
   FType integer,
   FSubType integer,
   FLen integer,
   FScale integer,
   FSegmentSize integer,
   FCharLen integer,
   FPrecision integer,
   SQLDialect integer = 3)
returns (TypeAsString varchar(200))
as
begin
   TypeAsString = '';
   if ((FCharLen = 0) or (FCharLen is NULL)) then
      FCharLen = FLen;

   if (FType = 261) then
      TypeAsString = ibec_Concat('BLOB SUB_TYPE ', FSubType, ' SEGMENT SIZE ', FSegmentSize);
   else if (FType = 14) then
      TypeAsString = 'CHAR(' || FCharLen || ')';
   else if (FType = 37) then
      TypeAsString = 'VARCHAR(' || FCharLen || ')';
   else if (FType = 12) then
      TypeAsString = 'DATE';
   else if (FType = 13) then
      TypeAsString = 'TIME';
   else if (FType = 35) then
   begin
      if (SQLDialect = 3) then
         TypeAsString = 'TIMESTAMP';
      else
         TypeAsString = 'DATE';
   end
   else if (FType = 7) then
   begin
      if ((FScale < 0) or (FSubType = 1) or (FSubType = 2)) then
      begin
         if (FSubType = 2) then
            TypeAsString = 'DECIMAL';
         else
            TypeAsString = 'NUMERIC';
         if (FPrecision > 0) then
            TypeAsString = TypeAsString || '(' || FPrecision || ',' || (FScale * -1) || ')';
         else
            TypeAsString = TypeAsString || '(4,' || (FScale * -1) || ')';
      end
      else
         TypeAsString = 'SMALLINT';
   end
   else if (FType = 8) then
   begin
      if ((FScale < 0) or (FSubType = 1) or (FSubType = 2)) then
      begin
         if (FSubType = 2) then
            TypeAsString = 'DECIMAL';
         else
            TypeAsString = 'NUMERIC';
         'if (FPrecision > 0) then
            TypeAsString = TypeAsString || '(' || FPrecision || ',' || (FScale * -1) || ')';
         else
            TypeAsString = TypeAsString || '(9,' || (FScale * -1) || ')';
      end
      else
         TypeAsString = 'INTEGER';
   end
   else if (FType = 27) then
   begin
      if ((FScale < 0) or (FSubType = 1) or (FSubType = 2)) then
      begin
         if (FSubType = 2) then
            TypeAsString = 'DECIMAL';
         else
            TypeAsString = 'NUMERIC';
         if (FPrecision > 0) then
            TypeAsString = TypeAsString || '(' || FPrecision || ',' || (FScale * -1) || ')';
         else
            TypeAsString = TypeAsString || '(9,' || (FScale * -1) || ')';
      end
      else
         TypeAsString = 'DOUBLE PRECISION';
   end
   else if (FType = 16) then
   begin
      if ((FScale < 0) or (FSubType = 1) or (FSubType = 2)) then
      begin
         if (FSubType = 2) then
            TypeAsString = 'DECIMAL';
         else
            TypeAsString = 'NUMERIC';
         if (FPrecision > 0) then
```

```
            TypeAsString = TypeAsString || '(' || FPrecision || ',' || (FScale * -1) || ')';
         else
            TypeAsString = TypeAsString || '(18,' || (FScale * -1) || ')';
      end
      else
         TypeAsString = 'BIGINT';
   end
   else if (FType = 10) then
      TypeAsString = 'FLOAT';
   suspend;
end
```

GensExtract.ibeblock

```
execute ibeblock (
   SetValues smallint = 0,
   FileStrm variant)
as
begin
   if (FileStrm is not null) then
      FS = FileStrm;
   else
      FS = ibec_fs_OpenFile('E:\BlockScript.sql', __fmCreate);

   for select g.rdb$generator_name
       from rdb$generators g
       where g.rdb$system_flag is null
       order by g.rdb$generator_name
       into :GenName
   do
   begin
      GenName = ibec_trim(GenName);
      s = 'CREATE GENERATOR ' || GenName || ';';
      if (SetValues = 1) then
      begin
         execute statement 'select gen_id(' || GenName || ', 0) from rdb$database' into :GenValue;
         s = s || ibec_Chr(13) || ibec_Chr(10) ||
            'SET GENERATOR ' || GenName || ' TO ' || GenValue || ';';
      end
      ibec_progress('Writing generator ' || GenName);
      ibec_fs_Writeln(FS, s);
      ibec_fs_Writeln(FS, '');
   end

  if (FS is null) then
     ibec_fs_CloseFile(FS);
  end
```

SPExtract.ibeblock

```
execute ibeblock ExtractProcedures (
   CodeDir varchar(1000) = 'E:\IBEBlocks\' comment 'Path to necessary IBEBlocks',
   CreateAlter varchar(6) = 'CREATE',
   Dialect smallint = 3,
   EmptyBody boolean = FALSE,
   FileStrm variant)
as
begin
   CRLF = ibec_CRLF;
   WriteDDLBlock =
      'execute ibeblock (sName variant, sDDL variant, sInParams variant, sOutParams variant, sSrc variant, FS variant)
   as
      CRLF = ibec_CRLF();
      if (sInParams <> ´´´´) then
         sDDL = sDDL || ´´ (´´ || CRLF || ´´ ´´ || ibec_Trim(sInParams) || ´´)´´;
      if (sOutParams <> ´´´´) then
         sDDL = sDDL || CRLF || ´´RETURNS (´´ || CRLF || ´´ ´´ || ibec_Trim(sOutParams) || ´´)´´;
      sDDL = sDDL || CRLF || ´´AS´´ || CRLF;
      sDDL = sDDL || sSrc || ´´^´´;
      ibec_progress(''Writing procedure ´´ || sName);
      ibec_fs_Writeln(FS, sDDL); ibec_fs_Writeln(FS, ´´´´); ibec_fs_Writeln(FS, ´´´´);
   end';

   RdbPrecisionExists = TRUE;
   FldTypeFunc = ibec_LoadFromFile(CodeDir || 'FldType.ibeblock');

   sName = ´´; sDDL = ´´; sInParams = ´´; sOutParams = ´´; sParam = ´´; iPrec = 0;
   if (FileStrm is not null) then
      FS = FileStrm;
   else
      FS = ibec_fs_OpenFile('E:\BlockScript.sql', __fmCreate);

   Stmt = ibec_Concat(
   'select pr.rdb$procedure_name,    ', CRLF,  -- 0
   '       pp.rdb$parameter_name,    ', CRLF,  -- 1
   '       pp.rdb$parameter_type,    ', CRLF,  -- 2
   '       fs.rdb$field_name,        ', CRLF,  -- 3
   '       fs.rdb$field_type,        ', CRLF,  -- 4
   '       fs.rdb$field_length,      ', CRLF,  -- 5
   '       fs.rdb$field_scale,       ', CRLF,  -- 6
   '       fs.rdb$field_sub_type,    ', CRLF,  -- 7
   '       fs.rdb$segment_length,    ', CRLF,  -- 8
   '       fs.rdb$dimensions,        ', CRLF,  -- 9
   '       cr.rdb$character_set_name,', CRLF,  -- 10
   '       co.rdb$collation_name,    ', CRLF,  -- 11
   '       pp.rdb$parameter_number,  ', CRLF,  -- 12
   '       fs.rdb$character_length,  ', CRLF,  -- 13
   '       fs.rdb$default_source     ', CRLF); -- 14

   if (not EmptyBody) then
      Stmt = ibec_Trim(Stmt) || ',' || CRLF || ' pr.rdb$procedure_source' || CRLF;
   else
      sSrc = 'BEGIN' || CRLF || ' EXIT;' || CRLF || 'END';

   if (RdbPrecisionExists) then
      Stmt = ibec_Trim(Stmt) || ',' || CRLF ||
             '        fs.rdb$field_precision' || CRLF;
   Stmt = Stmt ||
   'from rdb$procedures pr'               || CRLF ||
   'left join rdb$procedure_parameters pp on pp.rdb$procedure_name = pr.rdb$procedure_name' || CRLF ||
   'left join rdb$fields fs on fs.rdb$field_name = pp.rdb$field_source'                    || CRLF ||
   'left join rdb$character_sets cr on fs.rdb$character_set_id = cr.rdb$character_set_id'   || CRLF ||
   'left join rdb$collations co on ((fs.rdb$collation_id = co.rdb$collation_id) and'        || CRLF ||
   '                                (fs.rdb$character_set_id = co.rdb$character_set_id))'   || CRLF ||
   'order by pr.rdb$procedure_name, pp.rdb$parameter_type, pp.rdb$parameter_number';

   SetTermWritten = FALSE;

   for execute statement :Stmt into :SPProps
   do
   begin
      if (SetTermWritten = FALSE) then
      begin
         ibec_fs_Writeln(FS, 'SET TERM ^ ;' || CRLF);
         SetTermWritten = TRUE;
      end;
      if (RdbPrecisionExists = TRUE) then
         iPrec = ibec_IIF(EmptyBody = 1, SPProps[15], SPProps[16]);

      SPName = ibec_Trim(SPProps[0]);
      if (sName <> SPName) then
      begin
         if (sDDL <> ´´) then
            execute ibeblock WriteDDLBlock(sName, sDDL, sInParams, sOutParams, sSrc, FS);

         sName = SPName;
         if (not EmptyBody) then
            sSrc = ibec_Trim(SPProps[15]);
         sDDL = CreateAlter || ' PROCEDURE ' || SPName;
         sInParams = ´´; sOutParams = ´´; sParam = ´´;
```

```
      end
      if (SPProps[1] is not null) then
      begin
         execute ibeblock FldTypeFunc(SPProps[4], SPProps[7], SPProps[5], SPProps[6], SPProps[8],
                                      SPProps[13], SPProps[16], Dialect)
                returning_values :sParam;
         sParam = ibec_Trim(SPProps[1]) || ' ' || sParam;
         -- Character Set
         if ((SPProps[4] in (14, 37, 261)) and (SPProps[10] is not null)) then
            sParam = sParam || ' CHARACTER SET ' || ibec_trim(SPProps[10]);
         -- Default Value
         if ((SPProps[14] is not null) and (SPProps[14] <> '')) then
            sParam = sParam || ' DEFAULT ' || ibec_trim(SPProps[14]);
         if (SPProps[2] = 0) then
         begin
            if (sInParams <> '') then
               sInParams = sInParams || ',' || CRLF || ' ';
            sInParams = sInParams || sParam;
         end
         else if (SPProps[2] = 1) then
         begin
            if (sOutParams <> '') then
               sOutParams = sOutParams || ',' || CRLF || ' ';
            sOutParams = sOutParams || sParam;
         end
      end
   end
   if (sDDL <> '') then
      execute ibeblock WriteDDLBlock(sName, sDDL, sInParams, sOutParams, sSrc, FS);

   if (SetTermWritten) then
      ibec_fs_Writeln(FS, 'SET TERM ; ^' || CRLF);

   if (FileStrm is null) then
      ibec_fs_CloseFile(FS);
end
```

RunMe.ibeblock

```
execute ibeblock ExtractMetadata (
   CodeDir varchar(1000) = 'E:\IBEBlocks\' comment 'Path to necessary IBEBlocks'
   ScriptFile varchar(1000) = 'E:\BlockScript.sql' comment 'Name of the script file')
returns (TimeAll float)
as
begin
   Time1 = ibec_GetTickCount;
   SPExtr = ibec_LoadFromFile(CodeDir || 'SPExtract.ibeblock');
   DomExtract = ibec_LoadFromFile(CodeDir || 'DomExtract.ibeblock');
   GensExtract = ibec_LoadFromFile(CodeDir || 'GensExtract.ibeblock');

   FS = ibec_fs_OpenFile(ScriptFile, __fmCreate);
   execute ibeblock DomExtract (FS);
   execute ibeblock GensExtract (1, FS);
   execute ibeblock SPExtr (CodeDir, 'CREATE', 3, TRUE, FS);
   execute ibeblock SPExtr (CodeDir, 'ALTER', 3, FALSE, FS);
   ibec_fs_CloseFile(FS);
   Time2 = ibec_GetTickCount();
   TimeAll = (Time2 - Time1) / 1000;
   suspend;
end
```

Comparing databases using IBEBlock

```
execute ibeblock
as
begin
   create connection MasterDB dbname 'localhost:c:\MasterDB.fdb'
   password 'masterkey' user 'SYSDBA'
   clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

   create connection SubscriberDB dbname 'localhost:c:\SubscriberDB.fdb'
   password 'masterkey' user 'SYSDBA'
   sql_dialect 3
   clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';


   cbb = 'execute ibeblock (LogMessage variant)
     as
     begin
       ibec_progress(LogMessage);
     end';

ibec_CompareMetadata(MasterDB, SubscriberDB, 'E:\CompRes.sql', 'OmitDescriptions;
OmitGrants', cbb);

   close connection MasterDB;
   close connection SubscriberDB;
end
```

See also:
Extract Metadata

Comparing scripts with IBEBlock

```
execute ibeblock
as
begin
   cbb = 'execute ibeblock (
        LogMessage variant)
      as
      begin
        ibec_progress(LogMessage);
      end';

ibec_CompareMetadata('c:\myscripts\master.sql','c:\myscripts\subscriber.sql', 'E:\CompRes.sql','', cbb);
end
```

- Using the ServerVersion parameter (IBExpert version 2005.12.04):

```
ibec_CompareMetadata(MasterDB,
                     SubscriberDB,
                     'E:\CompRes.sql',
                     'OmitDescriptions; OmitGrants; ServerVersion=FB1?',
                     cbb);
```

See also:
Extract Metadata

Automatic database structure comparison with recompilation of triggers and procedures

```
execute ibeblock
as
begin
  create connection MasterDB dbname 'localhost:c:\db1.fdb'
  password 'masterkey' user 'SYSDBA'
  clientlib 'fbclient.dll';

  create connection SubscriberDB dbname 'localhost:c:\db2.fdb'
  password 'masterkey' user 'SYSDBA'
  clientlib 'fbclient.dll';


  cbb = 'execute ibeblock (LogMessage variant)
    as
    begin
      ibec_progress(LogMessage);
    end';
    ibec_CompareMetadata(MasterDB, SubscriberDB, 'E:\CompRes.sql', 'OmitDescriptions; OmitGrants', cbb);

  close connection MasterDB;
  close connection SubscriberDB;
end ;

input 'E:\CompRes.sql';

execute ibeblock
as
begin
  create connection SubscriberDB dbname 'localhost:c:\db2.fdb'
  password 'masterkey' user 'SYSDBA'
  clientlib 'fbclient.dll';

  e=ibec_RecompileProcedure(SubscriberDB, '');
  e=ibec_RecompileTrigger(SubscriberDB,'')
  close connection SubscriberDB;
end;
```

Data Comparer using cursors

The following example illustrates the use of cursors to compare two tables in different databases.

```
execute ibeblock (
  ProcessInserts boolean = TRUE,
   ProcessUpdates boolean = TRUE,
   ProcessDeletes boolean = TRUE)
returns (
   InsertedRecs integer = 0 comment ´´Records inserted´´,
   UpdatedRecs integer = 0 comment ´´Records updated´´,
   DeletedRecs integer = 0 comment ´´Records deleted´´,
   TotalTime double precision = 0 comment ´´Time spent (seconds)´´)
as
begin
   RecNum = 50000; -- How many records will be inserted into our test table
```

If the databases already exist we will not try to create them. Of course, this this approach does not apply to remote databases.

```
   if (not ibec_fileexists(´´c:\MasterDB.fdb´´)) then
      create database ´´localhost:c:\MasterDB.fdb´´ user ´´SYSDBA´´ password ´´masterkey´´
      page_size 4096 sql_dialect 3
```

`CLIENTLIB` isn't mandatory if you're using the standard `gds32.dll`.

```
      clientlib ´´C:\Program Files\Firebird\bin\fbclient.dll´´;

   if (not ibec_fileexists(´´c:\SubscriberDB.fdb´´)) then
      create database ´´localhost:c:\SubscriberDB.fdb´´ user ´´SYSDBA´´ password ´´masterkey´´
      page_size 4096 sql_dialect 3
      clientlib ´´C:\Program Files\Firebird\bin\fbclient.dll´´;
```

Creating two named connections to our databases...

```
   create connection MasterDB dbname ´´localhost:c:\MasterDB.fdb´´
   password ´´masterkey´´ user ´´SYSDBA´´
   clientlib ´´C:\Program Files\Firebird\bin\fbclient.dll´´;

   create connection SubscriberDB dbname ´´localhost:c:\SubscriberDB.fdb´´
   password ´´masterkey´´ user ´´SYSDBA´´
   sql_dialect 3
   clientlib ´´C:\Program Files\Firebird\bin\fbclient.dl´´';
```

Now we shall create the `IBE$$TEST_DATA` table in each database and populate it with some data:

```
   CreateStmt =
      ´´create table IBE$$TEST_DATA (
        ID integer not null,
        ID2 varchar(20) not null,
        F_INTEGER integer,
        F_VARCHAR varchar(100),
        F_DATE date,
        F_TIME time,
        F_TIMESTAMP timestamp,
        F_NUMERIC numeric(15,2),
        F_BOOL char(1) check (F_BOOL in (´´T´´, ´´F´´)),
        F_BLOB blob sub_type 1,
        F_SEASON varchar(15) check(F_SEASON in (´´Spring´´, ´´Summer´´, ´´Autumn´´, ´´Winter´´)))´´;
```

`IBE$$TEST_DATA` will have a primary key consisting of two fields. This is just to demonstrate how to do this when a primary key consists of more than one field.

```
   AlterStmt =
      ´´alter table IBE$$TEST_DATA add constraint PK_IBE$$TEST_DATA primary key (ID, ID2)´´;
```

First we're working with the `MasterDB`:

```
   use MasterDB;
```

If `IBE$$TEST_DATA` doesn't exist in the database we must create it:

```
   if (not exists(select rdb$relation_name from rdb$relations where rdb$relation_name = ´´IBE$$TEST_DATA´´)) then
   begin
```

Creating the table itself...

```
      execute statement :CreateStmt;
```

DDL statements must be committed explicitly:

```
      commit;
```

...and create a primary key:

```
execute statement :AlterStmt;
commit;
```

So, we've just created the table. Now we should populate it with data. We will generate some random data for each field, and use an autoincrement for the first primary key field value:

```
i = 0;
while (i < RecNum) do
begin
   fid2    = ibec_randomstring(1,20,65,90);
   fint    = ibec_random2(1, 100000);
   fvarc   = ibec_randomstring(1,100,65,90);
   fdate   = ibec_random2(20000,40000);
   ftime   = ibec_random(0);
   ftimest = ibec_random2(20000,40000) + ibec_random(0);
   fnum    = ibec_random2(1,40000) + ibec_random(0);
   fbool   = ibec_randomval(´´T´´,´´F´´);
   fblob   = ibec_randomstring(500, 1000, 0, 255);
   fseason = ibec_randomval(´´Spring'´´, ´´Summer´´, ´´Autumn´´, ´´Winter´´);

   insert into IBE$$TEST_DATA values (:i, :fid2, :fint, :fvarc, :fdate, :ftime, :ftimest, :fnum, :fbool, :fblob, :fseason);
   i = i + 1;
```

We will display a progress message after each 500 records inserted. In the SQL Editor it will be displayed on the progress panel above the Code Editor.

```
   if (ibec_mod(i, 500) = 0) then
   begin
      ibec_progress(i || ´´ records inserted...´´);
```

Don't forget to commit!

```
      commit;
   end
end
```

Once more `COMMIT`. Maybe there are some uncommited `INSERTS`...

```
   commit;
end
```

Let's work with the second connection...

```
use SubscriberDB;
```

If `IBE$$TEST_DATA` doesn't exist in the database we must create it:

```
if (not exists(select rdb$relation_name from rdb$relations where rdb$relation_name =  IBE$$TEST_DATA)) then
begin
execute statement :CreateStmt;
```

Don't forget to commit each DDL statement explicitly!

```
commit;
execute statement :AlterStmt;
commit;
```

The idea is that we fetch the data from the first database and insert it into `IBE$$TEST_TABLE` in the second database:

```
use MasterDB;

i = 0;
k = 0;
```

`FOR ... SELECT` will select data from the first database...

```
for select * from IBE$$TEST_DATA
into vals
do
begin
```

...and we will insert them into the second database:

```
   use SubscriberDB;
   k = k + 1; -- Just a counter...
```

Now we should modify some of the data. Otherwise we"ll have nothing to compare ;-)

```
   if (ibec_mod(k,100) <> 0) then
```

Each hundredth record will be skipped...

```
   begin
      if (ibec_mod(i,10) = 0) then
```

the 8th field of each tenth record will be changed to NULL...

```
            vals[7] = null;
         if (ibec_mod(i,30) = 0) then
```

...and the 10th field of each 30th record will be modified...

```
            vals[9] = ibec_randomstring(500, 1000, 0, 255);
```

Finally insert a record:

```
         insert into SubscriberDB.IBE$$TEST_DATA values :vals;
         i = i + 1;
```

After each 500 inserted records we will display a progress message. We will also commit after every 500 INSERTS:

```
         if (ibec_mod(i, 500) = 0) then
         begin
            ibec_progress(i || ´´records inserted...´´);
            commit;
         end
      end
   end
```

Once again COMMIT...

```
   use SubscriberDB;
   commit;
```

Now we will insert some more data into the second database just to provide further discrepancies between the two tables...

```
   i = k + 1;
   while (i < (RecNum + 1000 + 1)) do
   begin
      fid2    = ibec_randomstring(1,20,65,90);
      fint    = ibec_random2(1, 100000);
      fvarc   = ibec_randomstring(1,100,65,90);
      fdate   = ibec_random2(20000,40000);
      ftime   = ibec_random(0);
      ftimest = ibec_random2(20000,40000) + ibec_random(0);
      fnum    = ibec_random2(1,40000) + ibec_random(0);
      fbool   = ibec_randomval(´´T´´,´´F´´);
      fblob   = ibec_randomstring(500, 1000, 0, 255);
      fseason = ibec_randomval(´´Spring´´, ´´Summer´´, ´´Autumn´´, ´´Winter´´);

      insert into IBE$$TEST_DATA values (:i, :fid2, :fint, :fvarc, :fdate, :ftime, :ftimest, :fnum, :fbool, :fblob, :fseason);

      if (ibec_mod(i, 500) = 0) then
      begin
         ibec_progress(i || ´´records inserted...´´);
         commit;
      end
      i = i + 1;
   end
   commit;
end
```

So, let's begin to compare data. Our goal is make the second IBE$$TEST_DATA a full copy of the first IBE$$TEST_DATA.

First of all we should get the primary key of the reference table:

```
use MasterDB;
   i = 0;
   for select i.rdb$field_name
   from rdb$relation_constraints rc, rdb$index_segments i, rdb$indices idx
   where (i.rdb$index_name = rc.rdb$index_name) and
         (idx.rdb$index_name = rc.rdb$index_name) and
         (rc.rdb$constraint_type = ´´PRIMARY KEY´´) and
         (rc.rdb$relation_name = ´´IBE$$TEST_DATA´´)
   order by i.rdb$field_position
   into fldname
   do
   begin
      PKFields[i] = fldname;
      i = i + 1;
   end
```

Now we need to get a list of remaining fields:

```
   SelStmt = ´´select rdb$field_name
             from rdb$relation_fields
             where (rdb$relation_name = ´´IBE$$TEST_DATA´´)´´;
```

Here we add a condition to exclude primary key fields from the SELECT result:

```
   i = 0;
   HighDim = ibec_high(PKFields);
```

```
   for i = 0 to HighDim do
      SelStmt = SelStmt || ´´ and (rdb$field_name <> ´´ || ibec_trim(PKFields[i]) || ´´)´´;
```

We need the natural order of the fields...

```
   SelStmt = SelStmt || ´´ order by rdb$field_position´´;
```

Finally execute the `SELECT` statement just created and get an array of all non-PK fields:

```
   i = 0;
   for execute statement :SelStmt
   into :s
   do
   begin
```

Trim spaces, we don't need them...

```
      NonPKFields[i] = ibec_trim(:s);
      i = i + 1;
   end
```

Let's compose necessary statements:

```
SelStmt will be used to retrieve data
UpdStmt will be used to update the second table if two records differ:
   SelStmt = ´´´´;
   UpdStmt = ´´update ibe$$test_data set ´´;
   WhereClause = ´´ where ´´;

   HighDim = ibec_high(NonPKFields);
   for i = 0 to HighDim do
   begin
      SelStmt = SelStmt || NonPKFields[i];
      SelStmt = SelStmt || ´´, ´´;
      UpdStmt = UpdStmt || ibec_chr(13) || NonPKFields[i] || ´´ = :´´ || NonPKFields[i];
      if (i < HighDim) then
         UpdStmt = UpdStmt || ´´, ´´;
   end
```

Here we compose a `WHERE` clause with primary key fields: `WHERE (PK_FIELD1 = :PK_FIELD1) AND (PK_FIELD2 = :PK_FIELD2) AND ...`

```
   HighDim = ibec_high(PKFields);
   for i = 0 to HighDim do
   begin
      SelStmt = SelStmt || ibec_trim(PKFields[i]);
      WhereClause = WhereClause || ´´(´´ || ibec_trim(PKFields[i]) || ´´ = :´´ || ibec_trim(PKFields[i]) || ´´)´´;
      if (i < HighDim) then
      begin
         SelStmt = SelStmt || ´´, ´´;
         WhereClause = WhereClause || ´´ and ´´;
      end
   end

   SelStmt = ´´select ´´ || SelStmt || ´´ from IBE$$TEST_DATA order by ´´;

   for i = 0 to HighDim do
   begin
      SelStmt = SelStmt || ibec_trim(PKFields[i]);
      if (i < HighDim) then
         SelStmt = SelStmt || ´´, ´´;
   end

   PKFieldCount = ibec_high(PKFields)+1;
   PKFieldIndex = ibec_high(NonPKFields)+1;

   StartTime = ibec_gettickcount(); -- Note the time...

   MasterCR = ibec_cr_OpenCursor(MasterDB, SelStmt);
   SubscriberCR = ibec_cr_OpenCursor(SubscriberDB, SelStmt);
```

Compose the `INSERT` statement:

```
   InsFields = ´´´´; InsValues = ´´´´;
   FldCount = ibec_cr_FieldCount(SubscriberCR);
   for i = 0 to (FldCount-1) do
   begin
      FldName = ibec_Trim(ibec_cr_FieldName(SubscriberCR, i));
      InsFields = InsFields || FldName;
      InsValues = InsValues || '´´:´´ || FldName;
      if (i < (FldCount-1)) then
      begin
         InsFields = InsFields || ´´, ´´;
         InsValues = InsValues || ´´, ´´;
      end
   end
   InsStmt = ´´insert into ibe$$test_data (´´ || InsFields || ´´) values (´´ || InsValues || ´´)´´;

ibec_UseConnection(SubscriberDB);
```

```
    while (not (ibec_cr_Eof(MasterCR) and ibec_cr_Eof(SubscriberCR))) do
    begin
       CompResult = 0;
       if (ibec_cr_Eof(MasterCR)) then
          CompResult = 1;
       else if (ibec_cr_Eof(SubscriberCR)) then
          CompResult = -1;
       else
       begin
          ibec_cr_Fetch(MasterCR, MasterPK, PKFieldIndex, PKFieldCount);
          ibec_cr_Fetch(SubscriberCR, SubscriberPK, PKFieldIndex, PKFieldCount);
          CompResult = ibec_CmpRecords2(MasterPK, SubscriberPK);
       end

       if (ProcessUpdates and (CompResult = 0)) then
       begin
          ibec_cr_Fetch(MasterCR, MasterVals, 0, PKFieldIndex);
          ibec_cr_Fetch(SubscriberCR, SubscriberVals, 0, PKFieldIndex);
          CompResult = ibec_CmpRecords(MasterVals, SubscriberVals);
          if (CompResult <> -1) then
          begin
             UpdatedRecs = UpdatedRecs + 1;
             ibec_progress(´´Record must be updated...´´);
             ibec_cr_Fetch(MasterCR, MasterVals, 0, null);
             execute statement :UpdStmt || WhereClause values :MasterVals;
          end

          ibec_cr_Next(MasterCR);
          ibec_cr_Next(SubscriberCR);
       end
       else if (ProcessInserts and (CompResult < 0)) then
       begin
```

Redundant master record found. Insert it into the subscriber:

```
          InsertedRecs = InsertedRecs + 1;
          ibec_progress(´´Record must be inserted...´´);
          ibec_cr_Fetch(MasterCR, MasterVals, 0, null);
          execute statement :InsStmt values :MasterVals;
          ibec_cr_Next(MasterCR);
       end
       else if (ProcessDeletes and (CompResult > 0)) then
       begin
```

Redundant subscriber record found. Delete it.

```
          DeletedRecs = DeletedRecs + 1;
          ibec_progress(´´Record must be deleted...´´);
          ibec_cr_Fetch(SubscriberCR, SubscriberPK, PKFieldIndex, PKFieldCount);
          execute statement ´´delete from ibe$$test_data ´´ || WhereClause values :SubscriberPK;
         ibec_cr_Next(SubscriberCR);
       end;
    end

    ibec_cr_CloseCursor(MasterCR);
    ibec_cr_CloseCursor(SubscriberCR);

    commit;
```

Done. Close both connections:

```
    close connection MasterDB;
    close connection SubscriberDB;
```

Let's count the elapsed time...

```
    EndTime = ibec_gettickcount();
    TotalTime = (EndTime - StartTime) / 1000;
    suspend;
 end
```

See also:
Cursor functions

IBEBLOCK and Test Data Generator

The following IBEBlock creates a table named IBE$TEST_DATA and populates it with random data.

```
execute ibeblock
returns (info varchar(100))
as
begin
  RecNum = 10000;

  if (exists (select rdb$relation_name from rdb$relations where rdb$relation_name = 'IBE$$TEST_DATA')) then
  begin
    execute statement 'drop table IBE$$TEST_DATA';
    commit;
  end

 execute statement
  'create table IBE$$TEST_DATA (
     F_INTEGER integer,
     F_VARCHAR varchar(100),
     F_DATE date,
     F_TIME time,
     F_TIMESTAMP timestamp,
     F_NUMERIC numeric(15,2),
     F_BOOL char(1) check (F_BOOL in (´´T´´, ´´F´´)),
     F_BLOB blob sub_type 1,
     F_SEASON varchar(15) check(F_SEASON in (´´Spring´´, ´´Summer´´, ´´Autumn´´, ´´'Winter´´)),
     F_RELS varchar(64))';
  commit;

  StartTime = ibec_gettickcount();

  i = 0;
  for select rdb$relation_name
  from rdb$relations
  into :rel_names
  do
  begin
    rels[i] = :rel_names;
    i = i + 1;
  end

  i = 0;
  while (i < RecNum) do
  begin
    fint    = ibec_random2(1, 100000);
    fvarc   = ibec_randomstring(1,100,65,90);
    fdate   = ibec_random2(20000,40000);
    ftime   = ibec_random(0);
    ftimest = ibec_random2(20000,40000) + ibec_random(0);
    fnum    = ibec_random2(1,40000) + ibec_random(0);
    fbool   = ibec_randomval('T','F');
    fblob   = ibec_randomstring(500, 1000, 0, 255);
    fseason = ibec_randomval('Spring', 'Summer', 'Autumn', 'Winter');
    frel    = rels[ibec_random2(0,ibec_high(rels))];

    insert into IBE$$TEST_DATA values (:fint, :fvarc, :fdate, :ftime, :ftimest, :fnum, :fbool, :fblob, :fseason, :frel);
    i = i + 1;

    if (ibec_mod(i, 500) = 0) then
    begin
      ibec_progress(i || ' records inserted...');
      commit;
    end
  end

  commit;

  EndTime = ibec_gettickcount();
  info = 'Total time: ' || ((EndTime - StartTime) / 1000) || ' seconds';
  suspend;
  info = 'Per record: ' || ((EndTime - StartTime) / 1000 / RecNum) || ' seconds';
  suspend;
end
```

See also:
Test Data Generator

Joining tables from different databases

The following example illustrates how to join two tables from different databases:

```
execute ibeblock (iii integer, ivc varchar(100))
returns (id integer, ename varchar(100), company varchar(100))
as
begin

-- drop database 'localhost/3060:c:\db1.fdb' user 'SYSDBA' password 'masterkey' clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

-- drop database 'localhost/3060:c:\db2.fdb' user 'SYSDBA' password 'masterkey' clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

  create database 'localhost/3060:c:\db1.fdb' user 'SYSDBA' password 'masterkey'
  page_size 4096 sql_dialect 3
  clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

  create database 'localhost/3060:c:\db2.fdb' user 'SYSDBA' password 'masterkey'
  page_size 4096 sql_dialect 3
  clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

  create connection db1 dbname 'localhost/3060:c:\db1.fdb'
  password 'masterkey' user 'SYSDBA'
  clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

  create connection db2 dbname 'localhost/3060:c:\db2.fdb'
  password 'masterkey' user 'SYSDBA'
  sql_dialect 3
  clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

  use db1;

  vstmt = 'create table "employees" ( ' || '
     id integer not null primary key,
     full_name varchar(100),
     company_id integer)';

  execute statement :vstmt;

  commit;

  use default;

  select count(*) from help_items into :icount;

  use db1;

  insert into "employees" (id, full_name, company_id) values (1, 'Alexander Khvastunov', 2);
  insert into "employees" (id, full_name, company_id) values (2, 'Bill Gates', 1);
  insert into "employees" (id, full_name, company_id) values (3, 'John Doe', NULL);
  insert into "employees" (id, full_name, company_id) values (4, 'Vladimir Putin', 3);
  insert into "employees" (id, full_name, company_id) values (5, 'Somebody', 15);

  use db2;

  execute statement
  'create table companies (
     id integer not null primary key,
     company_name varchar(100))';

  commit;

  insert into companies (id, company_name) values (1, 'Microsoft');
  insert into companies (id, company_name) values (2, 'HK-Software');
  insert into companies (id, company_name) values (3, 'The Kremlin?');

  commit;

  use db1;

  for execute statement 'select id, full_name, company_id from "employees"'
  into :id, :ename, :cid
  do
  begin
    use db2;

    company = NULL;

    select company_name from companies
    where id = :cid
    into :company;

    suspend;
  end

  close connection db1;
  close connection db2;
end
```

Recreating indices 1

The following example illustrates how to recreate database indices:

```
execute ibeblock
returns (info varchar(1000))
as
begin
  i = 0;
  for select i.rdb$index_name, i.rdb$relation_name, i.rdb$unique_flag,
             i.rdb$index_inactive, i.rdb$index_type
      from rdb$indices i
      left join rdb$relation_constraints rc on (i.rdb$index_name = rc.rdb$index_name)
      where (i.rdb$system_flag is null) and (rc.rdb$index_name is null)
      into :IdxName, :IdxRelName, :IdxUnique, :IdxInactive, :IdxType
  do
  begin
    sFields = ´´;
    for select rdb$field_name from rdb$index_segments
        where rdb$index_name = :IdxName
        order by rdb$field_position
        into :ifields
    do
    begin
      if (sFields <> ´´) then
        sFields = sFields || ', ';
      sFields = sFields || ibec_formatident(ibec_trim(ifields));
    end
    DropStmt[i] = 'drop index ' || ibec_formatident(ibec_trim(IdxName));
    CreateStmt[i] = 'create ' || ibec_iif(IdxUnique = 1, 'unique ', ´´) || ibec_iif(IdxType = 1, 'descending ', ´´) ||
                    ' index ' || ibec_formatident(ibec_trim(IdxName)) ||
                    ' on ' || ibec_formatident(ibec_trim(IdxRelName)) || ' (' || sFields || ')';
    i = i + 1;
  end
  i = 0;
  while (i <= ibec_high(DropStmt)) do
  begin
    s = DropStmt[i];
    info = s;
    suspend;
    ibec_progress(info);
    execute statement :s;
    commit;

    s = CreateStmt[i];
    info = s;
    suspend;
    ibec_progress(info);
    execute statement :s;
    commit;

    i = i + 1;
  end
end
```

See also:
Firebird for the Database Expert: Episode 1 - Indexes
Recreating Indices 2

Recreating indices 2

The following example illustrates how to recreate database indices using AS DATASET:

```
execute ibeblock
returns (info varchar(1000))
as
begin
  select i.rdb$index_name, i.rdb$relation_name, i.rdb$unique_flag,
         i.rdb$index_inactive, i.rdb$index_type
  from rdb$indices i
  left join rdb$relation_constraints rc on (i.rdb$index_name = rc.rdb$index_name)
  where (i.rdb$system_flag is null) and (rc.rdb$index_name is null)
  as dataset ds_indices;

  while (not ibec_ds_eof(ds_indices)) do
  begin
    IdxName = ibec_trim(ibec_ds_getfield(ds_indices,0));
    IdxRelName = ibec_trim(ibec_ds_getfield(ds_indices,1));
    IdxUnique = ibec_ds_getfield(ds_indices,2);
    IdxInactive = ibec_ds_getfield(ds_indices,3);
    IdxType = ibec_ds_getfield(ds_indices,4);

    sFields = ´´;
    for select rdb$field_name from rdb$index_segments
        where rdb$index_name = :IdxName
        order by rdb$field_position
        into :IdxField
    do
    begin
      IdxField = ibec_trim(IdxField);
      if (sFields <> ´´) then
        sFields = sFields || ', ';
      sFields = sFields || ibec_formatident(IdxField);
    end

    DropStmt   = 'drop index ' || ibec_formatident(IdxName);
    CreateStmt = 'create ' || ibec_iif(IdxUnique = 1, 'unique ', ´´) || ibec_iif(IdxType = 1, 'descending ', ´´) ||

                 ' index ' || ibec_formatident(IdxName) ||
                 ' on ' || ibec_formatident(IdxRelName) || ' (' || sFields || ')';

    info = DropStmt;
    suspend;
    ibec_progress(info);
    execute statement :DropStmt;
    commit;

    info = CreateStmt;
    suspend;
    ibec_progress(info);
    execute statement :CreateStmt;
    commit;

    ibec_ds_next(ds_indices);
  end

  close dataset ds_indices;
end
```

See also:
Firebird for the Database Expert: Episode 1 - Indexes
Recreating Indices 1

Building an OLAP cube

The following illustrates the construction of an OLAP cube:

```
execute ibeblock
as
begin
  SelectSQL = 'select rf.rdb$relation_name, f.rdb$field_type, f.rdb$field_length, f.rdb$field_precision
               from rdb$relation_fields rf, rdb$fields f
               where rf.rdb$field_source = f.rdb$field_name';

  vDimensions[0] = 'FieldName=RDB$RELATION_NAME; Alias="Table Name"';
  vDimensions[1] = 'FieldName=RDB$FIELD_TYPE; Alias="Field Type';

  vMeasures[0] = 'FieldName=RDB$FIELD_TYPE; Alias="Field Count"; CalcType=ctCount; Format=0';
  vMeasures[1] = 'FieldName=RDB$FIELD_LENGTH; Alias="Total Length"; CalcType=ctSum; Format=0';
  vMeasures[2] = 'FieldName=RDB$FIELD_PRECISION; Alias="Avg Precision"; CalcType=ctAverage';
```

Build and save cube in binary format:

```
  ibec_BuildCube('C:\test_cub.cub', SelectSQL, vDimensions, vMeasures, null);
```

Build and save cube in XML format:

```
  ibec_BuildCube('C:\test_cub.xml', SelectSQL, vDimensions, vMeasures, null);
end
```

Inserting files into a database

IBEBlock can be used to insert files extremely simply and quickly into your database:

```
execute ibeblock
as
begin
  MyVar = ibec_LoadFromFile(C:\f.jpg);
  insert into ... values (..., :MyVar);
  commit;
end
```

Another possible way is to use different `SET BLOBFILE` statements before each `INSERT/UPDATE` statement:

```
SET BLOBFILE 'C:\f.jpg';
INSERT INTO ... VALUES (..., :h00000000_FFFFFFFF);
SET BLOBFILE 'C:\f2.jpg';
INSERT INTO ... VALUES (..., :h00000000_FFFFFFFF);
SET BLOBFILE 'C:\f3.jpg';
INSERT INTO ... VALUES (..., :h00000000_FFFFFFFF);
```

See also:
Inserting file data into a database

Inserting file data into a database

The following script should be executed in the IBExpert Script Executive or with IBEScript.

```
set names win1251;
set sql dialect 3;
set clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

create database 'localhost/3060:D:\allscripts.fdb'
user 'SYSDBA' password 'masterkey'
page_size 8192 default character set WIN1251;

create generator gen_script_id;

create table scripts (
  ID INTEGER NOT NULL PRIMARY KEY,
  FILENAME VARCHAR(2000),
  SCRIPT_TEXT BLOB SUB_TYPE TEXT);

create trigger script_bi for scripts
active before insert position 0
as
begin
  if (new.id is null) then
    new.id = gen_id(gen_script_id, 1);
end;

execute ibeblock
as
begin
  ibec_progress('Searching for script files...');
  files_count = ibec_getfiles(files_list, 'D:\', '*.sql',_gfRecursiveSearch + __gfFullName);

  if (files_count > 0) then
  begin
    i = 0;
    while (i < ibec_high(files_list)) do
    begin
      file_name = files_list[i];
      file_size = ibec_filesize(file_name) / 1024 / 1024; -- File size in megabytes
      if (file_size < 10) then
      begin
        script_data = ibec_loadfromfile(file_name);
        ibec_progress('Adding script file ' || :file_name);
        insert into scripts (filename, script_text) values (:file_name, :script_data);
        commit;
      end
      i = i + 1;
    end
  end
end;
```

See also:
Inserting files into a database

Importing data from a CSV file

The following example creates a simple comma-separated values (CSV) file and imports its data into a database:

```
execute ibeblock
  returns (outstr varchar(100))
  as
  begin
```

First, let's create a simple CSV-file with some data:

```
    FS = ibec_fs_OpenFile('C:\MyData.csv', __fmCreate);
    if (not FS is null) then
    begin
      s = '1:John:Doe:M';
      ibec_fs_Writeln(FS, s);
      s = '2:Bill:Gates:M';
      ibec_fs_Writeln(FS, s);
      s = '3:Sharon:Stone:F';
      ibec_fs_Writeln(FS, s);
      s = '4:Stephen:King:M';
      ibec_fs_Writeln(FS, s);
      ibec_fs_CloseFile(FS);
    end
```

If table `IBE$$TEST_PEOPLE` exists we'll drop it:

```
    if (exists(select rdb$relation_name from rdb$relations where rdb$relation_name = 'IBE$$TEST_PEOPLE')) then
    begin
      s = 'DROP TABLE IBE$$TEST_PEOPLE';
      execute statement s;
      commit;
    end
```

Let's create a new table that will store the imported data:

```
    s = 'CREATE TABLE IBE$$TEST_PEOPLE (
         ID integer,
         FIRST_NAME varchar(50),
         LAST_NAME varchar(50),
         SEX varchar(1))';
    execute statement s;
    commit;

    i = 0; (-- Just a counter of inserted records)
    FS = ibec_fs_OpenFile('C:\MyData.csv', __fmOpenRead);
    if (not FS is null) then
    begin
      while (not ibec_fs_Eof(FS)) do
      begin
        s = ibec_fs_Readln(FS);
        ValCount = ibec_ParseCSVLine(Vals, s, ´´, ':', __csvEmptyStringAsNull);
        INSERT INTO IBE$$TEST_PEOPLE (ID, FIRST_NAME, LAST_NAME, SEX) VALUES :Vals;
        commit;
        i = i + 1;
      end
      ibec_fs_CloseFile(FS);
    end

    outstr = i || ' records inserted into IBE$$TEST_PEOPLE';
    suspend;
  end
```

See also:
Create multiple CSV files from a script
Import CSV Files
`INSERTEX` (CSV file import)

Importing data from a file

1. Load the script into the Script Executive.
2. Make any necessary modifications.
3. Press [F9] to execute the script.

Script

```
set names win1251;
set sql dialect 3;
set clientlib 'C:\Program Files\Firebird\bin\fbclient.dll';

create database 'localhost/3060:D:\allscripts.fdb'
user 'SYSDBA' password 'masterkey'
page_size 8192 default character set WIN1251;

create generator gen_script_id;

create table scripts (
  ID INTEGER NOT NULL PRIMARY KEY,
  FILENAME VARCHAR(2000),
  SCRIPT_TEXT BLOB sub_type text);

create trigger script_bi for scripts
active before insert position 0
as
begin
  if (new.id is null) then
    new.id = gen_id(gen_script_id, 1);
end;

execute ibeblock
as
begin
  ibec_progress('Searching for script files...');
  files_count = ibec_getfiles(files_list, 'D:\', '*.sql', __gfRecursiveSearch + __gfFullName);

  if (files_count > 0) then
  begin
    i = 0;
    while (i < ibec_high(files_list)) do
    begin
      file_name = files_list[i];
      if (ibec_filesize(file_name) < 10240000) then
      begin
        script_data = ibec_loadfromfile(file_name);
        ibec_progress('Adding script file ' || :file_name);
        insert into scripts (filename, script_text) values (:file_name, :script_data);
        commit;
      end
      i = i + 1;
    end
  end
end;
```

Export data into DBF

The following illustrates use of the `SELECT ... EXPORT AS ... INTO` function:

```
execute ibeblock
as
begin
  SELECT * FROM RDB$FIELDS
  EXPORT AS DBF INTO 'E:\TestExport.dbf'
  OPTIONS 'ConvertToDOS; LongStringsToMemo; DateTimeAsDate';
```

Creating a script from a Database Designer model file

The following IBEBlock illustrates how to create a script from a Database Designer Model file:

```
execute ibeblock
as
begin
  FileName = 'C:\model.grc';
  if ibec_FileExists(FileName) then
    ibec_CreateModelScript(FileName, 'C:\model.sql', __msoDontQuoteIdents + __msoIncludeDescriptions);
end
```

Creating an UPDATE script with domain descriptions

The following IBEBlock creates a script with UPDATE statements for all database domains that have a description:

```
execute ibeblock
  as
  begin
    FHSQL = ibec_fs_OpenFile('E:\DomDescs.sql', __fmCreate);
    FHBlobs = ibec_fs_OpenFile('E:\DomDescs.lob', __fmCreate);
    if ((not FHSQL is null) and (not FHBlobs is null)) then
    begin
      ibec_fs_Writeln(FHSQL, 'SET BLOBFILE ´´E:\DomDescs.lob´´;');
      ibec_fs_Writeln(FHSQL, ´´);
      for select rdb$field_name, rdb$description
          from rdb$fields
          where (rdb$description is not null)
          order by 1
          into :FieldName, :FieldDesc
      do
      begin
        if (FieldDesc <> ´´) then
        begin
          FieldName = ibec_Trim(FieldName);
          iOffs = ibec_fs_Position(FHBlobs);
          iLen = ibec_fs_WriteString(FHBlobs, FieldDesc);
          sParamName = ':h' || ibec_IntToHex(iOffs, 8) || '_' || ibec_IntToHex(iLen, 8);
          UpdStmt = 'UPDATE RDB$FIELDS' || ibec_Chr(13) || ibec_Chr(10) ||
                    'SET RDB$DESCRIPTION = ' || :sParamName ||
                    ibec_Chr(13) || ibec_Chr(10) ||
                    'WHERE (RDB$FIELD_NAME = ´´' || FieldName || ´´');';
          ibec_fs_Writeln(FHSQL, UpdStmt);
          ibec_fs_Writeln(FHSQL, ´´);
        end
      end
      ibec_fs_Writeln(FHSQL, 'COMMIT WORK;');
      ibec_fs_CloseFile(FHSQL);
      ibec_fs_CloseFile(FHBlobs);
    end
    commit;
  end;
```

IBEBlock User Forms

1. Copy all IBEBlocks into a separate directory.
2. Open `TableDDL.ibeblock` and change the path to `FldTypeHTML.ibeblock` in the first statement.
3. Load `RunMe.ibeblock` into the SQL Editor.
4. Press [F9] to execute the block.

The sample IBEBlocks include:

- `FldTypeHTML.ibeblock`
- `InputForm.ibeblock`
- `TableDDL.ibeblock`
- `RunMe.ibeblock`

See also:
User Form functions

FldTypeHTML.ibeblock

```
execute ibeblock (
   FType integer,
   FSubType integer,
   FLen integer,
   FScale integer,
   FSegmentSize integer,
   FCharLen integer,
   FPrecision integer,
   SQLDialect integer = 3)
returns (TypeAsString varchar(200))
as
begin
   TypeAsString = ´´;
   if ((FCharLen = 0) or (FCharLen is NULL)) then
      FCharLen = FLen;

   if (FType = 261) then
      TypeAsString = '<B>BLOB SUB_TYPE</B> ' || FSubType || ' <B>SEGMENT SIZE</B> ' || FSegmentSize;
   else if (FType = 14) then
      TypeAsString = '<B>CHAR</B>(' || FCharLen || ')';
   else if (FType = 37) then
      TypeAsString = '<B>VARCHAR</B>(' || FCharLen || ')';
   else if (FType = 12) then
      TypeAsString = '<B>DATE</B>';
   else if (FType = 13) then
      TypeAsString = '<B>TIME</B>';
   else if (FType = 35) then
   begin
      if (SQLDialect = 3) then
         TypeAsString = '<B>TIMESTAMP</B>';
      else
         TypeAsString = '<B>DATE</B>';
   end
   else if (FType in (7, 8, 27, 16)) then
   begin
      if ((FScale < 0) or (FSubType = 1) or (FSubType = 2)) then
      begin
         if (FSubType = 2) then
            TypeAsString = '<B>DECIMAL</B>';
         else
            TypeAsString = '<B>NUMERIC</B>';

         sPrec = FPrecision;
         if (FPrecision is NULL) then
         begin
            if (FType = 7) then
               sPrec = '4';
            else if (FType = 8) then
               sPrec = '9';
            else if (FType = 27) then
               sPrec = '15';
            else if (FType = 16) then
               sPrec = '18';
         end
         else
         sPrec = FPrecision;
            TypeAsString = TypeAsString || '(' || sPrec || ',' || (FScale * -1) || ')';
      end
      else if (FType = 7) then
         TypeAsString = '<B>SMALLINT</B>';
      else if (FType = 8) then
         TypeAsString = '<B>INTEGER</B>';
      else if (FType = 27) then
         TypeAsString = '<B>DOUBLE PRECISION</B>';
      else if (FType = 16) then
         TypeAsString = '<B>BIGINT</B>';
   end
   else if (FType = 10) then
      TypeAsString = '<B>FLOAT</B>';
   suspend;
end
```

See also:
User Form functions

InputForm.ibeblock

```
execute ibeblock (
returns (htmlpage blob)
as
begin
   htmlpage = '<SCRIPT> function ShowDDL(){location.href = "TableChanged"
this.focus()}</SCRIPT>';
   htmlpage = htmlpage || '<P>Select a table from the list below to get its DLL:</P>
               <SELECT ID="TableSelect" OnChange="ShowDDL()">';
   for select rdb$relation_name, rdb$relation_id from rdb$relations
       order by rdb$relation_name
       into :rel_name, :rel_id
   do
   begin
      rel_name = ibec_Trim(rel_name);
      htmlpage = htmlpage || ibec_chr(13) || ibec_chr(10) || ' <option value="' || :rel_id || '">' || rel_name || '</OPTION>';
   end
   htmlpage = htmlpage || ibec_chr(13) || ibec_chr(10) || '</SELECT>';
   htmlpage = htmlpage || '<P></P><P ID="FAKE">';
end
```

See also:
User Form functions

TableDDL.ibeblock

```
execute ibeblock (
   Frm variant,
   Op variant)
as
begin
   FldType = ibec_LoadFromFile('E:\IBEBlocks\FldTypeHTML.ibeblock');

   TableID = ibec_uf_GetElementAttribute(Frm, 'TableSelect', 'value', 0);
   sDDL = '';
   if (TableID is not null) then
   begin
      select rdb$relation_name from rdb$relations where rdb$relation_id = :TableID into :sTableName;
      sTableName = ibec_trim(sTableName);
      sDDL = '<B>CREATE TABLE</B> ' || sTableName || '(' || ibec_Chr(13) || ibec_chr(10);

      for select rf.rdb$field_name, rf.rdb$field_source, rf.rdb$field_position,
                 f.rdb$field_type, f.rdb$field_length, f.rdb$field_scale,
                 f.rdb$field_sub_type, f.rdb$field_precision, f.rdb$character_length,
                 f.rdb$segment_length, rf.rdb$null_flag, chr.rdb$character_set_name
         from rdb$relation_fields rf, rdb$relations r, rdb$fields f
         left join rdb$character_sets chr on (f.rdb$character_set_id = chr.rdb$character_set_id)
         where  (rf.rdb$relation_name = r.rdb$relation_name) and
                (rf.rdb$field_source = f.rdb$field_name) and
                (r.rdb$relation_id = :TableID)
         order by 2
         into :FieldName, :fDomain, :FieldPos, :fType, :fLen, :fScale, :fSubType, :fPrec, fCharLen,
:fSegLen, :fNullFlag, :fCharset
      do
      begin
         sType = ibec_trim(fDomain);
         IsDomainBased = FALSE;
         if (ibec_Copy(sType, 1, 4) <> 'RDB$') then
            IsDomainBased = TRUE;
         execute ibeblock FldType(:fType, :fSubType, :fLen, :fScale, :fSegLen, :fCharLen, :fPrec, 3)
                  returning_values :FieldType;
         sType = ibec_IIF(IsDomainBased, sType, FieldType);

         if (fNullFlag = 1) then
            sType = sType || ' <B>NOT NULL</B>';

         if (((fType = 37) or (fType = 14) or (fType = 261)) and (IsDomainBased = FALSE) and (fCharset is not NULL)) then
         begin
            sType = sType || ' <B>CHARACTER SET</B> ' || ibec_trim(fCharset);
         end
         sType = ibec_IIF(IsDomainBased, sType || ' <I>/* ' || FieldType || ' */</I> ', sType);
         sDDL = sDDL || ' ' || ibec_trim(FieldName) || ' ' || sType || ',' || ibec_Chr(13) || ibec_chr(10);
         suspend;
      end
      iLen = ibec_Length(sDDL) - 3;
      sDDL = ibec_Copy(sDDL, 1, iLen);
      sDDL = sDDL || ');';
   end

   OldData = ibec_uf_GetFormData(Frm);
   iPos = ibec_Pos('<P ID="FAKE">', OldData);
   if (iPos > 0) then
      OldData = ibec_Copy(OldData, 1, iPos + 12);

   sDDL = OldData || '<P>The DDL of the selected table is:</P><P></P><PRE>' || sDDL || '</DDL>';

   ibec_uf_SetFormData(Frm, sDDL);
   ibec_uf_SetElementAttribute(Frm, 'TableSelect', 'value', TableID, 0);
end
```

See also:
User Form functions

RunMe.ibeblock

```
execute ibeblock (
   CodeDir varchar(1000) = 'E:\IBEBlocks\' comment 'Path to necessary IBEBlocks')
as
begin

   FrmBlock = ibec_LoadFromFile(CodeDir || 'TableDDL.ibeblock');

   Block1 = ibec_LoadFromFile(CodeDir || 'InputForm.ibeblock');
   execute ibeblock Block1 returning_values :MyPage;

   MyFrm = ibec_uf_CreateForm(MyPage);
   if (MyFrm is not null) then
   begin
      Res = ibec_uf_ShowForm(MyFrm, 'Caption="Select table from the list below"; Top=100; Height=600; BarTitle="Super Puper
Form!"', FrmBlock);
   end
end
```

See also:
User Form functions

Performing a daily backup of the IBExpert User Database

The following example demonstrates the usage of ibec_reg_xxx functions to perform a daily backup of the IBExpert User Database:

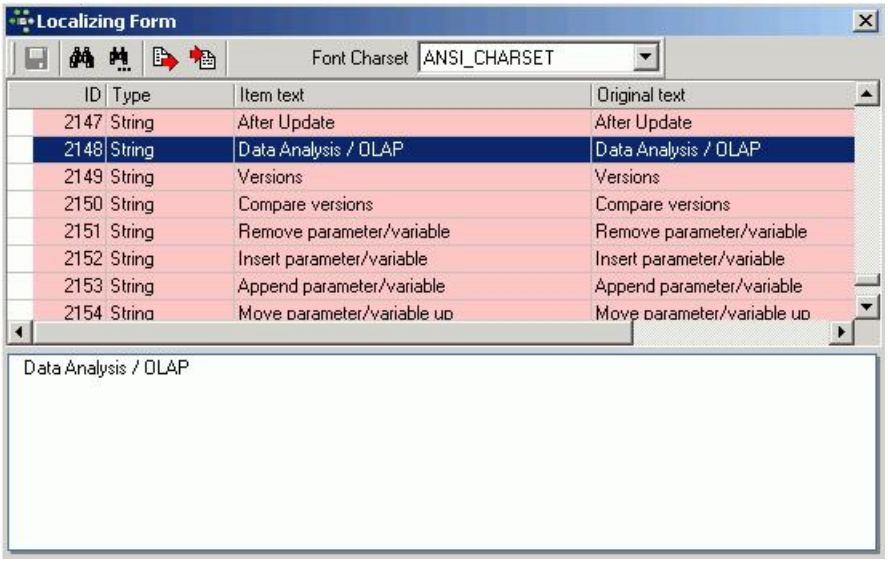
```
execute ibeblock
as
begin
  CurrentDate = ibec_Date();

  reg = ibec_reg_Open(__HKEY_CURRENT_USER, 0);
  try
    if (ibec_reg_OpenKey(reg, 'Software\HK Software\IBExpert\CurrentData', FALSE)) then
    begin
      try
        UDBLastBackupDate = ibec_reg_ReadDate(reg, 'UDBLastBackupDate');
        if (UDBLastBackupDate = CurrentDate) then
          Exit;
      except
      end;
      UDBConnectString = ibec_reg_ReadString(reg, 'UDBConnectString');
      UDBClientLib = ibec_reg_ReadString(reg, 'UDBClientLib');
      UDBUserName = ibec_reg_ReadString(reg, 'UDBUserName');
      UDBPassword = ibec_reg_ReadString(reg, 'UDBPassword');
    end
  finally
    ibec_reg_Close(reg);
  end;

  if ((UDBConnectString is null) or (UDBConnectString = ´´)) then
    Exit;

  ibec_Progress('Starting backup of IBExpert User Database...');
  BackupDir = 'D:\Backups\IBExpert User Database\';
  ibec_ForceDirectories(BackupDir);

  ibec_DecodeDate(CurrentDate, iYear, iMonth, iDay);
  BackupFileName = BackupDir || iDay || ' || iMonth || ' || iYear || '.fbk';

  res = ibec_BackupDatabase(UDBConnectString, BackupFileName,
                            'ClientLib=' || UDBClientLib || '; Password=' ||
                                UDBPassword || '; User=' || UDBUserName,
                            null);

  if (ibec_FileExists(BackupFileName)) then
  begin
    ibec_Progress('Compressing ' || BackupFileName || '...');
    res = ibec_Exec('"C:\Program Files\WinRAR\rar.exe" a "' || BackupFileName || '.rar" "' ||
                    BackupFileName || '" -m5 -ri1', ´´, null);
    if (res = 0) then
      ibec_DeleteFile(BackupFileName);
  end

  if (res = 0) then
  begin
    reg = ibec_reg_Open(__HKEY_CURRENT_USER, 0);
    try
      if (ibec_reg_OpenKey(reg, 'Software\HK Software\IBExpert\CurrentData', FALSE)) then
        ibec_reg_WriteDate(reg, 'UDBLastBackupDate', CurrentDate);
    finally
      ibec_reg_Close(reg);
    end;
  end
end
```

Disable and enable IBExpert features

Using this feature it is possible to disable all menu items, and blend only those in, which you wish the user to see. A particularly useful security feature!

```
execute ibeblock as begin
 ibec_DisableFeature(0);     --disable all
 ibec_EnableFeature(1003);  --enable Tools menu
 ibec_EnableFeature(2148);  --enable menuitem tools-data analysis
end
```

The example above enables only the IBExpert Tools menu item, Data Analysis. The numbers quoted directly after the IBEBlock keyword can be found in the IBExpert Tools menu, Localize IBExpert.


Localizing Form
Font Charset
ANSI_CHARSET
ID
Type
Item text
Original text
2147 String After Update After Update
2148 String Data Analysis / OLAP Data Analysis / OLAP
2149 String Versions Versions
2150 String Compare versions Compare versions
2151 String Remove parameter/variable Remove parameter/variable
2152 String Insert parameter/variable Insert parameter/variable
2153 String Append parameter/variable Append parameter/variable
2154 String Move parameter/variable up Move parameter/variable up
Data Analysis / OLAP

Retrieve all valid e-mail addresses from an input text

This IBEBlock retrieves all valid e-mail addresses from an input text (any_text):

```
execute ibeblock (any_text varchar(10000))
returns (email varchar(100))
as
begin
  re = ibec_re_Create('[_a-zA-Z\d\-\.]+@[_a-zA-Z\d\-]+(\.[_a-zA-Z\d\-]+)+');
  try
    Res = ibec_re_Exec(re, any_text);
    while (Res) do
    begin
      email = ibec_re_Match(re, 0);
      suspend;
      Res = ibec_re_ExecNext(re);
    end
  finally
    ibec_re_Free(re);
  end
end
```

Working with POP3 servers

The following is an example of using the Functions for working with POP3 servers:

```
execute ibeblock
as
begin
  CRLF = ibec_CRLF();

  ses = ibec_pop3_OpenSession('Host=mypop3.com; User=iam; Pass=12345');
  try
    --Alternative way to set pop3 session properties:
    --sHost = ibec_pop3_SetProperty(ses, 'Host', 'mypop3.com');
    --sUser = ibec_pop3_SetProperty(ses, 'UserName', 'iam');
    --sPass = ibec_pop3_SetProperty(ses, 'Password', '12345');
    --sPort = ibec_pop3_SetProperty(ses, 'Port', 'pop3');

    ibec_Progress('Connecting to mypop3...');
    if (ibec_pop3_ConnectAndAuth(ses)) then
    begin
      ibec_Progress('Retrieving Uidl...');
      Res = ibec_pop3_Uidl(ses);
      sResp = ibec_pop3_GetProperty(ses, 'Uidl');

      UidlItems = ibec_Explode(CRLF, sResp);
      foreach (UidlItems as UID key Idx skip nulls) do
      begin
        if (UID = '') then
          Continue;
        UidData = ibec_Explode(' ', UID);
        iMsgNum = ibec_Cast(UidData[0], __typeInteger);
        ibec_Progress('Getting message ' + UidData[1] + '...');
        Res = ibec_pop3_Retr(ses, iMsgNum);
        if (Res) then
        begin
          ibec_ForceDirectories('D:\Mails');
          MsgData = ibec_pop3_GetProperty(ses, 'MsgData');
          ibec_SaveToFile('D:\Mails\' + UidData[1], MsgData, 0);
        end;
      end;
    end;
    ibec_Progress('Quit...');
    ibec_pop3_Quit(ses);
  finally
    ibec_pop3_CloseSession(ses);
  end;
end;
```

See also:
Functions for working with POP3 servers

IBExpertWebForms - The First Steps

- What is required for using IBExpertWebForms?
- How do I set up the database?
- Which control elements are available in IBExpertWebForms?
- How do I insert control elements in my IBExpertWebForm?
- How do I create an event?
- How do I handle the database components?
- You would like more examples?

IBExpertWebForms Tutorials

If you are new to IBExpertWebForms, then the first three tutorials should help you get started:

- IBExpertWebForms Tutorial Lesson 1 - Installation
- IBExpertWebForms Tutorial Lesson 2 - My First WebForm
- IBExpertWebForms Tutorial Lesson 3 - Database driven WebForms

What is required for using IBExpertWebForms?

Since IBExpert version 2008.01.28 all IBExpert fully licensed versions, i.e. single, multiple, Site, Junior VAR and full VAR licenses, include our fully integrated IBExpertWebForms module.

If you have a customer version of IBExpert, you are allowed to use IBExpertWebForms on your registered computer. If you have a Site License, you can use IBExpertWebForms on any computer in your company. If you have a VAR or Junior VAR License, you are allowed to distribute IBExpertWebForms together with your applications to your customers.

For details about purchasing or upgrading any of the IBExpert customer versions, please refer to http://ibexpert.net/ibe/pmwiki.php?n=Main.IBExpertLicenses.

With IBExpertWebforms you can create database-based web applications. Just place your VCL components in the integrated *Form Designer*, connect them with your tables or queries as a data source using the integrated *Object Inspector*, and create your events as stored procedures inside your Firebird or InterBase database.

The result is handled by a PHP script, which is used by the Apache web server on Windows, Linux or any other operating system which supports Apache, PHP and Firebird or InterBase.

The main advantage: you do not need any know-how regarding Java script, HTML, Ajax, PHP, etc. to create your database web application. All operations are done inside your database and you just need to learn some very simple extensions and rules based on your existing Firebird and InterBase knowledge.

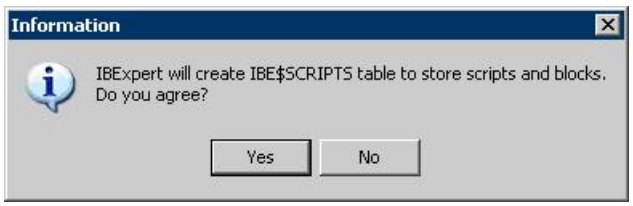
How do I set up the database?

You can use IBExpertWebForms with any InterBase (6.0-2007) or Firebird Database (1.0-2.1). Everything you need is automatically installed with the IBExpert Trial or IBExpert Customer Version. This includes a fully functional Apache Web Server and PHP5.

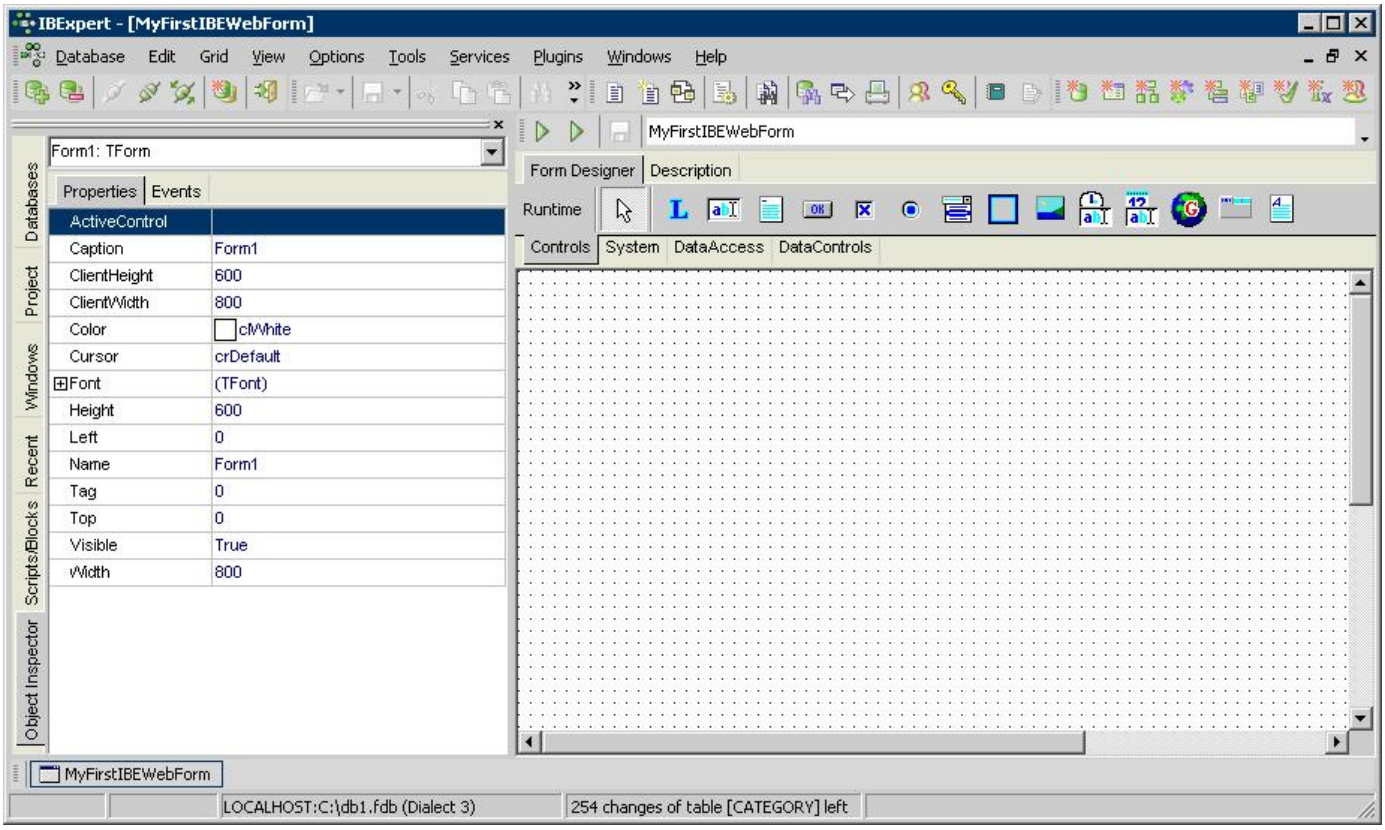
The following example is shown based on the IBExpertDemoDatabase, which can be found in `C:\Program Files\HK-Software\IBExpert Demo Databases\db1.sql`.

If you want to create the same database, please copy `rfunc.dll` from this directory to your Firebird `UDF` directory before executing the `db1.sql` script in IBExpert Tools menu / Script Executive. To create demo data in the database, execute the procedure initially with the parameter `10000`.

After registering and opening your database in IBExpert, click with the right mouse button on the database *Scripts* node and select *New WebForm*. Confirm the following dialog for creating the script table inside the database automatically.



This opens the *Form Designer*. First of all you should allocate a new name (in the top right-hand corner of the dialog) for the IBExpertWebForm (e.g. *MyFirstIBEWebForm*). Any alterations can be saved using the *Save* button.



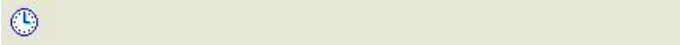
Here you can see the *Form Designer*. This allows you to add several components to your IBExpertWebForm application. If you already have experience with an environment such as Delphi or VB, you will see that it is very similar.

Which control elements are available in IBExpertWebForms?

The IBExpertWebForms *Form Designer* has a component-oriented structure (similar to Delphi). Each component can be selected in the *Form Designer* and positioned in the form itself. The components are grouped in four categories: *Standard*, *System*, *DataAccess* and *DataControls*.

Standard

System



ataAccess

ataControls

Under *Standard* you can find all common components for the display of texts, control elements for text input and selection elements, a *PageControl* for the administration of multi-page display areas, as well as a control element for the input of formatted texts (similar to WordPad). Under *System*, there is a *Timer*, which can trigger an event at regular intervals.

Under *DataAccess* and *DataControls* you can find all components that work together with the database. The *DataAccess* components are pure database components, such as *Database Connection*, *Transaction Control*, *Dataset* and *Datasource*. *DataControls* contains all visual database components. These components can be used to display and modify database contents.

How do I insert control elements in my IBExpertWebForm?

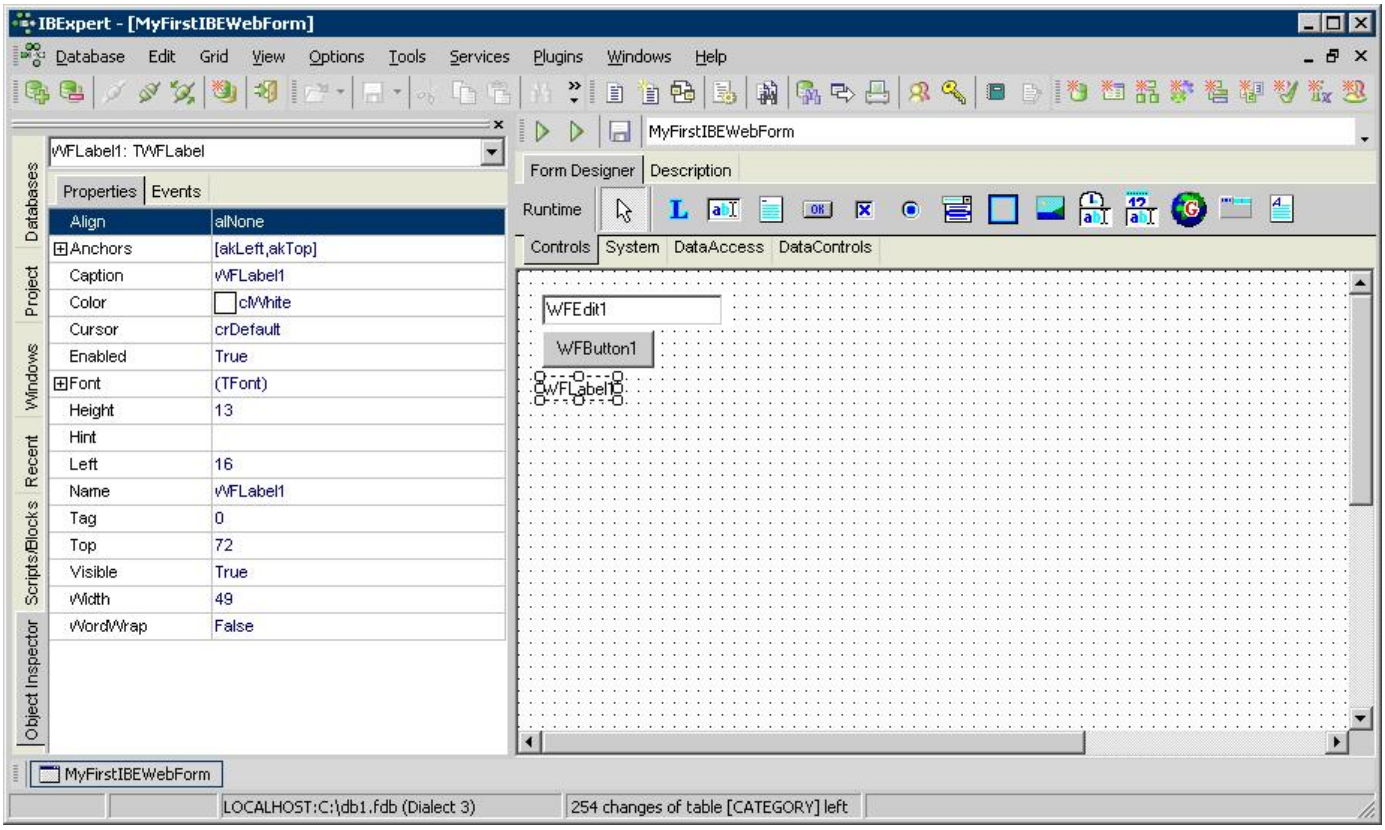
Important tips for the use of the Form Designer and the Object Inspector

As IBExpertWebForms is currently still in the development phase we would like to point out two problems:

1. When you click on a component in the *Form Designer*, in order to, for example, edit a property, the cursor may occasionally get "caught" on the mouse cursor (recognizable by the thick black frame representation of the component, as is the case with drag 'n' drop operations). Should this occur simply click the mouse a second time.
2. When a property or event in the *Object Inspector* is altered or created, the value is not immediately saved. A previously edited value is only saved when you immediately briefly click on any other property directly after editing.

Click on the desired component, using the tabs *Standard*, *System*, *DataAccess* and *DataControls* in the *Form Designer*. Then click with the mouse on the IBExpertWebForm where you wish this component to be positioned.

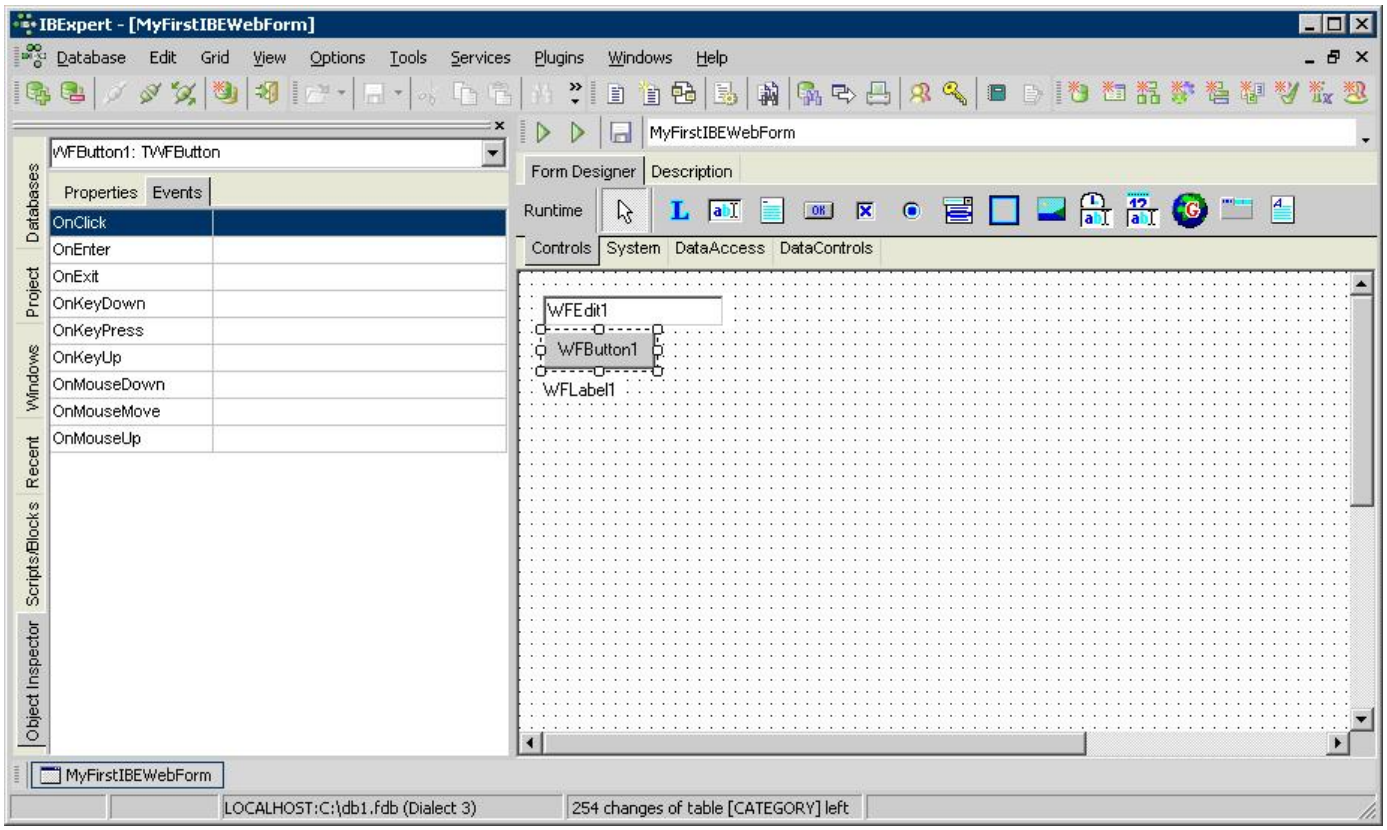
The first example shows the typical “Hello, World” Application. So we need 3 components, a *TWFEdit*, a *TWFButton* and a *TWFLabel*.



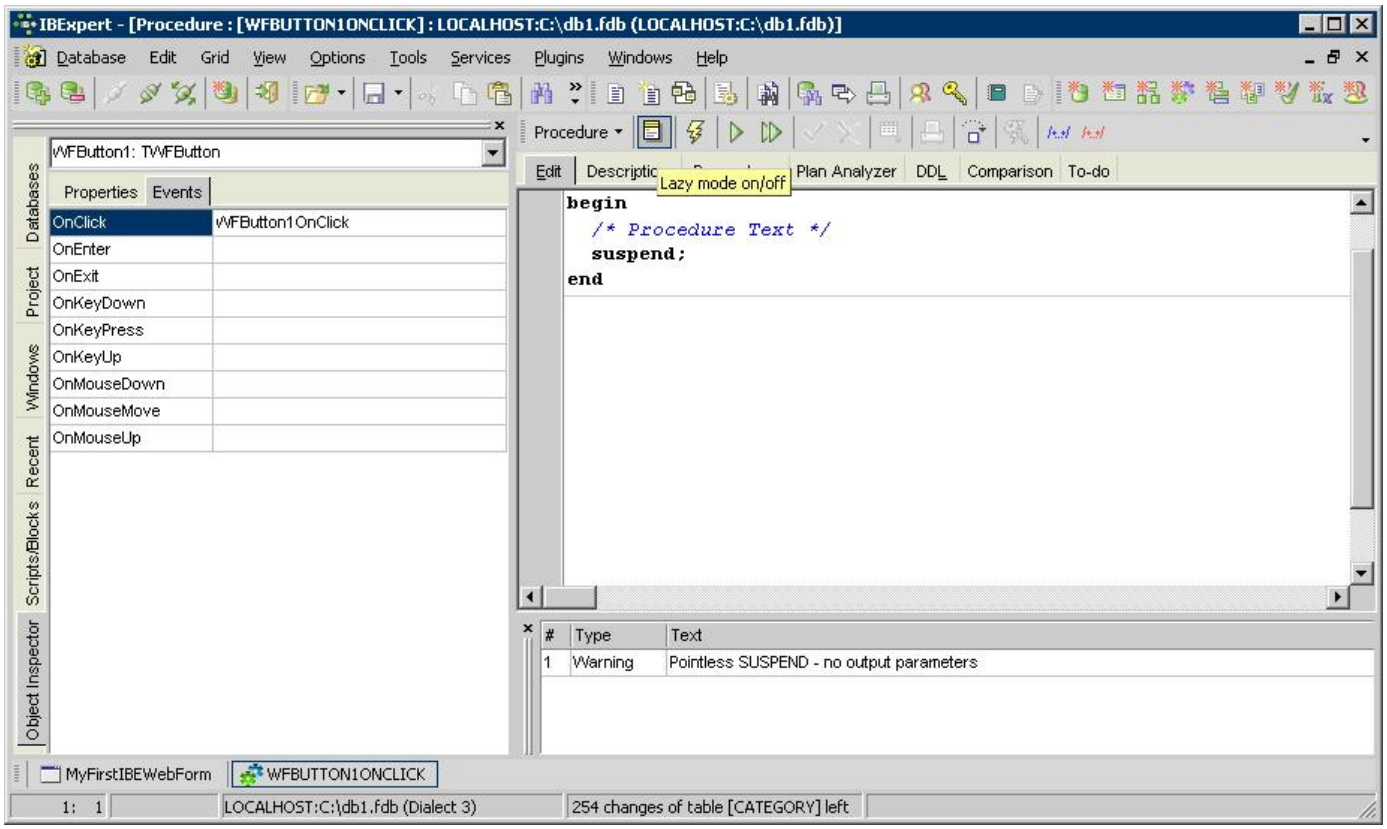
On the left-hand side of the *Object Inspector*, you can create and modify all properties and events of a selected component. For example, the text of a *TWFLabel* component can be modified using the property *Caption*, or the font modified using the *TFont Properties Editor* (select the property *Font* and then click on …). The *bject Inspector* can be used to modify a whole range of properties and events.

How do I create an event?

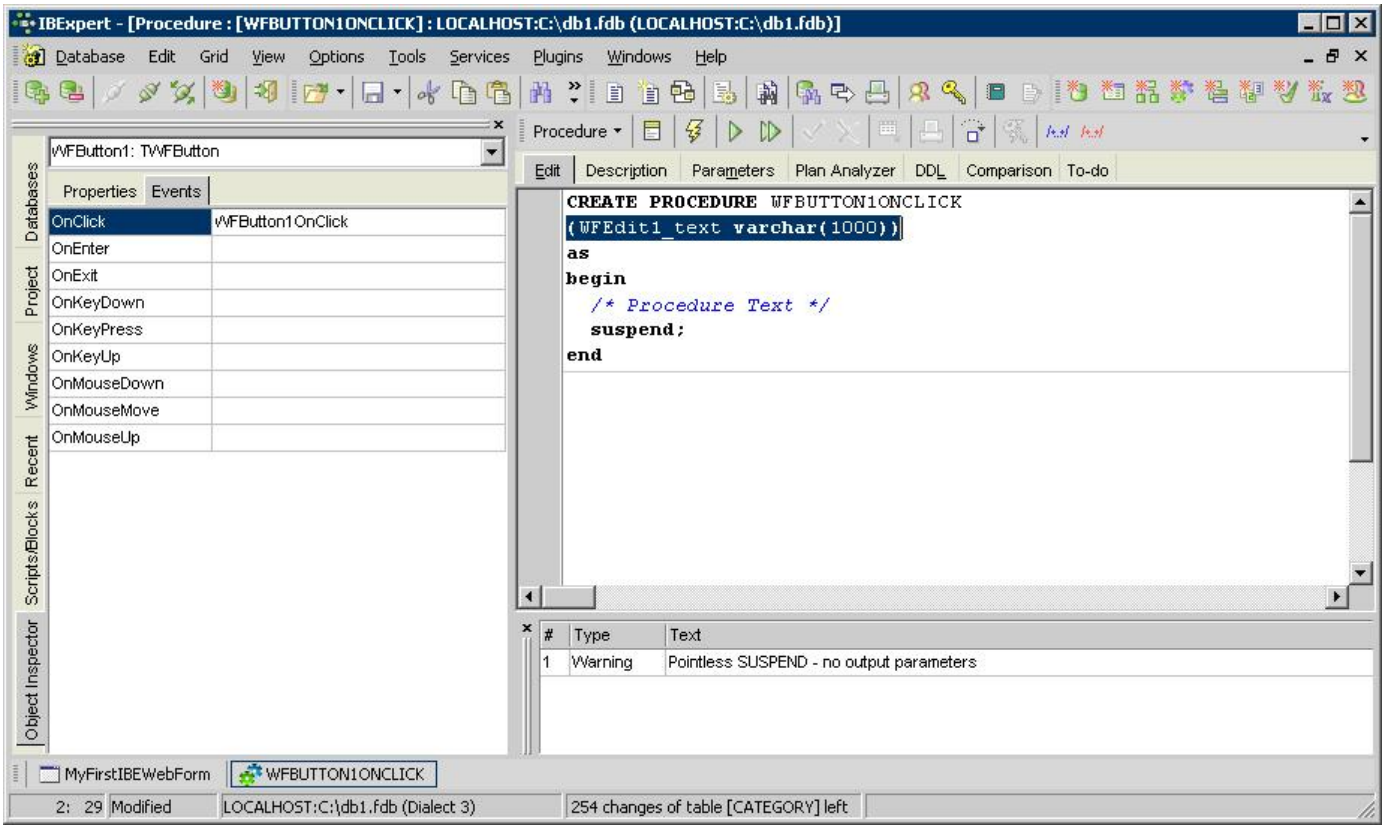
Events are incidents that occur during runtime which are, for example, triggered by clicking on a button. A stored procedure in the database can be assigned to each event. For example: in order to create an *OnClick* event for a *TWFButton*, go to the *Events* page in the *Object Inspector*, and simply double-click *OnClick*. The Stored Procedure Editor is opened, and you can specify the event using PSQL.



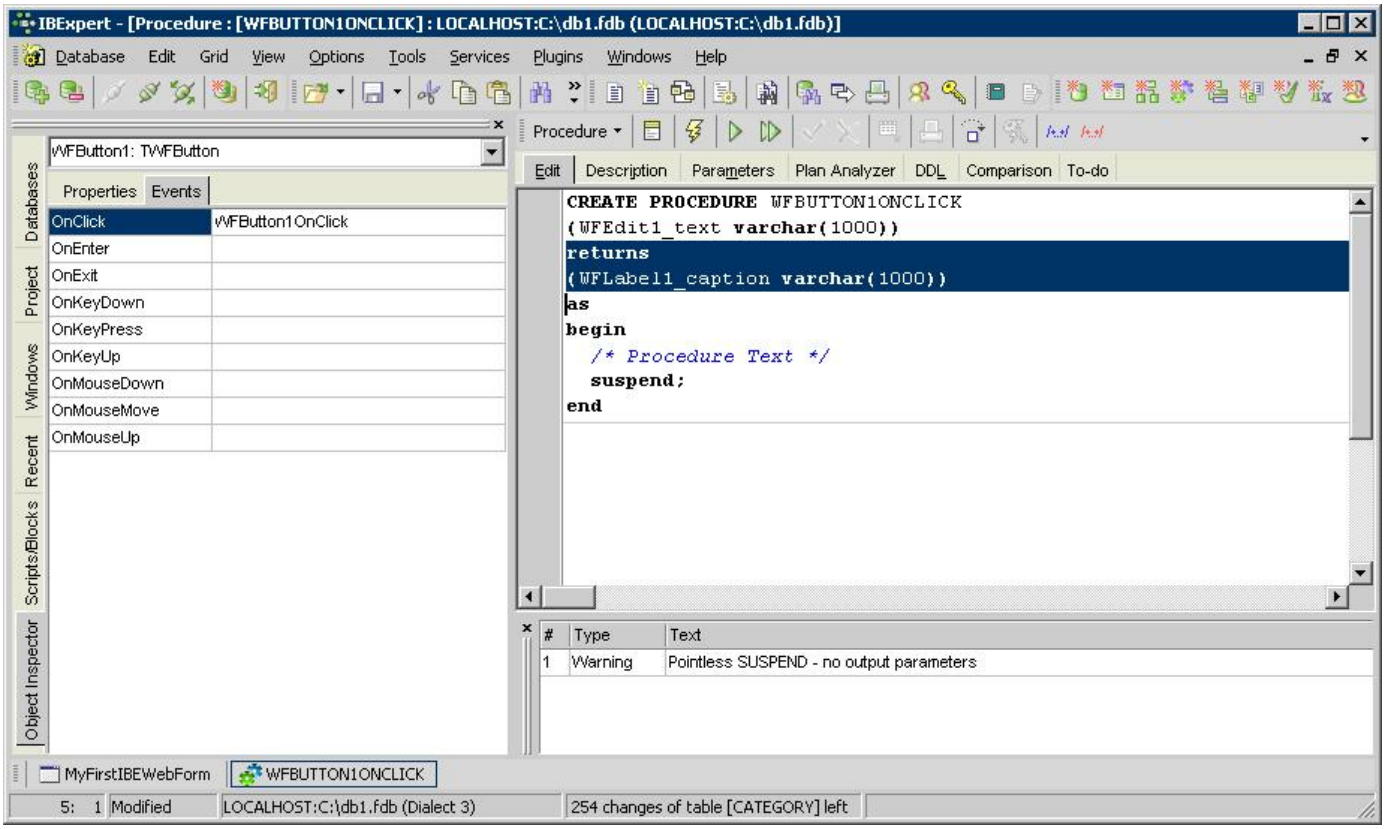
Double-click *OnClick* in the *Object Inspector* (*Events* page). The Stored Procedure Editor is opened, and you can formulate the event using PSQL. If necessary, deactivate the Lazy Mode in order to view the complete stored procedure.



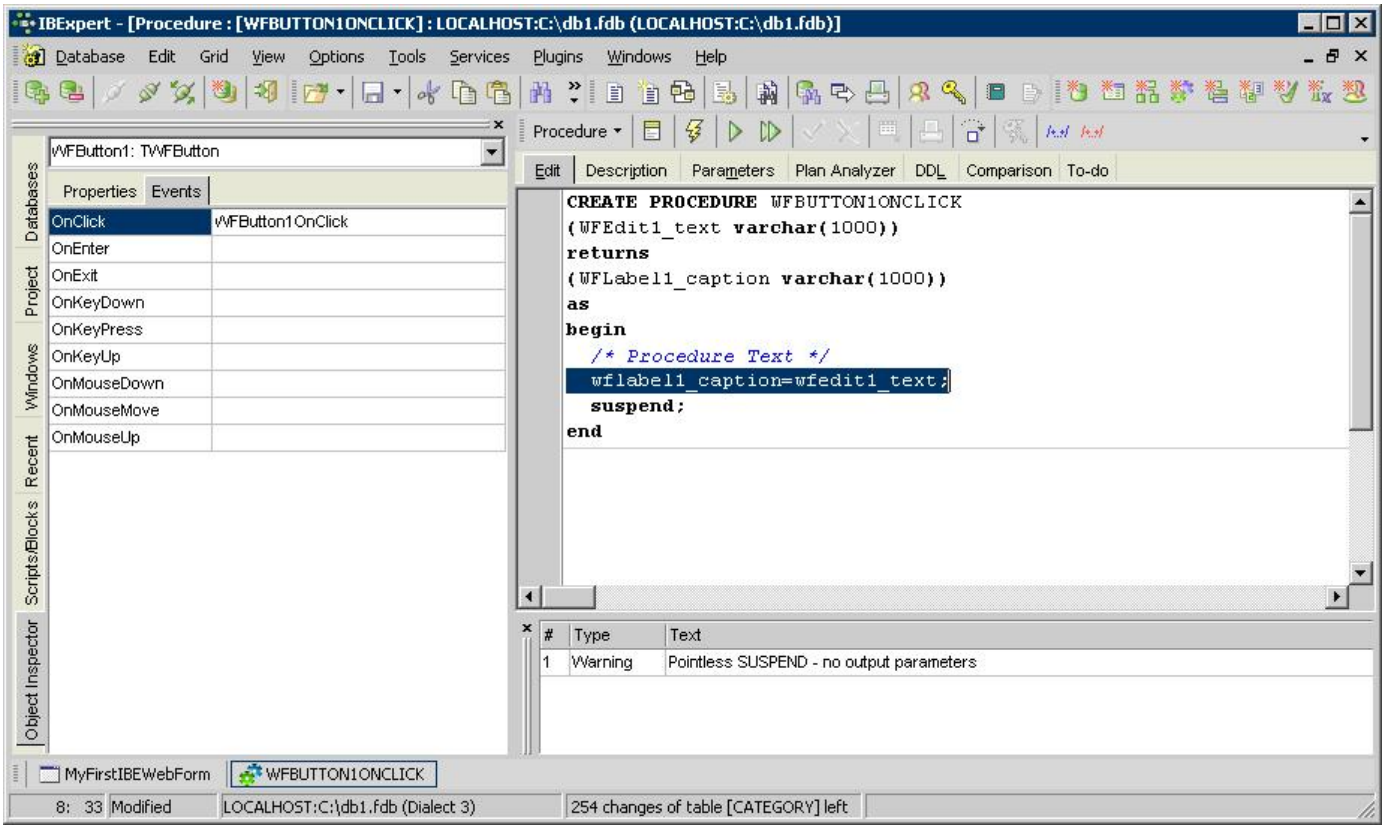
As an example we will now display the contents of the input field, when the button is clicked on. For this we first need an input parameter for the stored procedure, which we shall phrase as follows:



Now we need a return parameter for the text element, which can be defined as follows:



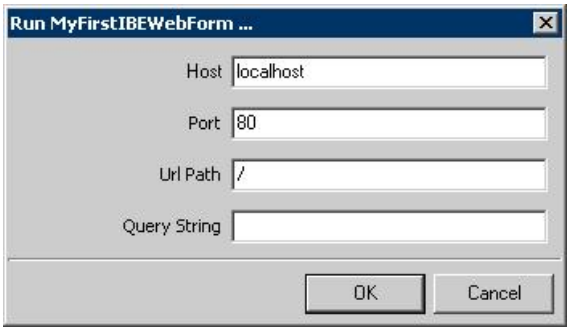
We now need to make a statement in the stored procedure body, so that the contents of the input element can be allocated to the text element.



Finally the stored procedure needs to be compiled, by clicking on the following icon:

For the first test, close the procedure Editor after compiling and save the WebForm.

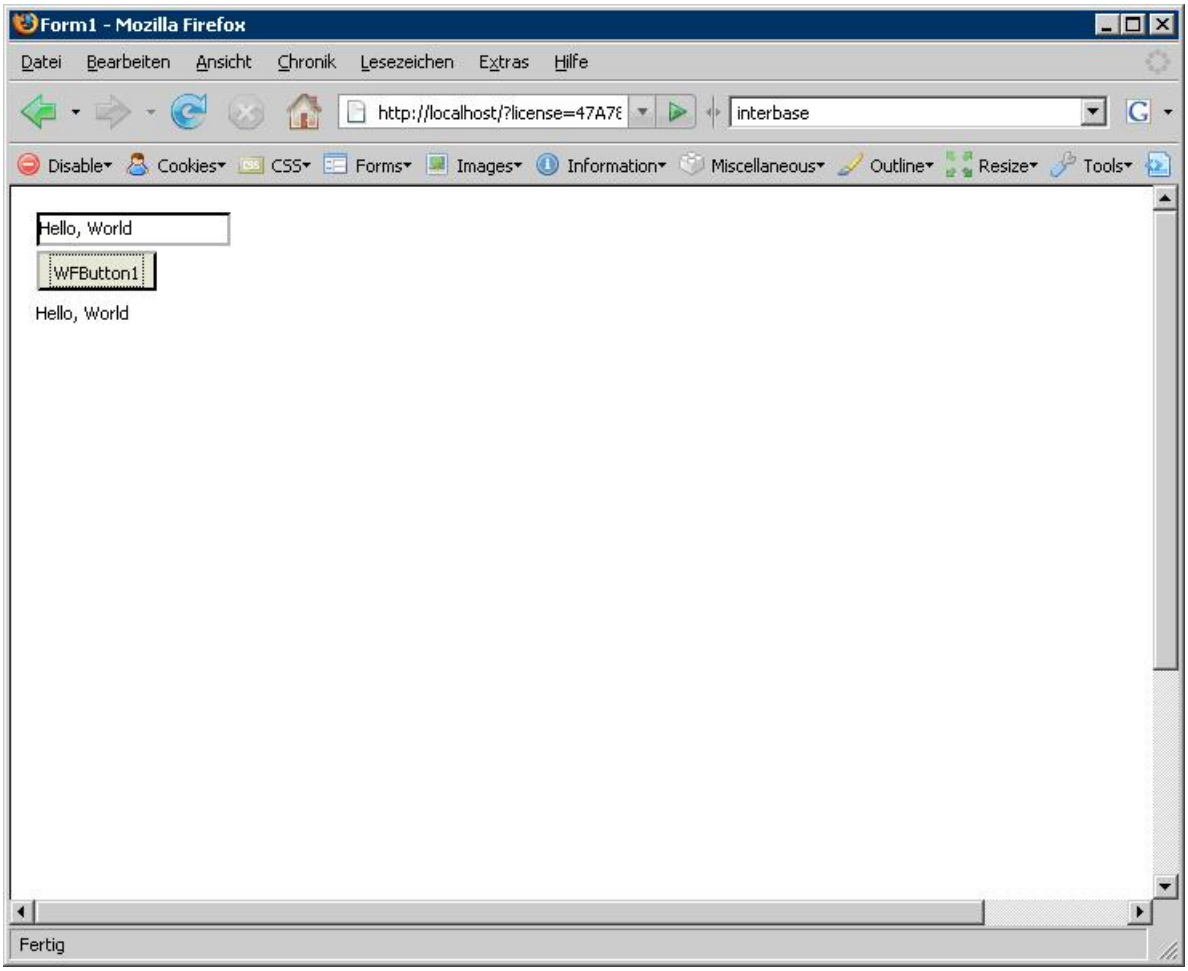
Now you can test the form in the browser by pressing [Ctrl+F9]. This will display the *Config* dialog:



The default *Port value* is `3080`. If you want to use this application with a typical `http` port, just change it to the standard port `80` before clicking *OK*. This will change the configuration of your integrated Apache server to use this port. *Please note:* If this port is already in use by another application, change it to a free port, for example *3080*.

After changing the Apache configuration, IBExpert will automatically start your Web browser and show you this application.

After changing the text, just press the button *WFButton1* and it will be shown in the *WFLabel1*.

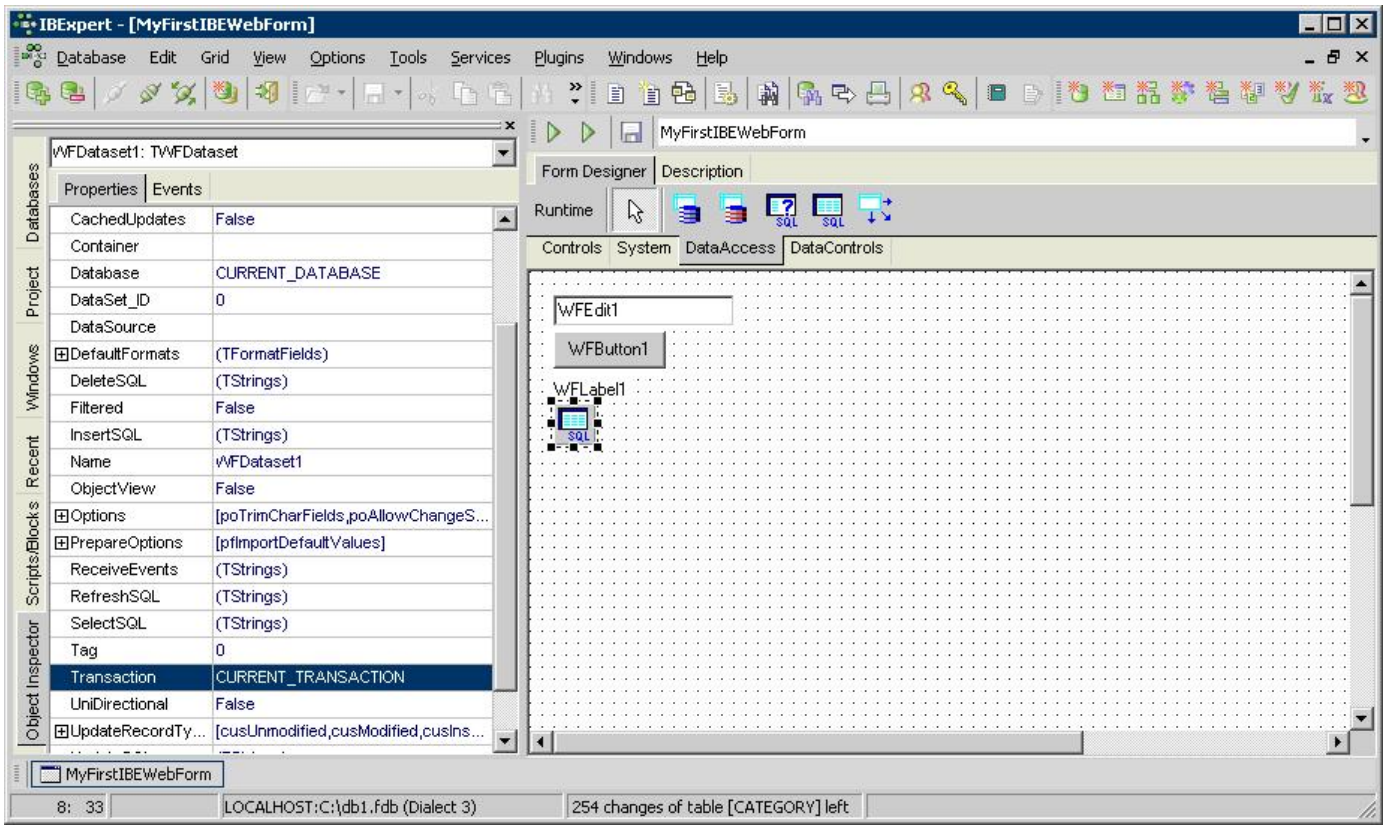

Form1 - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://localhost/?license=47A7E
interbase
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools
Hello, World
WFButton1
Hello, World
Fertig

How do I handle the database components?

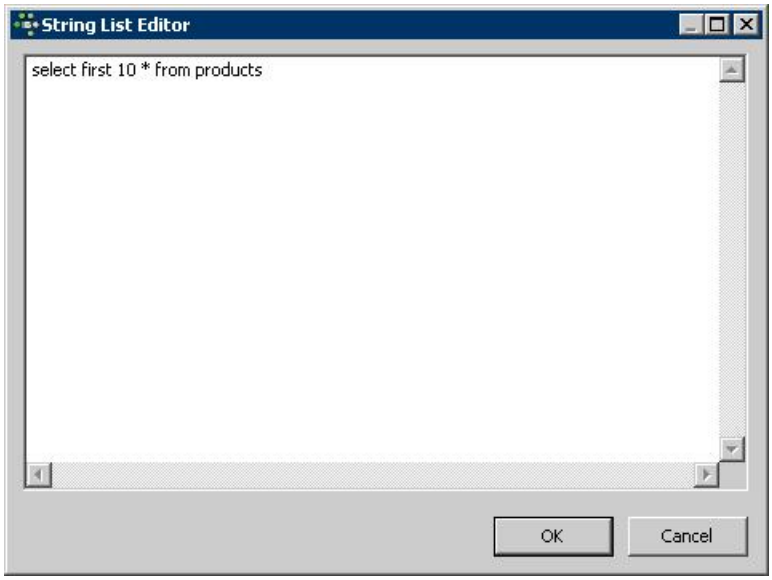
In IBExpertWebForms there is a component bar, *DataAccess*, with a range of components with which you can create a database connection and start database queries. These components are not visual, i.e. they cannot be seen in the web browser later. Another component bar, *DataControls*, can be subsequently used to edit data in the web browser if wished.

If you just want to work with table data in the current database, you do not have to create the `TWFDatabase` and the `TWFTransaction` component, since an instance which will be used in our example called `CURRENT_DATABASE` and `CURRENT_TRANSACTION` is automatically created.

To send a database query to the database or to specify a `SELECT` SQL for the display of data, create a *TWFDataset* component and connect this with the *Database* and *Transaction* properties to the available instances.

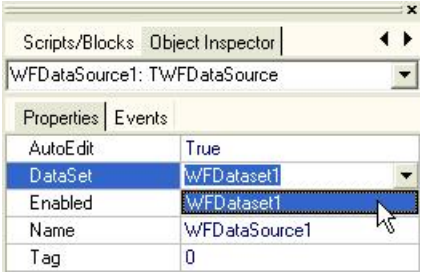


To put a `SELECT` statement to the database, use the *SelectSQL* property. This opens the *Property Editor*, where you specify any `SELECT` statement. You can also use the IBExpert Query Builder (IBExpert Tools menu), to create `SELECT` SQLs. We will now use the following `SELECT` statement `select first 10 * from products` and confirm with *OK*.



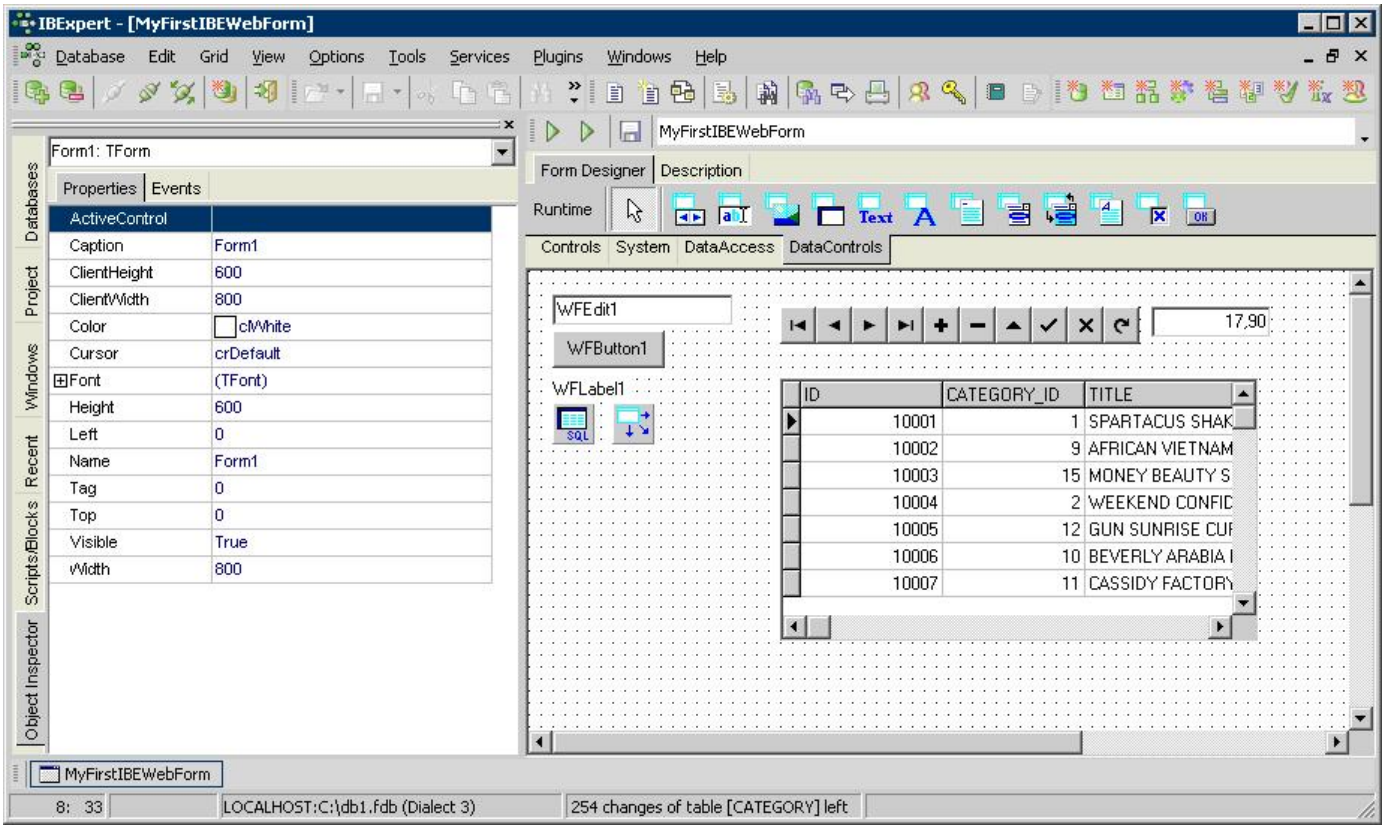
To test the query, double-click on the *Active* property. If the status changes from *Active* to *True*, the query is error-free and the properties *Database* and *Transaction* have been set correctly; otherwise you will receive a corresponding error message.

Now we need a *TWFDataSource* component, to obtain a data source for our visual database component in the component bar *DataControls*. Set the property *DataSet* to *WFDataset1*.



After changing the *dataset* property, you should click on the *Name* property, so that the properties are stored. Finally we can place components from the *DataControls* onto the form.

In the following example a *TWFDBNavigator*, *TWFDBEdit* and a *TWFDBGrid* have been created:



The following lists the properties and their values, as defined to achieve the above result:

DBNavigator1

```
DataSource=WFDataSource1
```

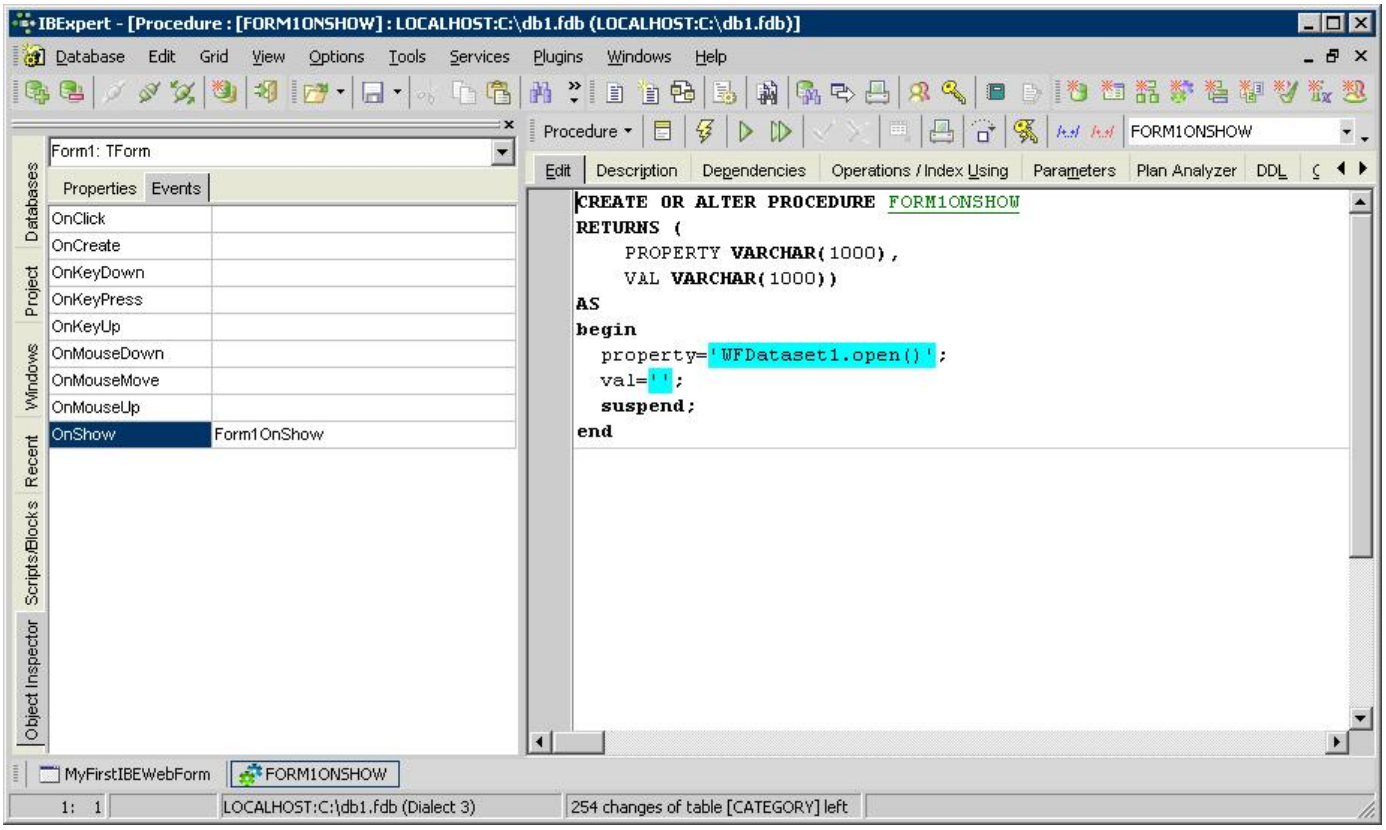
DBEdit1

```
DataField=PRICE
DataSource=WFDataSource1
```

DBGrid1

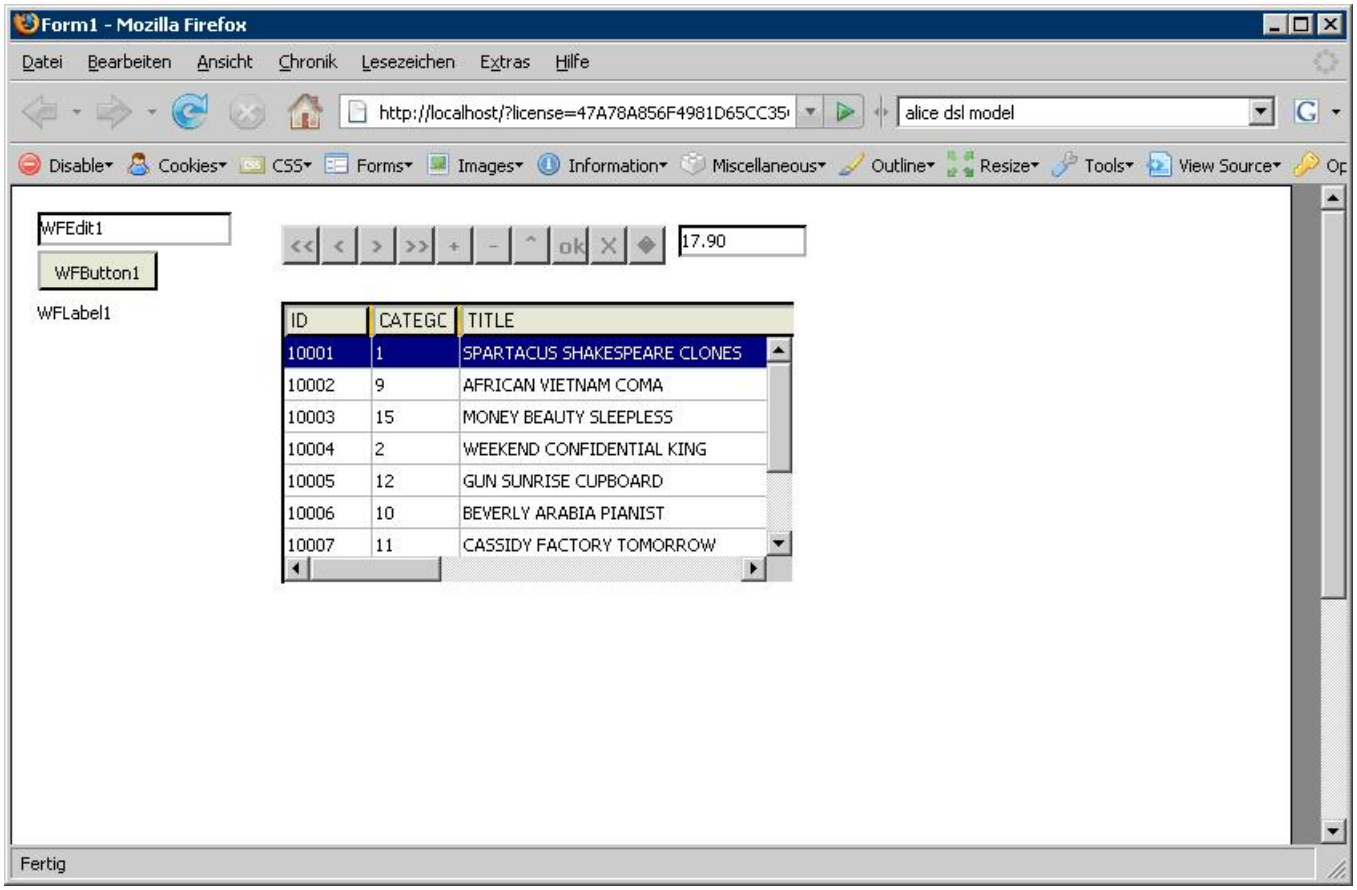
```
DataSource=WFDataSource1
```

We still need a stored procedure to ensure that *WFDataSet1* is opened, the moment the form is displayed in the web browser. For this we will select the *Form* `form1` and create an *OnShowEvent* procedure. Simply double click on the *OnShowEvent* and add the procedure source code as shown. This procedure only has return parameters and no input parameters.



The return parameter property may contain any of the supported properties and methods of the available components. In the example the open method of the *WFDataset1* component is invoked. The return parameter, `val`, allows values for properties to be deposited. However in our example, `val` was not used, as we want to invoke a method.

Now start the WebForm again with [F9]. This button hides the *Config* dialog and starts the new WebForm in your browser.



You would like more examples?

Download the Pizzashop Demo from http://www.ibexpert.com/download/IBExpertWebForms/pizza.zip.

After downloading `pizza.zip`, unpack it and do a restore with Firebird 2.0. We recommend storing the database on your local machine in a directory, for example, `c:\pizza\pizza.fdb` (this path is hard coded in the database component, should you wish to change it).

Register the database in IBExpert and open the *Pizza Shop* form for a simple demo of the pizza web shop example or the *hkx* form for a more complex example. We will add documentation in the near future.

If you want to analyze the forms, just take a look at the table `ibe$scripts`.

IBExpertBackupRestore

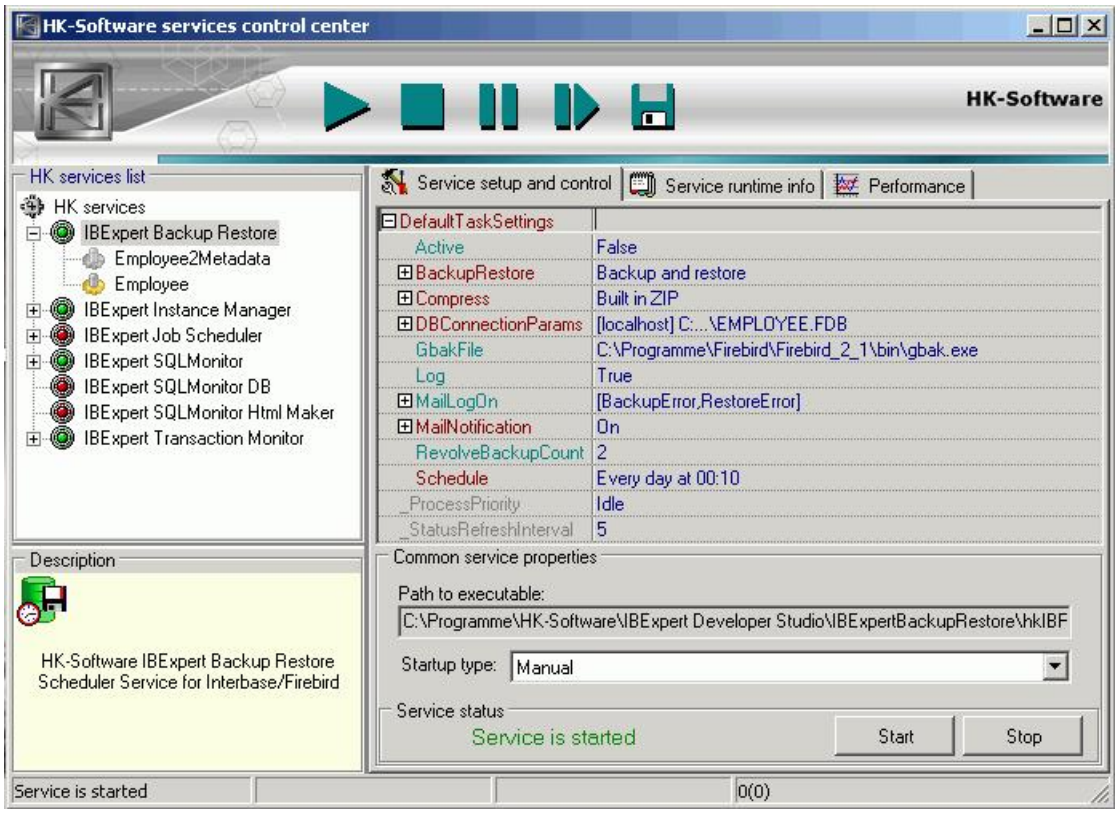
What is IBExpertBackupRestore?

The IBExpertBackupRestore scheduler service is an comprehensive utility, providing automatic backup and restore facilities for Firebird and InterBase databases with backup file compression even an option to automatically mail backup/restore log files.

This service is part of IBExpert KG IBExpert Developer Studio for Firebird and InterBase database development and administration.

Service Description

Using IBExpertBackupRestore it is possible to set up automatic backups for any number of databases, with separate backup, restore, schedule and log mailing parameters for each database. The service is controlled by the HK-Software Services Control Center (SCC) utility, which can be found in the IBExpert Services menu.



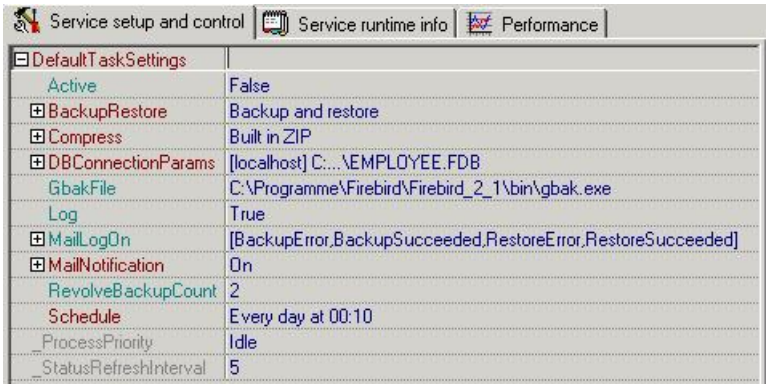
Here you can see the screenshot of the HK-Software SCC with the IBExpertBackup/RestoreScheduler configuration loaded. In the HK Services list tree view you can actually see the service item with two tasks below it. Each task is a database backup/restore schedule configuration.

Setup and usage

Start the HK-Software Services Control Center, found in the IBExpert Services menu, and select *IBExpert Backup Restore* in the *HK services* list.

We now need to configure the default task settings. We know that some parameters will remain the same for all further tasks (for example: path to `gbak.exe`, SMTP settings, etc.), so we should configure those first.

Expand the *DefaultTaskSettings* item on the *Service setup and control* page.



The following lists the various default settings and options available:

- Active
- Backup and Restore
- Compress
- Database connection configuration
- Path to gbak.exe
- Logging
- Mail Notification
- Revolve Backup Count
- Schedule

After configuring the default task settings, all new tasks will have this configuration when created. It is of course possible to alter specific options for individual tasks.

Default task settings

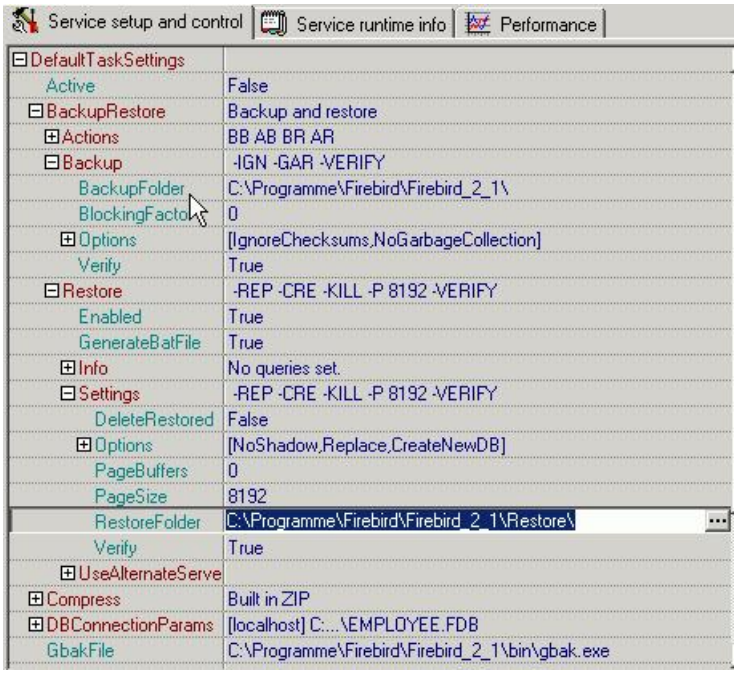
Active

When `True` then the task just created will be active.

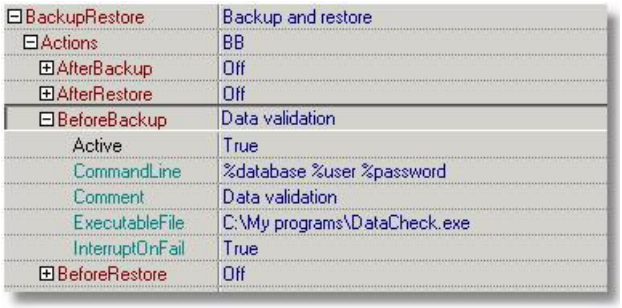
Backup and Restore

This contains the basic backup and restore settings, processed by `gbak.exe`. Also there are few settings specific to the HK service, such as:

- **BackupFolder:** the folder where all backups will be stored
- **Restore/Enabled:** when `True`, then service will restore a database from a successful backup file. This can be used to validate the backup file.
- **RestoreFolder:** the folder to restore the database to, from the backup file just made.



If you need to perform any additional operations before/after the backup/restore (for example script execution, data validation, etc.) you may use the *Actions* options in the IBExpertBackupRestore scheduler service. The screenshot below shows the corresponding section with the *BeforeBackup* action expanded in SCC on the *Service setup and control* page.



Imagine that you've configured this task to backup a database `my_server:c:\my_database.gdb` and username and password are `SYSDBA/masterkey`. The *BeforeBackup* configuration example above means that before starting the database backup, the service will execute the command line:

```
C:\My programs\DataCheck.exe my_server:c:\my_database.gdb SYSDBA masterkey
```

If you need to interrupt the backup/restore process because some data validation or other operation has failed, you can use the *InterruptOnFail* option of the corresponding action. The execution of any action will be recognized as failed if the executed program sets the exit code not equal to 0 (zero).

The command line for each action may be configured using executable file parameters as well as with scheduler service macros. The macros will be replaced with corresponding values.

Here a description of the macros:

Macro	Value
`%database`	Full connection string to source database.
`%server`	Database server name.
`%database_file`	Database file path.
`%restored_database`	Full connection string to restored database.
`%backup_file`	Path to backup file.
`%role`	SQL role from *DBConnectionParams*.
`%user`	Username from *DBConnectionParams*.
`%password`	Password from *DBConnectionParams*.

To test the functionality of *Actions* you may use the special executable, `DumpAction.exe`, which only writes its command line to a log file (`DumpAction.exe.log`) and sets the exit code necessary. The exit code for this executable should be configured using a template such as:

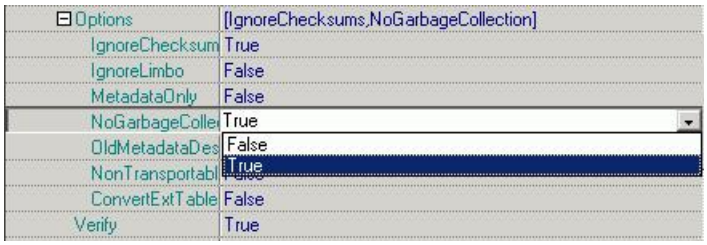
```
DumpAction.exe -RESULT <integer_value>
```

For example, such a configuration of a *BeforeRestore* action will always stop the scheduler performing the restore, because the exit code of such an action will be `2`.

All actions with the corresponding results will be listed in the service report e-mail message as in the example shown below:

```
Started 10.08.2006 at 17:52:25

Database localhost:C:\Firebird\examples\employee.fdb
   - before backup (BB test). (result = 0)
   - backup to E:\HK\IBRS\bin\My Backups\Employee\gbk_2006
   - after backup (AB test). (result = 0)
   - before restore (BR test). (result = 0)
   - restore to "localhost:E:\HK\IBRS\bin\My Restores\empl
   - after restore (AR test). (result = 0)
   - built-in ZIP compress succeeded.
---------------------
```

In the *Backup / Options* section you can configure the backup options as required by simply setting the corresponding items to *True*. The *Verify* options were introduced in IBExpert version 2008.08.08.



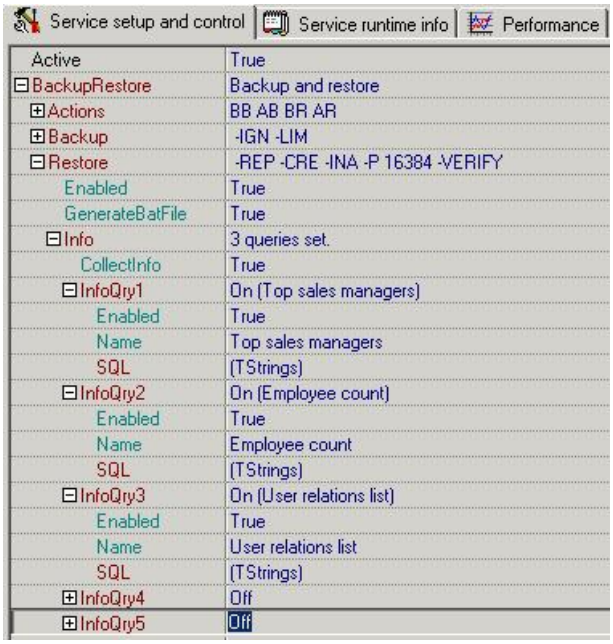
After that you will see the selected items in square brackets [] under *Backup / Options*,

and the corresponding `gbak` command line parameters under *Backup*.

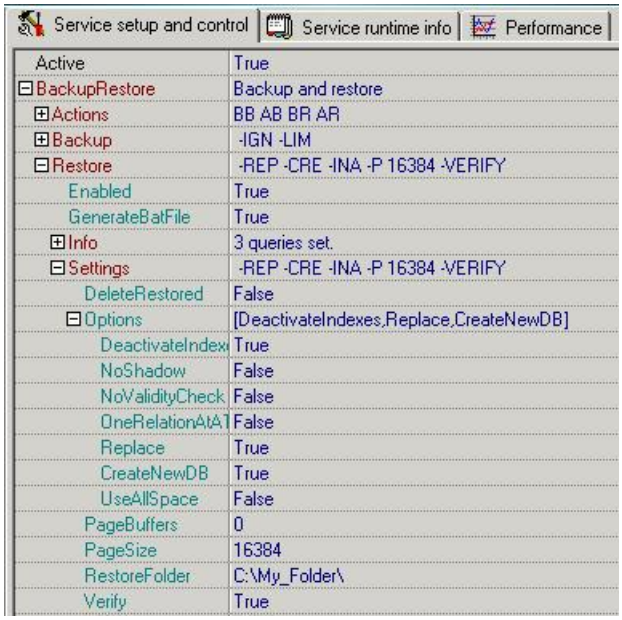
In the screenshots shown above you can see the backup configuration specified with the *No garbage collection* and *Ignore checksum* options.

When *Restore / Enabled* is set to *True*, the IBExpertBackupRestore restore scheduler will perform a restore from the backup just made. This feature can be useful if you want to validate the backup file or wish to use the freshly restored database for better performance.

The restored database information collection functionalitiy was introduced in IBExpert version 2008.08.08. *Restore / Info* can be used to execute up to 5 different queries, enabling you to obtain useful information about the status of the database, for example, the record count of a particular table, the last logged update timestamp or some special report. When *CollectInfo* is set to *True*, the restored database's main parameters, such as fle size, page size, pages count etc., can be viewed.



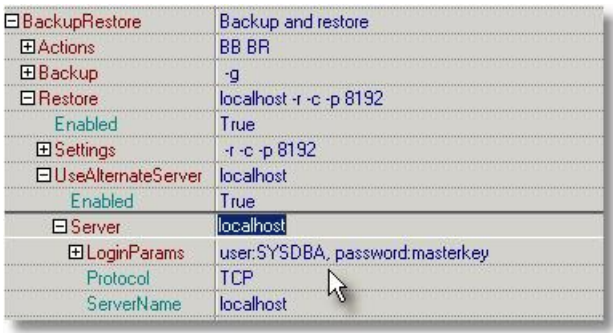
In the *Restore / Settings* section you can set up the desired restore parameters, such as *restore folder*, *restore options*, *database page size*, etc. For example, if you want to restore a database from fresh backup into `c:\My_Folder`, create a database file, if no such file yet exists in the restore folder, or replace it if the file aleady exists. If you wish you may also deactivate indices (*DeactivateIndexes*) to improve the performance of the restore. And perhaps you wish to re-specify the page size (*PageSize*) of the restored database to 16384. The screenshot below displays the corresponding *Restore/Settings* configuration:



If you want make a restore just to validate a fresh backup file, you probably don't need to store the restored database file. So it is even possible to configure the IBExpertBackupRestore Scheduler to delete the restored database file following the restore. Just set the corresponding option to *True*.

Restore to an alternatative server

Backup and restore is very resource-consuming operation. To help your main database server breathe more easily, you can set the scheduler service to perform restores on an alternative server. This can be done using the *UseAlternateServer* option found in the *Restore* parameters.

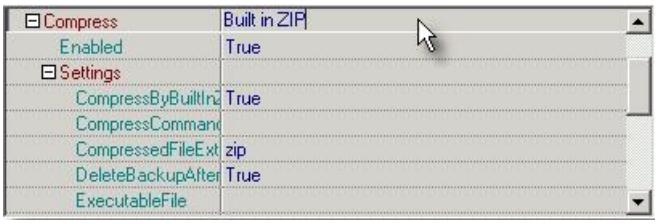


When this option is enabled you can backup your database from one server and restore it to another.

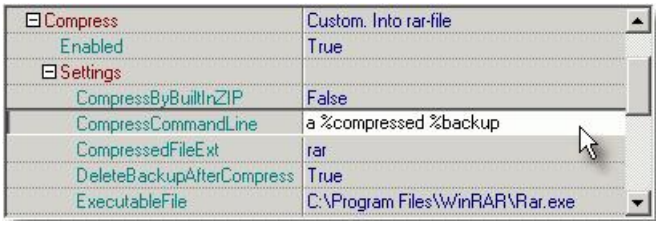
Compress

If you want to compress a successfully created backup file, you should use this configuration section. You can also configure the service here to delete the backup file, following the successful compression (*DeleteBackupAfterCompress* option).

To make the backup compression work you should set *Enabled* to *True*, and then configure the appropriate compress settings. You can use the built-in ZIP compressor or configure the service to run an external compressor exe file. Here is a screenshot of the compress settings configured to use the built-in ZIP compressor:



Here is a screenshot of a configuration using an external compressor (for example WinRAR):



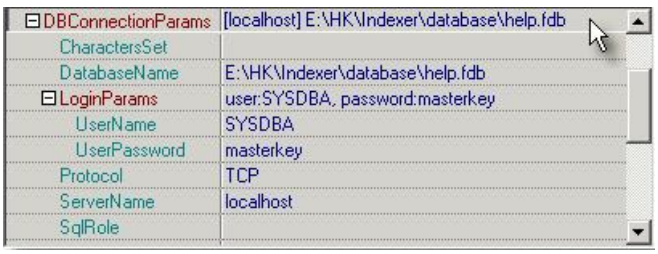
The *CompressCommandLine* option can contain three macros, which will be replaced with the corresponding values when calling the compressor:

`%backup`	Backup file name with extension.
`%compressed`	Compressed file name = backup file name + extension.
`%back_filename`	Backup filename without extension.

The extension is configured in *CompressedFileExt*.

Database connection configuration

The essential key to any database manipulation (except moving it into the recycler!) is establishing the database connection. All necessary properties can be configured in the *DBConnectionParams* section:



This is fairly self-explanatory; although should you require detailed information regarding Firebird/InterBase database connection parameters, please refer to the online IBExpert documentation.

Path to gbak.exe

The IBExpertBackupRestore Scheduler collaborates with `gbak.exe` to enhance the backup/restore tasks. So you need to let the service know where this file can be found:

Logging

It's likely you'd like to have log files of your backup/restore operations. Those files may help you to understand what's wrong with your database, should an error occur during the backup/restore process. To enable such log files, just set the corresponding option to *True*, as shown below:

Mail notification

You may use the mail notification feature if you want to receive reports about the IBExpertBackupRestore Scheduler activity. The service sends an e-mail message with log files attached when the backup/restore task is completed.

The *MailLogOn* option is used to define the situations, when log files should be mailed. For example, if you'd like to receive log files when a backup or restore has failed, you should specify the options as follows:

To use the mail notification feature, the *Enabled* parameter in the *MailNotification* section should be set to *True*.

The IBExpertBackupRestore Scheduler uses a built-in SMTP client to send e-mails, so you need to set up the SMTP parameters in the task configuration to enable this to work properly. Simply double-click on the *SmtpSettings* option, to open the configuration dialog window.

In this dialog you should set up the *Sender*, *SMTP server configuration* and one or more recipients.

Revolve backup count

The IBExpertBackup/RestoreScheduler works as a rotator when creating a new backup. If a new backup is successfully created, the oldest one will be deleted. Such mechanics let you configure the service to store just *n* last backups. The *n* value can be configured in this option:

Schedule

So far you may still be confused as to why we have decided to call this tool "a scheduler". Well, it's quite simply because that's just what it is!

Double-click on the *Schedule* option to open the schedule configuration dialog window:

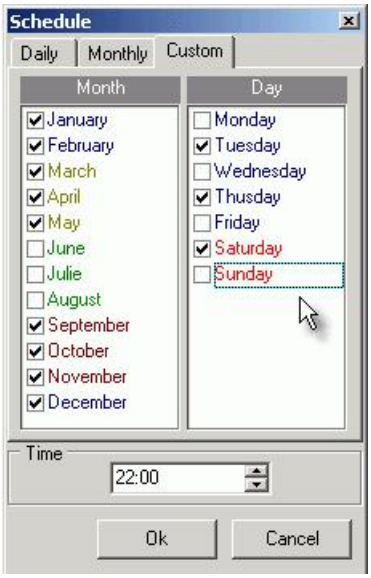
Daily schedule:

- every day at the specified time.
- every *n*th day, starting from date.
- every given day of week.

Monthly schedule:

Every *n*th day of the selected months at the given time.

Custom schedule:



Selected days of every week of selected months at given time.

_ProcessPriority

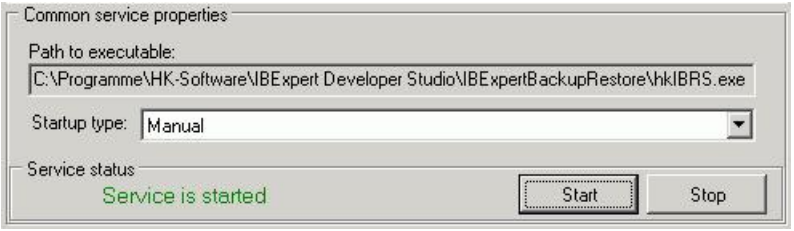
This parameter can be set to *Idle*, *Normal* or *High* (the default is *Idle*).

_StatusRefreshInterval

Here the refresh interval in seconds can be specified (default value is 5).

Common service properties

The path to the executable file, `hkIBRS.exe` is displayed. You can specify the *Startup type* selecting an option from the drop-down list (options: *Manual*, *Automatic* or *Disabled*).



The *Service Status* can be viewed at the bottom of the window, and the *Start* and *Stop* buttons used to manually start or stop the service.

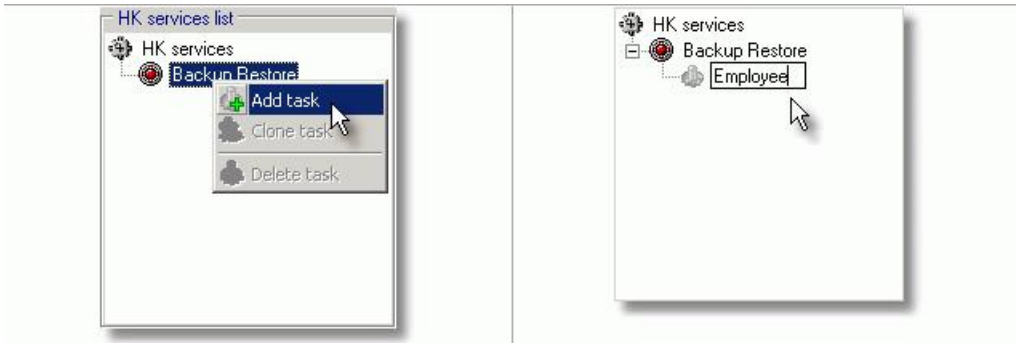
Once you are sure you've configured your default settings as you need them, don't forget to save your configuration by clicking the disk icon in the toolbar, before moving on to configuring your individual databases for their backup.

Configuring the database for a backup

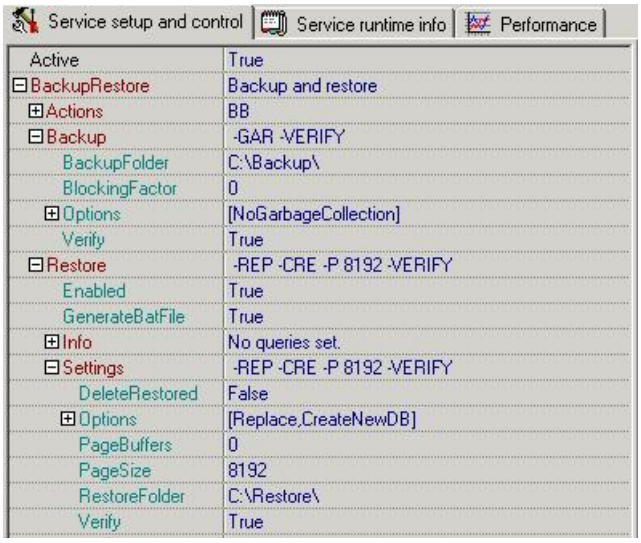
After configuring the default task settings, all new tasks will have the same configuration when created. You can of course alter specific options in the individual tasks if wished.

Let's configure the IBExpertBackupRestore Scheduler to backup our database:

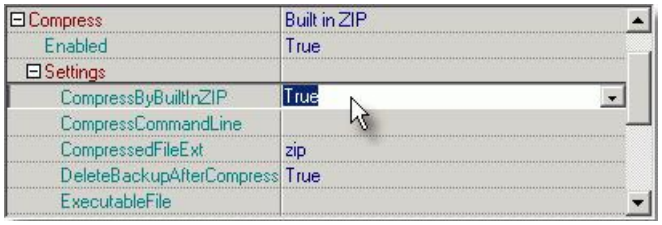
1. Right-click on the *IBExpert Backup Restore* service's item in the SCC. Then click *Add task* in the popup menu. After that you will see the new task item (Task 0) under the *Backup Restore* service's item. You may rename it by clicking on the name simultaneously holding the [Ctrl] key down. In the example below you can see a new task, renamed to *Employee*.



2. Configure the *Actions* (if any) and the *Backup* and *Restore* settings (backup folder, restore folder, page size, backup options and restore options) as shown in the screenshot:



3. Setup the GBK file compress with the built-in ZIP compressor, as shown on the screenshot below:



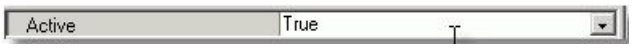
Of course you can also specify an external compressor application of your choice. (Further information can be found in the previous chapter.)

4. Setup the database connection parameters and path to `gbak.exe`:

5. Set up the *MailNotification* and *SMTP* settings as required:

6. Set up the schedule for your local time + 5 minutes so that we can see this task running.

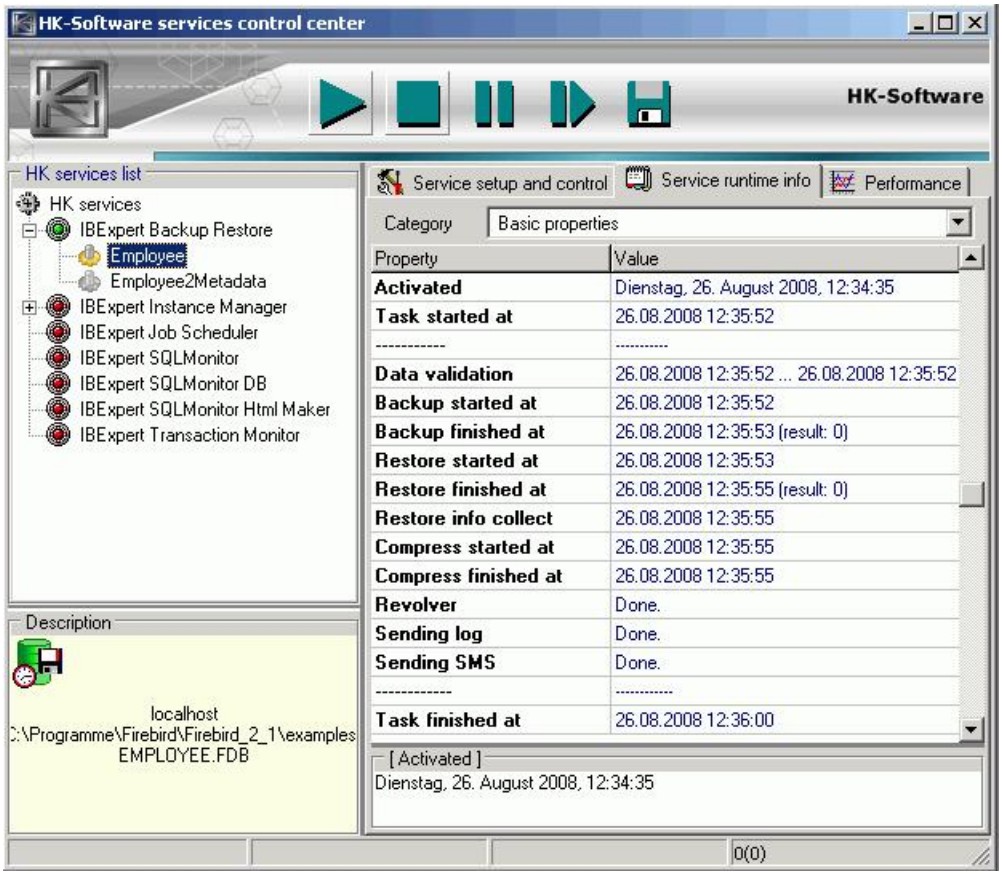
7. Set the *Active* property to *True*.



8. Save the service's configuration by pressing the *Save* button in the SCC:

9. Run the IBExpertBackup/RestoreScheduler service by pressing the *Run* button in the SCC:

10. Now select the task in the HK services list, then switch to the *Service runtime info* page to see the task-related service activity:



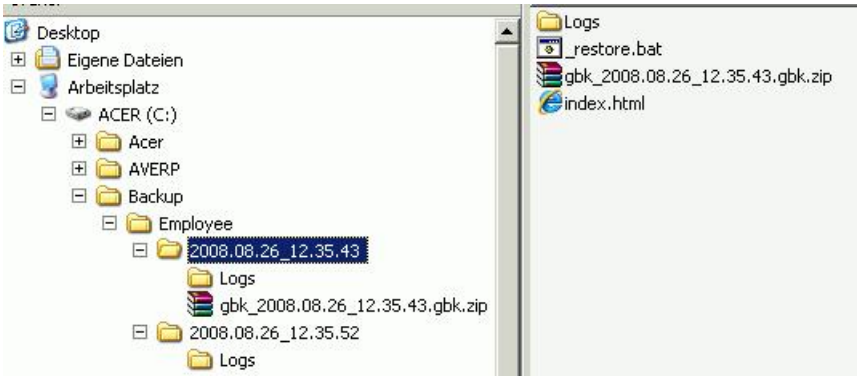
Also, if you check your mail for the address configured in the SMTP settings, there should be a report message from the backup/restore service, provided of course that you have specified mail notification of both a successful and unsuccessful backup:

Betreff: HK-Backup/Restore service report **Von:** Your database 12:35

```
Task Employee
Backup cl:
"C:\Programme\Firebird\Firebird_2_1\bin\gbak.exe" -B  -user "SYSDBA" -pas "masterkey" -y
"C:\Backup\Employee\2008.08.26_12.35.52\Logs\blog_2008.08.26_12.35.52.log"  -GAR -VERIFY
"localhost:C:\Programme\Firebird\Firebird_2_1\examples\EMPLOYEE.FDB"
"C:\Backup\Employee\2008.08.26_12.35.52\gbk_2008.08.26_12.35.52.gbk"

Restore cl:
"C:\Programme\Firebird\Firebird_2_1\bin\gbak.exe" -REP -CRE -P 8192 -VERIFY -user "SYSDBA" -pas "masterkey"
-y "C:\Backup\Employee\2008.08.26_12.35.52\Logs\rlog_2008.08.26_12.35.52.log"
"C:\Backup\Employee\2008.08.26_12.35.52\gbk_2008.08.26_12.35.52.gbk" "localhost:C:\Restore\EMPLOYEE.FDB"

Started 26.08.2008 at 12:35:52

Database localhost:C:\Programme\Firebird\Firebird_2_1\examples\EMPLOYEE.FDB
   - before backup (Data validation). (result = 0)
   - backup to C:\Backup\Employee\2008.08.26_12.35.52\gbk_2008.08.26_12.35.52.gbk succeded. (result = 0)
   - restore to "localhost:C:\Restore\EMPLOYEE.FDB" succeded. (result = 0)
   - built-in ZIP compress succeeded.
---------------------
Compressed file name(s):
---------------------
        GBK file deleted after compress.
```

blog_2008.08.26_12.35.52.log rlog_2008.08.26_12.35.52.log

In the backup folder you can find fresh backup and backup/restore log files.



And in the restore folder the restored database.

A new `index.html` is produced, if you have specified information collection in the *Restore* list of parameters. This displays the main database information and, if you have specified queries, the queries list on the right-hand side.

Common database Info	
Server Version	WI-V6.3.0.4201 Firebird 1.5 Release Candidate 8
Ods Version	10.1
PageSize	8192
Pages Allocated	551
DB File Size	4 513.8 KB
Server	DDKING
Database File	E:\HK\IBRS\BIN\MY RESTORES\EMPLOYEE.GDB

Info Queries

Top sales managers
Employee count
User relations list

When the sample *Top sales managers* query is clicked on the report appears below:

Top sales managers Show SQL

NAME_DEPT	TOTAL_SALES	LAST_ORDER_DATE
Claudia Sutherland (Field Office: Canada)	960008	09.08.1993
Michael Yanowski (Sales and Marketing)	502192.23	07.02.1994
Jacques Glon (Field Office: France)	462600.49	18.12.1993
K. J. Weston (Field Office: East Coast)	139450.5	31.12.1993
Roberto Ferrari (Field Office: Italy)	122693	27.10.1993
Luke Leung (Pacific Rim Headquarters)	37475.69	13.02.1994
Takashi Yamamoto (Field Office: Japan)	24190.4	12.12.1993
Pierre Osborne (Field Office: Switzerland)	1980.72	06.01.1994

Clicking the *ShowSQL* button displays the query:

Top sales managers Show SQL

```
select first 10
e.first_name||' '||e.last_name||' ('||d.department||')' as Name_Dept,
sum(s.total_value) as Total_sales,
max(s.order_date) as Last_Order_Date
from employee e
join department d on d.dept_no = e.dept_no
left join sales s on s.sales_rep = e.emp_no
group by 1
having sum(s.total_value)>0
order by 2 desc
```

NAME_DEPT	TOTAL_SALES	LAST_ORDER_DATE
Claudia Sutherland (Field Office: Canada)	960008	09.08.1993
Michael Yanowski (Sales and Marketing)	502192.23	07.02.1994
Jacques Glon (Field Office: France)	462600.49	18.12.1993
K. J. Weston (Field Office: East Coast)	139450.5	31.12.1993
Roberto Ferrari (Field Office: Italy)	122693	27.10.1993
Luke Leung (Pacific Rim Headquarters)	37475.69	13.02.1994
Takashi Yamamoto (Field Office: Japan)	24190.4	12.12.1993
Pierre Osborne (Field Office: Switzerland)	1980.72	06.01.1994

IBExpertInstanceManager

What is the IBExpertInstanceManager?

IBExpertInstanceManager is a new module in the HK-Software Control Center. It allows you to install several instances of the Firebird server on one Windows machine using different ports. Additional functions include monitoring and performance.

Using multiple instances of the Firebird server has numerous advantages, for example, using different SYSDBA passwords, using multiple CPUs more effectively, or using old and new Firebird versions on one machine.

This service is part of IBExpert KG IBExpert Developer Studio for Firebird and InterBase database development and administration.

The IBExpert Junior VAR license or the VAR license entitles you to distribute the IBExpertInstanceManager with your application.

Specify Firebird instances in 12 easy steps

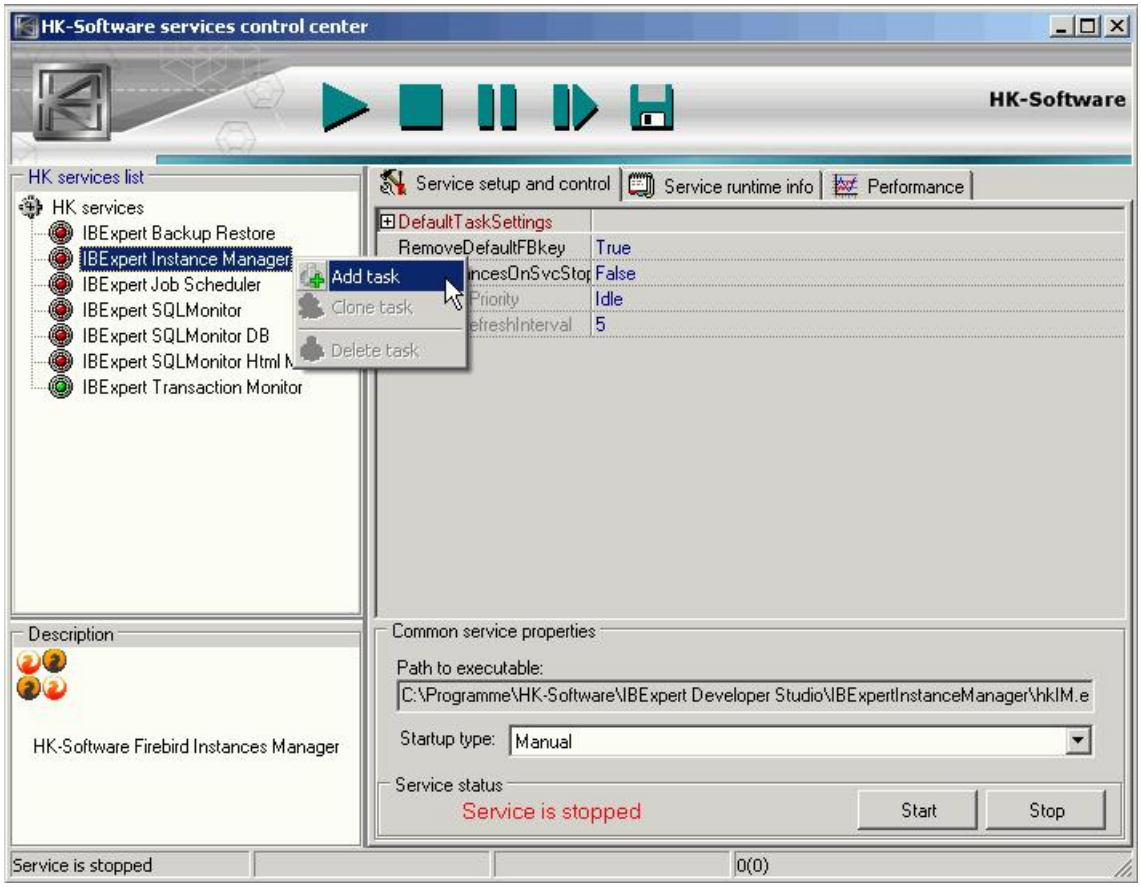
1. Be sure that there is already a Firebird instance installed on your machine using the default Firebird installer. Refer to the Download and install Firebird chapter in the IBExpert documentation for instructions on how to install Firebird.

2. Install the new IBExpert version. The IBExpert documentation chapter, Download and install IBExpert, explains installation of the various IBExpert versions in detail. If you prefer to install the HK-Software Services Control Center manually (without using the setup program) you should take the following steps:

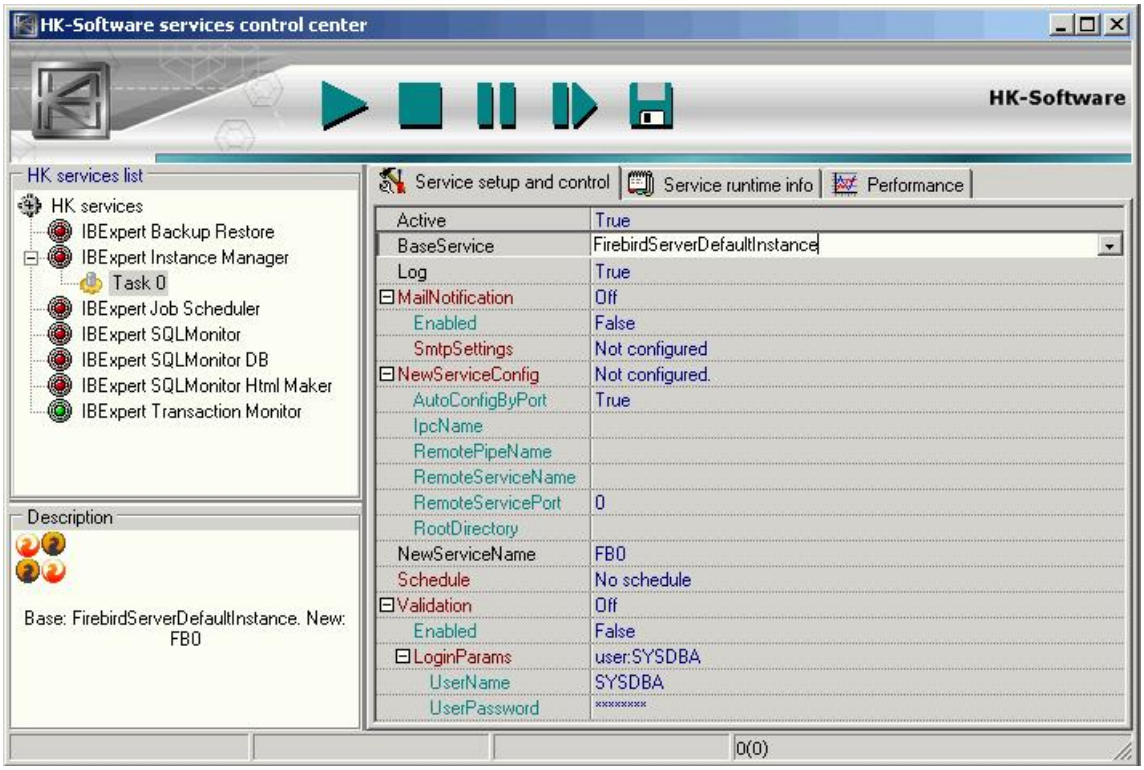
- Execute `hkIM.exe /reinstall`.
- Put `hkIMsvc.hks` in the `svc.data` folder near your `hkSCC.exe`.

3. Start the Services-HK Software Services Control Center. In IBExpert you can find this in the IBExpert Services menu item, HK-Software Services Control Center.

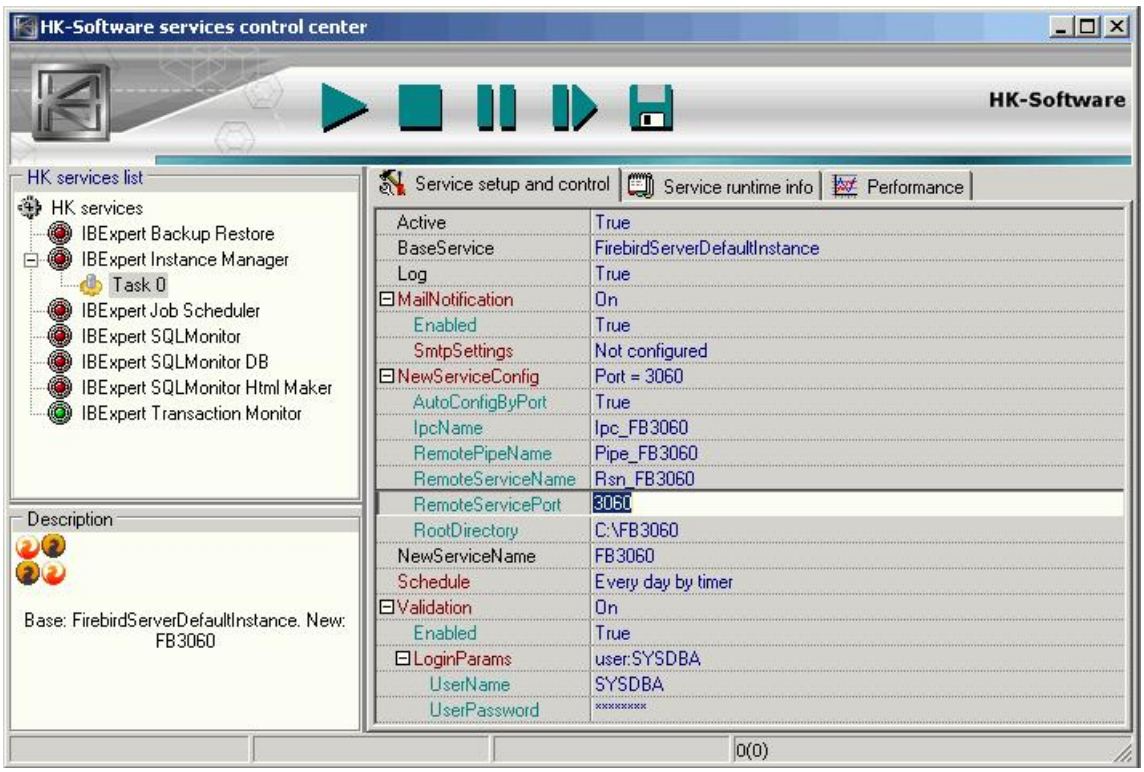
4. Select the IBExpertInstanceManager service. Right click on it and select *Add task*.



5. Click on this task on the left, and select the *BaseService* from the list of Firebird instances installed on your PC.

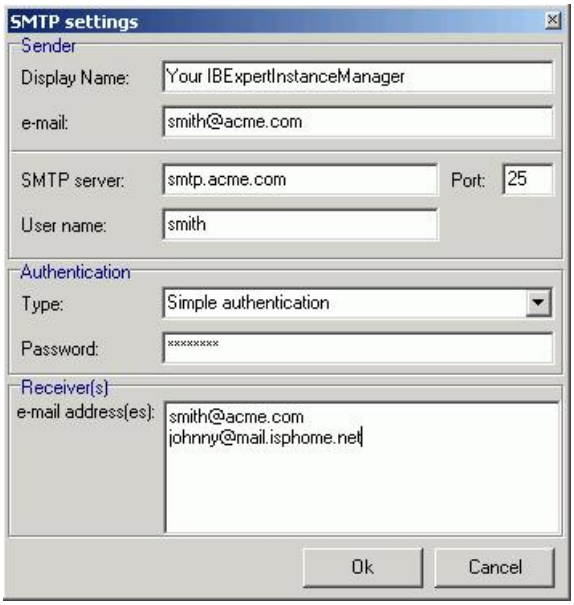


6. Set the port number for the Firebird instance you are going to create. All other instance configuration settings will be generated automatically.



7. Setup mail notification if required. To use this feature, set the *Enabled* parameter in the *MailNotification* section to *True*.

   The IBExpertInstanceManager uses a built-in SMTP client to send e-mails, so you need to set up the SMTP parameters in the task configuration to enable this to work properly. Simply double-click on the *SmtpSettings* option, to open the configuration dialog window.



In this dialog you should set up the *Sender*, *SMTP server configuration* and one or more recipients.

1. The *Schedule* offers *Daily*, *Monthly* or *Custom* specifications. Double-click on the *Schedule* option to open the schedule configuration dialog window.

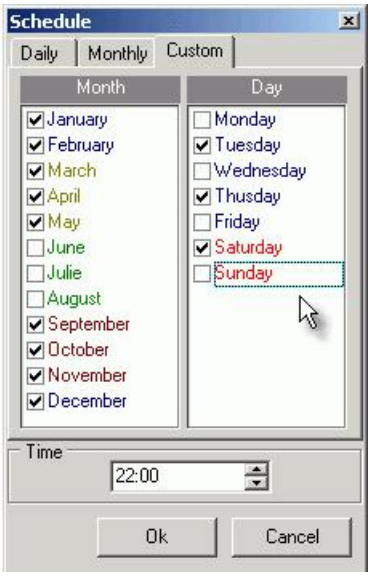
   Daily schedule:

   - every day at the specified time.
   - every *n*th day, starting from date.
   - every given day of week.

   Monthly schedule:

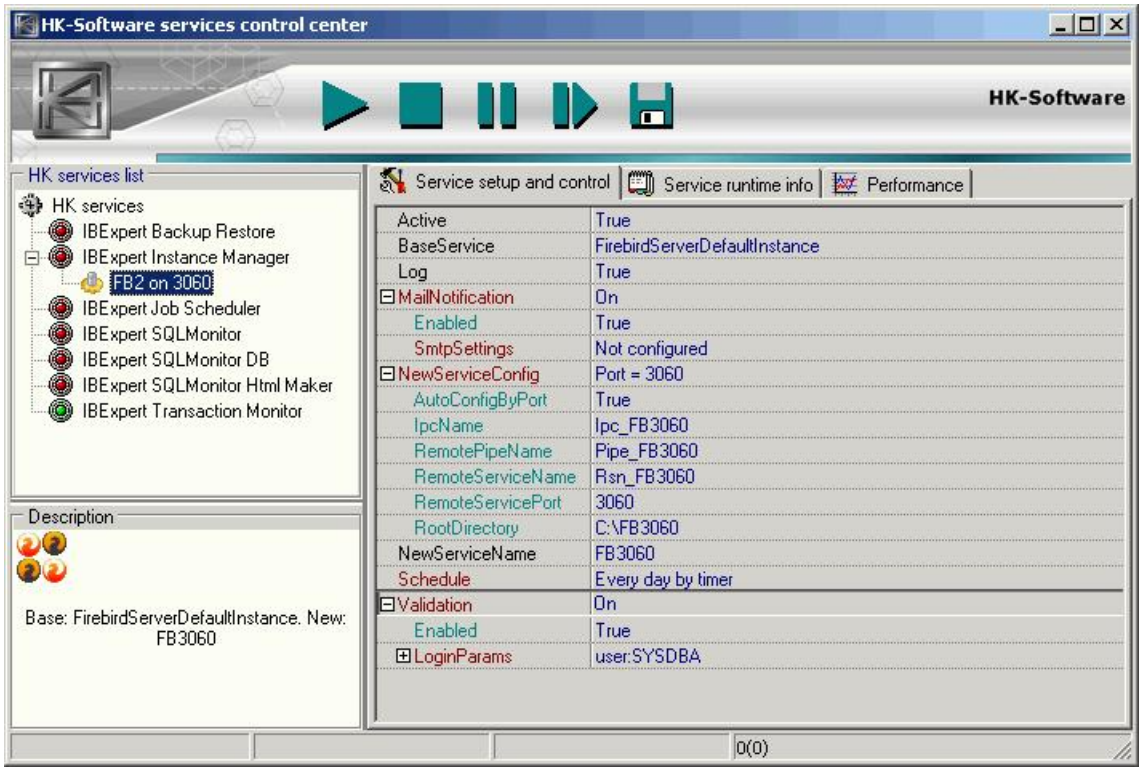
Every *n*th day of the selected months at the given time.

Custom schedule:

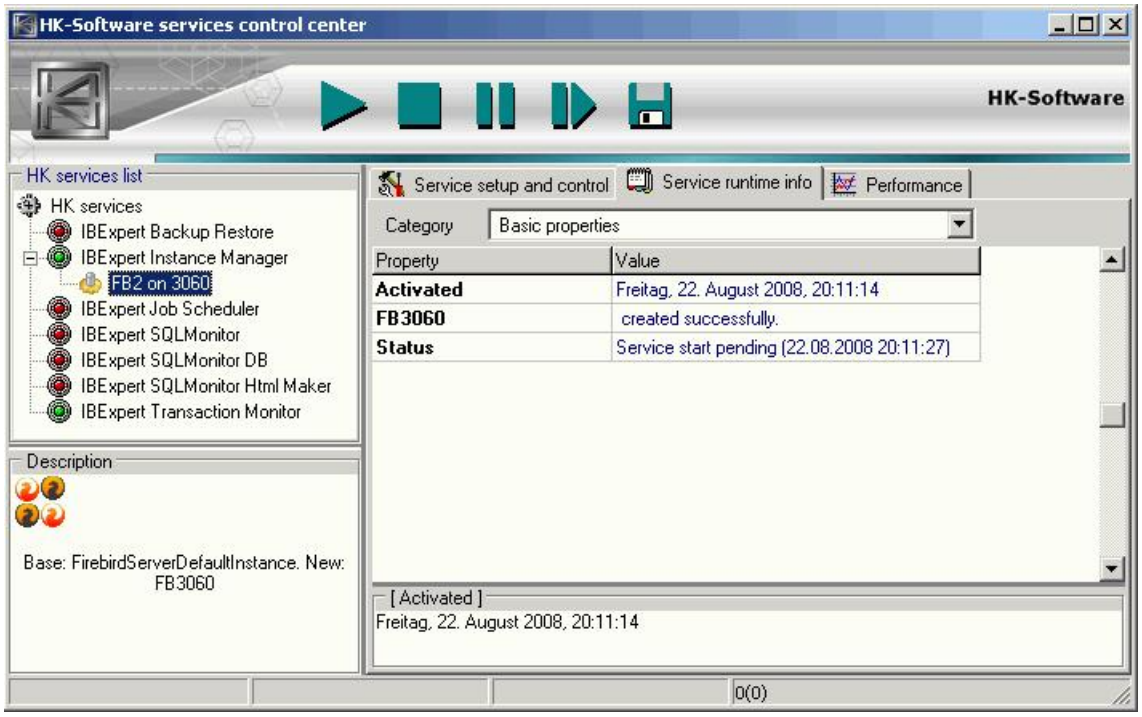


Selected days of every week of selected months at given time.

9. Set up validation parameters if needed. Validation is simply a test connection to the new instance's `security.fdb`, using the instance's port number.
10. Set the task's *Active* parameter to *True*.
11. To rename the task, click on the task name with the [Ctrl] key pressed down.



12. When you are happy with your specifications, they can be saved using the disk icon in the toolbar. Then you can simply run the service. When properly configured the running task should show runtime info on the first run. This can be viewed on the *Service runtime info* page.



IBExpertJobScheduler

... currently in work.

...currently in work.

What is the IBExpertJobScheduler?

IBExpertJobScheduler is a new module in the HK-Software Control Center.

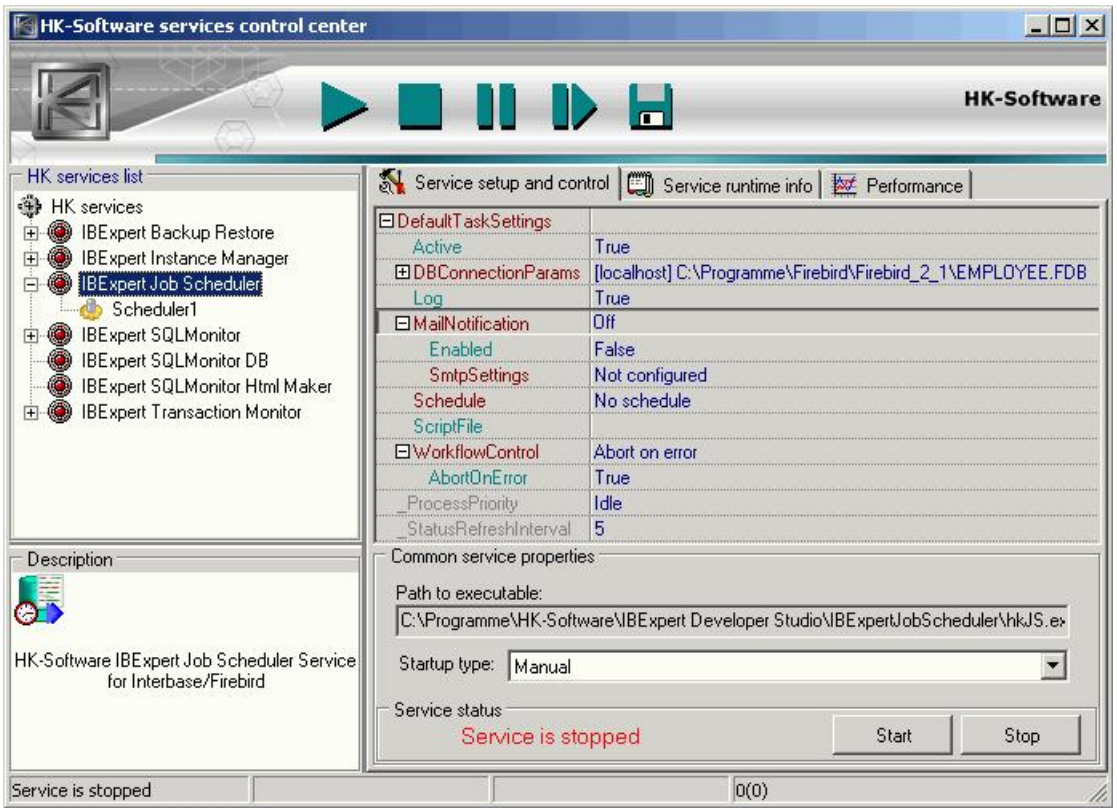
The IBExpert Junior VAR license or the VAR license entitles you to distribute the IBExpertJobScheduler with your application.

Setup and usage

Start the HK-Software Services Control Center, found in the IBExpert Services menu, and select *IBExpert Job Scheduler* in the *HK services list*.

We now need to configure the default task settings. As some parameters will remain the same for all further tasks (for example: SMTP settings), these should be configured first.

Expand the *DefaultTaskSettings* item on the *Service setup and control* page.



The following lists the various default settings and options available:

- Active
- Database connection configuration
- Mail notification
- Schedule
- Workflow Control

After configuring the default task settings, all new tasks will have this configuration when created. It is of course possible to alter specific options for individual tasks.

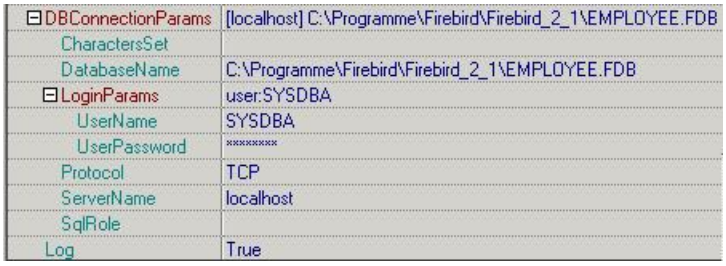
Default task settings

Active

When `True` then the task just created will be active (see illustration above).

Database connection configuration

The next step is to establish the database connection. All necessary properties can be configured in the *DBConnectionParams* section:



This is fairly self-explanatory; although should you require detailed information regarding Firebird/InterBase database connection parameters, please refer to the online IBExpert documentation.

Mail notification

The mail notification feature sends reports concerning the IBExpertJobScheduler activity. The service sends an e-mail message with log files attached when the job is completed.

To use this feature, set the *Enabled* parameter in the *MailNotification* section to *True*.

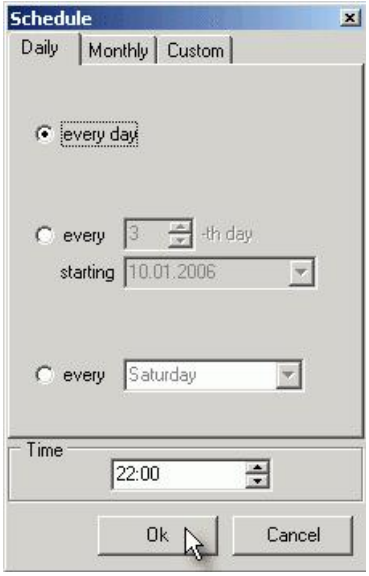
The IBExpertJobScheduler uses a built-in SMTP client to send e-mails, so you need to set up the SMTP parameters in the task configuration to enable this to work properly. Simply double-click on the *SmtpSettings* option, to open the configuration dialog window.

In this dialog you should set up the *Sender*, *SMTP server configuration* and one or more recipients.

Schedule

Double-click on the *Schedule* option to open the schedule configuration dialog window:

Daily schedule:

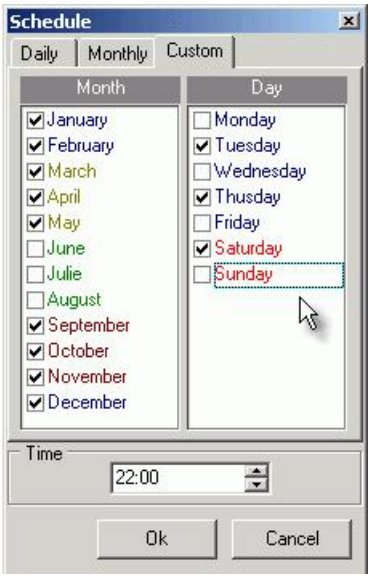


- every day at the specified time.
- every *n*th day, starting from date.
- every given day of week.

Monthly schedule:

Every *n*th day of the selected months at the given time.

Custom schedule:



Selected days of every week of selected months at given time.

Workflow control

Here you can specify what the IBExpertJobScheduler should do when it encounters an error.

_ProcessPriority

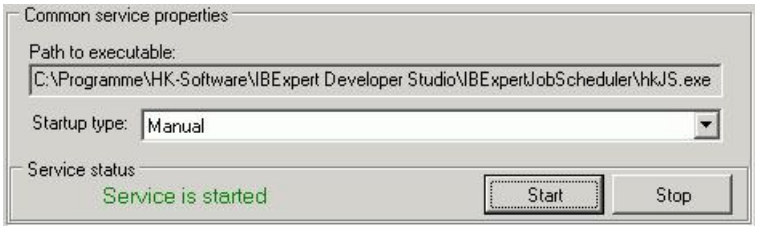
This parameter can be set to *Idle*, *Normal* or *High* (the default is *Idle*).

_StatusRefreshInterval

Here the refresh interval in seconds can be specified (default value is 5).

Common service properties

The path to the executable file, `hkJS.exe` is displayed. You can specify the *Startup type* selecting an option from the drop-down list (options: *Manual*, *Automatic* or *Disabled*).

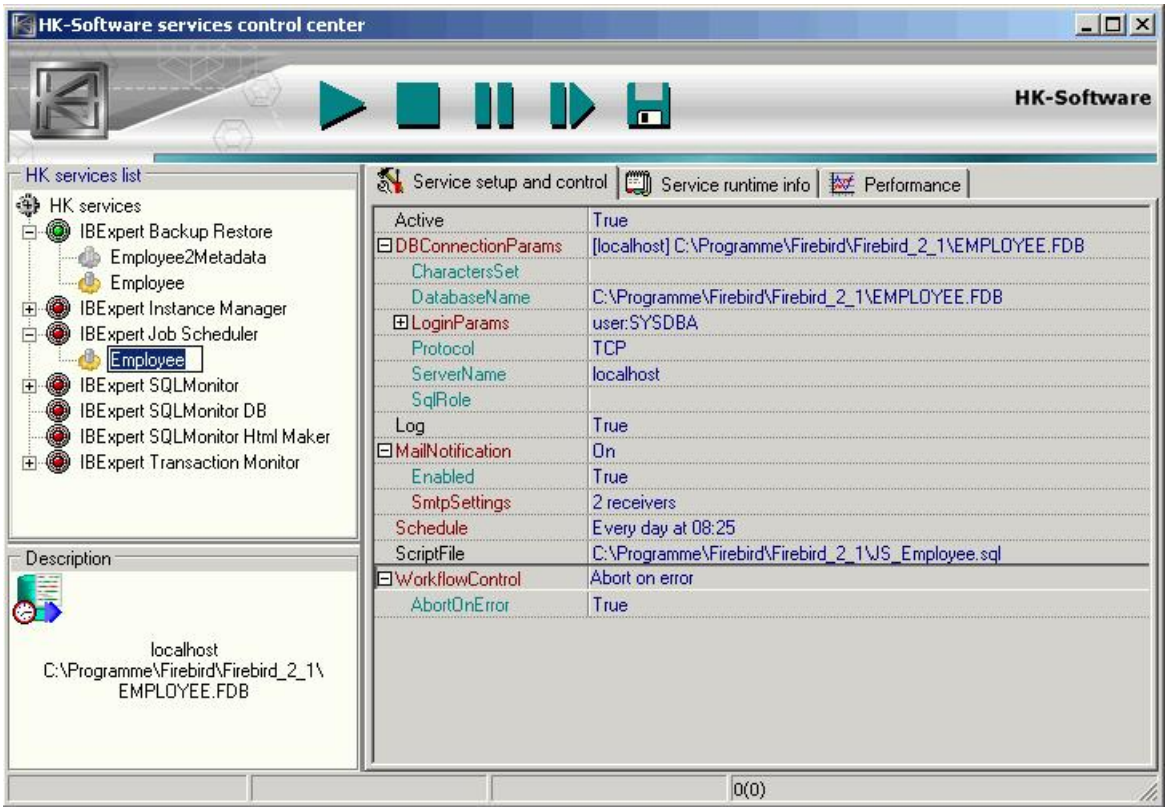


The *Service Status* can be viewed at the bottom of the window, and the *Start* and *Stop* buttons used to manually start or stop the service.

When you are happy with your specifications, they can be saved using the disk icon in the toolbar. After configuring the default task settings, all new tasks will have the same configuration when created. You can of course alter specific options for individual tasks if wished.

Preparing a task

To create individual job schedules, you now need to create a task. Right-click on the *IBExpert Job Scheduler* service's item in the SCC. Then click *Add task* in the popup menu. After that you will see the new task item (Task 0) under the *Job Scheduler* service's item. You may rename it by clicking on the name simultaneously holding the [Ctrl] key down. In the example below you can see a new task, renamed to *Employee*.



Alter your default settings if necessary. Then you can simply run the service.

IBExpertLive

What is IBExpertLive?

IBExpert KG has implemented a streaming system based on the Firebird database server, which publishes pictures and audio, as needed to view the presentations from the 2004 and 2005 Firebird Conferences. We will also be adding IBExpert tutorial videos enabling you to learn more about working with Firebird and InterBase with IBExpert.

IBExpertLive is part of the IBExpert Developer Studio. There is currently about 20GB of video data available, with around 100 hours of firebird-related presentations from last two Firebird Conferences and other events.

To use IBExpertLive, you need a firebird connection via Internet using port `13050` to our server on IP `80.237.154.78`. If it does not work, please check your firewall settings. For reporting any other questions/problems regarding IBExpertLive, please use the following contact addresses: E-mail: ibexpertlive@ibexpert.biz.

There might be some videos that are not working yet, even if they are on the list. Please bear with us; we'll have everything up and running as soon as possible.

The download address is: www.ibexpert.com/ibexpertlive/IBExpertLive_setup.exe.

Download and install IBExpertLive

IBExpertLive is installed as default, when installing the one of the IBExpert customer versions. Alternatively download the IBExpertLive setup file from: http://www.ibexpert.com/ibexpertlive/IBExpertLive_setup.exe, and save to your hard drive (e.g. `C:\Program Files\HK-Software`).

Start the `setup.exe` file and follow through the installation instructions.

When starting IBExpertLive for the first time, you will need to request a password. Simply enter your valid e-mail address and check *Request password*:

Your password will be sent to the e-mail address specified in a matter of minutes!

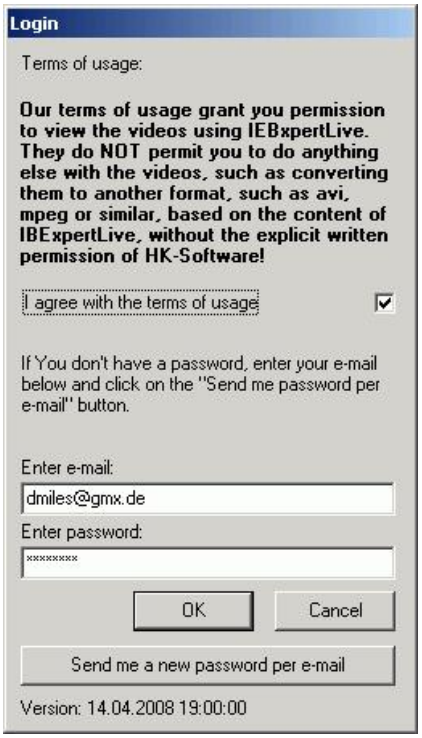
There might be some videos that are not working yet, even if they are on the list. Please bear with us; we'll have everything up and running as soon as possible.

Should you have any questions or encounter any problems please send an e-mail to ibexpertlive@ibexpert.biz.

Using IBExpertLive

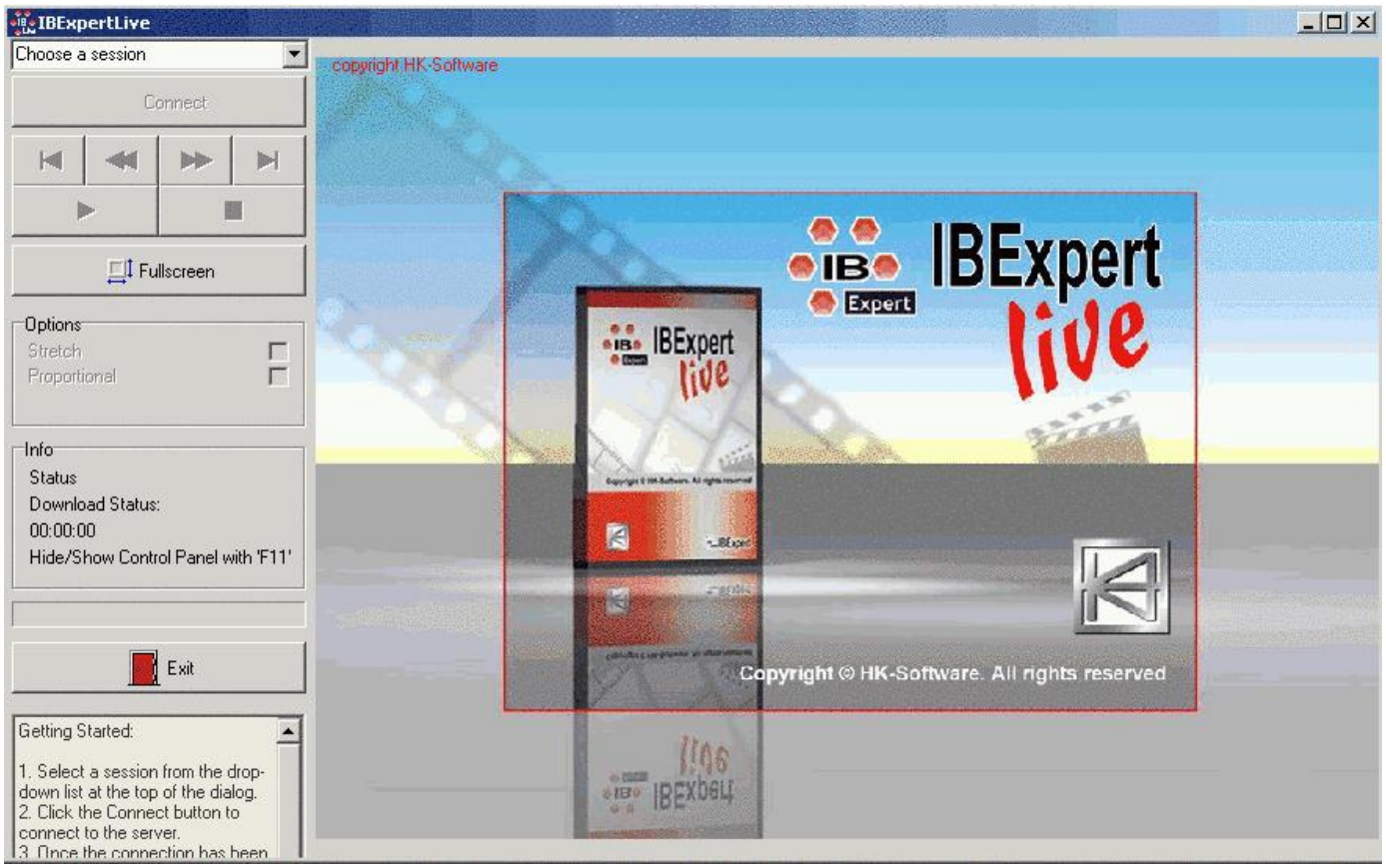
IBExpertLive is an extremely simple and self-explanatory application.

Start IBExpertLive by agreeing with the *Terms of Usage* (checkbox option), and then entering your valid e-mail and password. When starting IBExpertLive for the first time, you will need to apply for a password:



Every time IBExpertLive is started following the *Login*, the application checks for any available updates which are then automatically installed. In such a case IBExpertLive restarts and it is necessary to log in again.

You will see the IBExpertLive control panel on the left, and the video screen to the right:



Using the IBExpertLive control panel is intuitive. There are however a number of hidden functions (please refer to Keyboard shortcuts for details).

1. Select the session of your choice from the *Choose Video* drop-down list, at the top of the control panel.

2. Click the *Connect* button to connect to the server.

3. Once the connection has successfully been made, streaming starts automatically. The status is displayed in the *Info* box. If problems are incurred whilst attempting to make the connection, an error message appears.

4. Navigate the video using the upper row of directional buttons.

5. Use the *Fullscreen* button to switch between full screen mode and normal mode.

6. Adjust the image to fill the program window using the *Stretch* checkbox option.

7. The *Proportional* option can be used to adjust the image size in the program window proportionally to the image's side length.

Keyboard shortcuts

To make life easier, there are a few hidden keyboard shortcut functions in IBExpertLive:

[Ctrl + arrow to the left]	Rewind
[Ctrl + arrow to the right]	Fast forward
[Ctrl + upwards arrow]	Back to the beginning
[Ctrl + downwards arrow]	Spring to the end
Double-click on the video screen	Switches between normal mode and full-screen mode
Space bar	Stop / play
[F]	Full screen / pause

Available Films

As films are being added all the time, please check the available sessions regularly using the IBExpertLive *Choose Video* pull-down list. All films are in either the English or German language (recognizable by the film title).

Status April 2008

300 FBCON2007 A10 Jason Chapman - FB School eng SQL Basics

001 HK German Tutorial: Grundlagen SQL und Einrichtung IBExpert Demodatenbank
002 HK German Tutorial: Einfache Firebird SQL Befehle
003 HK German Tutorial: Tabellen mit SQL verknüpfen
004 HK German Tutorial: Where Bedingungen
005 HK German Tutorial: Erstellen einer eigenen Datenbank
006 HK German Tutorial: Tabellen erstellen
007 HK German Tutorial: Tabellen mit Fremdschlüssel erstellen
008 HK German Tutorial: Benutzer erstellen und Rechte vergeben
009 HK German Tutorial: Datenbank Parameter und Hintergrundwissen
010 HK German Tutorial: Tabellen abfragen, Indizes erstellen und SQL Performance vergleichen
011 HK German Tutorial: Auswirkungen langer Char Felder
012 HK German Tutorial: UDF benutzerdefinierte Funktionen einbinden
013 HK German Tutorial: IBExpert Demo Database Collection
014 HK German Tutorial: Database Designer Entity Relationship Modeling
016 HK German Tutorial: Erstellen eines triggerbasierenden Transaktionslogs
017 HK German Tutorial: Logging in IBExpert und Trigger in der Employee Datenbank
018 HK German Tutorial: Sprachelemente für Stored Procedures
019 HK German Tutorial: Rekursionen in Prozeduren und erste eigene Prozeduren
020 HK German Tutorial: Prozeduren entwickeln und optimieren
021 HK German Tutorial: Trigger entwickeln für Transaktionsprotokolle
022 HK German Tutorial: Views und Updatable Views
023 HK German Tutorial: Views für mehrere Tabellen
024 HK German Tutorial: Wie funktioniert Firebird intern? I
025 HK German Tutorial: Wie funktioniert Firebird intern? II
026 HK German Tutorial: Wie funktioniert Firebird intern? III
027 HK German Tutorial: Wie funktioniert Firebird intern? IV
028 HK German Tutorial: FBConnections, gfix, Cache und sonstige Parameter festlegen
029 HK German Tutorial: IBETransactionMonitor, Server Properties, Backup Restore, Logging, Quellcodemanagement
030 HK German Tutorial: IBExpertSQLMonitor, Firebird ODBC und MS Access, Export, IBEBlock ODBC
031 HK German Tutorial: IBExpert Spezialfunktionen, Plananalyzer, Selektivität,Kommandozeilenprogramme, External Files
032 HK German Tutorial: Installation und Vergleich FB15-FB20, Performanceanalyse, IBExpert Doku, Extract Metadaten, Datenbanken reparieren
033 HK German Tutorial: Alias.conf, firebird.conf, Temp Pfad und Dateien
034 HK German Tutorial: firebird.log, Freeadhocudf
035 HK German Tutorial: Demodatabase Transaktionslog, rfunc UDF, Replikation
036 HK German Tutorial: Internet Firebirdverbindung mit Zebedee
037 HK German Tutorial: Testdaten generieren, Performancemessung, Indizes, Plan, Selektivität
038 HK German Tutorial: Set Statistics, Indexoptimierung, order by, Fremschlüsselindizes
039 HK German Tutorial: Datenbankstatistik, Backup Restore Optimierung
040 HK German Tutorial: Trigger statt FK, MGA, Versionierung
041 HK German Tutorial: SQL ,in,exists,updateable views,Performance, firebird.conf Konfiguration, lange Varchar, Cache
042 HK German Tutorial: external files,csv export
043 HK German Tutorial: IBExpert Export
044 HK German Tutorial: CSV Import Insertex, Tabelle in andere Datenbank kopieren
045 HK German Tutorial: IBExpert.usr Menüs einschränken, Sprachanpassung
046 HK German Tutorial: mit ibeblock Metadatenextract per Kommandozeile automatisieren und mit DLL in eigene Programme integrieren
047 HK German Tutorial: Metadatenextract mit Daten, Blobunterstützung in Scripts
048 HK German Tutorial: ibeblock ODBC Zugriff, MS Access Datenbanken einbinden,Daten von ODBC nach Firebird kopieren
049 HK German Tutorial: ibeblock Datenbanken verknüpfen
050 HK German Tutorial: Dateien importieren, Bilder importieren per Script
051 HK German Tutorial: Dateien per Script updaten
052 HK German Tutorial: Verbindungen über das Internet mit Zebedee verschlüsseln und komprimieren, Dyndns mit no-ip.com nutzen
053 HK German Tutorial: Performance Internetbetrieb von Datenbankservern, Pingzeit,Route
054 HK German Tutorial: HK Services, Transaction Monitor
055 HK German Tutorial: Delphi BDE Applikationen auf IBObjects umstellen mit GReplace
056 HK German Tutorial: Datenbanken nachträglich Character Set und Dialekt konvertieren
057 HK German Tutorial: Extrahieren von Daten und ausführen per DLL
058 HK German Tutorial: Applikation Optimieren durch Einsatz spezieller IBO Komponenten, IB_DSQL, IB_CURSOR
059 HK German Tutorial: Performancevergleich IBO Query, IBO Cursor, BDE Query etc.
060 HK German Tutorial: Performanceprobleme durch Autobackground Commit Close Open vermeiden, Master Detail
061 HK German Tutorial: Lazarus: Open Source Delphi für Windows und Linux, Zeos, AvERP Open Source Warenwirtschaft, Datenmodellierung, Laufzeitformular

110 FBCON2006 B01-Paul Ruizendaal - Solution Stacks Built on Firebird and PHP - Another Flame in the Lamp?
111 FBCON2006 C01-Milan Babuskov - Developing Cross Platform Aplications with Firebird and wxWidgets
112 FBCON2006 B02-Martijn Tonies - The Firebird System Tables
113 FBCON2006 C02-Mauritio Longo - Supporting Complex On Line Systems with Satellite Databases
114 FBCON2006 B03-Thomas Steinmaurer - Owner Migration the Easy Way
115 FBCON2006 C03-Andrew Morgan - Towards a Universal UDF Testing Framework
116 FBCON2006 B04-Björn Reimer/Dirk Baumeister - Firebird Clients and System Tables
117 FBCON2006 C04-Fikret Hasovic - Cross - Platform Development Using Lazarus
118 FBCON2006 A05-FirebirdFoundation - Opening and Welcome (just audio)

119 FBCON2006 A06-PaulReeves - Building Firebird on Windows and Linux
120 FBCON2006 B06-DmitrySibiryakov - Replication with IBReplicator
121 FBCON2006 C06-CarlosCantu - New Shutdown Modes and Backups in Firebird 2.0
122 FBCON2006 A07-DmitryYemanov - Cost-based Optimization and Statistics
123 FBCON2006 B07-ErickSasse - N-Tier applications with Firebird and RemObjects DataAbstract
124 FBCON2006 C07-HolgerKlemt - Creating modern database webaplications using Firebird, php and AJAX
125 FBCON2006 A08-IvanPrenosil - Data Types in Practice/Optimizing Counts(Dual Topics)
126 FBCON2006 B08-GaryFranklin/BillOliver - Real World Applications Using Firebird
127 FBCON2006 C08-AlexSkvirski - Firebird Connectivity Tools or is there any Performance loss out there?
128 FBCON2006 A09-RomanRokytskyy - External routines:interface, usage and possibilities
129 FBCON2006 B09-MauricioLongo - Applications with Morfik WebOS and Firebird
130 FBCON2006 C09-DmitriKouzmenko - Optimizing Server Performance
131 FBCON2006 A10-HolgerKlemt - Creating Transaction Logs in Interbase
132 FBCON2006 B10-MilanBabuskov - Managing Metadata Chaanges
133 FBCON2006 C10-GaryFranklin/BillOliver - Delivering and using the Vulcan embeddet Server as Part of SAS
134 FBCON2006 D11-VladislavHorsun - New SQL Features in coming Versions of Firebird
135 FBCON2006 B12-RomanRokytskyy - Jaybird new release new features
136 FBCON2006 C12-DmitriKouzmenko - Database Health and Corruption

Roadshow Hamburg2006 Part1
Roadshow Hamburg2006 Part2

101 HK Presentation: Improving performance with IBExpert (Budapest 02/2005)
102 HK Presentation: A review of IBExpert's range of functions (Budapest 02/2005)

FDD 1
FDD 2

151 FBCON2005 Ann Harrison - Detecting correcting and preventing database corruption (FBC2005)
153 FBCON2005 Nando Dessena - Deploying Firebird transparently on Windows (FBC2005)
154 FBCON2005 Paul Reeves - From Basic to Advanced ISQL scripting (FBC2005)
150 FBCON2005 Andrew Morgan - Embedding and using sophisticated mathematics in Firebird (FBC2005)
155 FBCON2005 Fikret Hasovic - Open source Delphi (FBC2005)
156 FBCON2005 Luiz Paulo de Oliveira Santos - Firebird API in Delphi Lazarus and Free Pascal (FBC2005)
157 FBCON2005 Jason Wharton - IB Objects for Newbies (FBC2005)
158 FBCON2005 Luiz Paulo de Oliveira Santos - Techniques for migrating from MySQL to Firebird (FBC2005)
159 FBCON2005 Stefan Heymann - What Developers Should Know about Character Sets and Unicode etc (FBC2005)
160 FBCON2005 Lucas Franzen - STORED PROCEDURES (FBC2005)
152 FBCON2005 Evgeney Putilin - Firebird and Java Stored Procedures (FBC2005)
161 FBCON2005 Claus Heeg - Migration and integration of other databases into Firebird using Cold Fusion (FBC2005)
162 FBCON2005 Holger Klemt - Setting Up a bidirectional Replication based on EXECUTE STATEMENT Commands (FBC2005)
163 FBCON2005 Paul Ruizendaal - Solution Stacks Built on Firebird and PHP - Another Flame in the LAMP (FBC2005)
164 FBCON2005 Pavel Cisar - Firebird Quality Assurance (FBC2005)
165 FBCON2005 Dmitri Kouzmenko - Firebird Performance Optimization for Different Applications (FBC2005)
166 FBCON2005 Thomas Steinmaurer - Audit Trails Transaction Log Redo with the IB LogManager product family (FBC2005)
167 FBCON2005 Milan Babuskov - The power of Firebird events (FBC2005)
168 FBCON2005 Martijn Tonies - The Firebird system tables (FBC2005)
169 FBCON2005 Kim Madsen - SOA using kbmMW (FBC2005)
170 FBCON2005 Daniel Magin - Developing DataBase ASP.net Application with Delphi 2006 (FBC2005)
171 FBCON2005 Mauricio Longo - Dynamic Databases - A Conceptual Overview (FBC2005)
172 FBCON2005 Carlos Cantu - PSQL in Action (FBC2005)
173 FBCON2005 Jim Starkey - Configuring Firebird and Vulcan (FBC2005)
174 FBCON2005 Jim Starkey - Vulcan status features and goals (FBC2005)
175 FBCON2005 Arno Brinkman - Understanding the Optimizer I (FBC2005)
176 FBCON2005 Paul Beach Dmitry Yemanov - Firebird future development (FBC2005)
177 FBCON2005 Mauricio Longo - FireQ - Firebird Based Messaging Infrastructure (FBC2005)
178 FBCON2005 Roman Rokytskyy - JayBird - JCA/JDBC driver for Firebird (FBC2005)
179 FBCON2005 Roman Rokytskyy - JayBird - JCA/JDBC driver for Firebird (FBC2005)
180 FBCON2005 Lester Caine - Firebird on PHP Integrate or Abstract (FBC2005)
181 FBCON2005 Jeanot Bijpost - An introduction to Cathedron (FBC2005)
182 FBCON2005 Andrew Morgan - Creating and managing recursive structures (FBC2005)
183 FBCON2005 Holger Klemt - Server Performance - How to make your application run faster (FBC2005)
184 FBCON2005 Ann Harrison - First steps in performance tuning (FBC2005)
185 FBCON2005 Milan Babuskov - FlameRobin - administration tool for Firebird DBMS (FBC2005)
186 FBCON2005 Claus Heeg - Building ERP web applications based on Firebird and cold Fusion (FBC2005)
187 FBCON2005 Serg Vostrikov - Getting Started with FIBPlus (FBC2005)
188 FBCON2005 Arno Brinkman Dmitry Yemanov - Under the hood Data access paths (FBC2005)
189 FBCON2005 Pavel Cisar - Making your own Firebird PowerTools with Python (FBC2005)
190 FBCON2005 Stefan Heymann - Using Firebird for Quality Management Software (FBC2005)
191 FBCON2005 Serg Vostrikov - Special FIBPlus features network traffic optimization and FIBPlus Repository (FBC2005)
192 FBCON2005 Fikret Hasovic - Firebird in n-tier setup with Delphi and kbmMW (FBC2005)
193 FBCON2005 Jeanot Bijpost - From Model Driven Development to Model Driven Architectures (FBC2005)
194 FBCON2005 Alex Peshkov - New security features in Firebird 2.0 (FBC2005)
196 FBCON2005 Arno Brinkman - Understanding the Optimizer II (FBC2005)

202 FBCON2004 Helen Borrie - Creating a shop (the new Firebird Example Database) (FBC2004)
203 FBCON2004 Helen Borrie - Stocking the Shelves and Browsing the store (FBC2004)
204 FBCON2004 Arno Brinkmann - Understanding the Optimizer in Firebird (FBC2004)
205 FBCON2004 Arno Brinkmann - The Optimizer in SQL Examples (FBC2004)
206 FBCON2004 Pavel Cisar - Wrestling Firebird (FBC2004)
207 FBCON2004 Pavel Cisar - Firebird QA (FBC2004)

208 FBCON2004 Lucas Franzen - STORED PROCEDURES I (FBC2004)
209 FBCON2004 Lucas Franzen - STORED PROCEDURES II (FBC2004)
210 FBCON2004 Ann Harrison - Lock print (FBC2004)
211 FBCON2004 Frank Ingermann - Client Performance (FBC2004)
212 FBCON2004 Frank Ingermann - FBFreeDB (FBC2004)
213 FBCON2004 Holger Klemt - The Power of "Execute Statement" (FBC2004)
214 FBCON2004 Holger Klemt - corrupt Databases, examples and Solutions (FBC2004)
215 FBCON2004 Holger Klemt - Firebird performance Workshop (FBC2004)
216 FBCON2004 Manuel Morbitzer - PHP and Firebird (FBC2004)
217 FBCON2004 Manuel Morbitzer - Firebird and Visual Studio .NET (FBC2004)
218 FBCON2004 Paul Reeves - Building Firebird Installation Kits for Win32 (FBC2004)
219 FBCON2004 Paul Reeves - Firebird System Tables (FBC2004)
220 FBCON2004 Nicolay Samofatov - External tables (FBC2004)
221 FBCON2004 Nicolay Samofatov - new backup technology (FBC2004)
222 FBCON2004 Jim Starkey - Vulcan Architecture (FBC2004)
223 FBCON2004 Jim Starkey - Vulcan Design Goals (FBC2004)
224 FBCON2004 Thomas Steinmaurer - Serverseitige Programmier-Techniken (FBC2004)
225 FBCON2004 Thomas Steinmaurer - Neuerungen in Firebird 1.5 (FBC2004)
226 FBCON2004 Martijn Tonies - An Introduction to Firebird for database developers (FBC2004)
227 FBCON2004 Martijn Tonies - The Firebird PSQL language (FBC2004)
228 FBCON2004 Jason Wharton - IBO and Firebird / IBO Advanced (FBC2004)
229 FBCON2004 Hilmar Brodner - AvERP I Grundlagen und Administration (FBC2004)
231 FBCON2004 Paul Ruizendaal - Moving applications from Oracle to Firebird (FBC2004)
232 FBCON2004 Artur Anjos - Using Firebird over the Internet (FBC2004)
233 FBCON2004 Bastian Morbitzer - PHPtree - Firebird basierende PHP Anwendung für Dokumentation und Hilfssystem (FBC2004)
234 FBCON2004 Bastian Morbitzer - The Future of a global Firebird Online Documentation System (FBC2004)
200 FBCON2004 Frank Ingermann - The Sparkey of the year Event (FBC2004)
201 FBCON2004 Ann Harrison - QLI (FBC2004)
235 FBCON2004 Lester Caine - PHP - Life after Builder6 (FBC2004)
236 FBCON2004 Marc o Dunehue - Firebird and Java I (FBC2004)
237 FBCON2004 Marc o Donehue - Firebird and Java II (FBC2004)
238 FBCON2004 Bernd UA - Delphi 8 and Firebird .NET Provider (FBC2004)

IBExpertSQLMonitor

What is IBExpertSQLMonitor?

IBExpertSQLMonitor is a Firebird/InterBase administrator/developer tool, combining SQL monitor functionality with server performance monitoring and additional security features. SQL monitor ability is access library independent, so you can log SQL traffic made by any components or tools connecting to a Firebird/InterBase server by TCP/IP.

The main module – SQL Proxy - is a proxy that works between client and server and maps all traffic from one TCP/IP port/address combination to another. This module logs SQL traffic and calculates traffic statistics. SQL Proxy also works as a simple firewall between clients and server and provides corresponding functionality.

If you need to see SQL logs, made by SQL Proxy, in HTML format – just use the StatToHtml service. This module is used to transform logs and statistics into HTML form. It can also filter logs by execution time, enabling you to see only time-consuming statements.

If your logs are to be stored in a database – just use the StatToDB service, specially made to write logs into a selected Firebird/InterBase database, enabling you further analyze the contents.

All modules are controlled by a single HK-Software Services Control Center (SCC) application. Using the SCC you can start/stop any of the IBExpertSQLMonitor services and change any available settings, to set up the configuration you need. It is also possible to view all running services' runtime info, Firebird/InterBase client server traffic logs and statistics on the SCC interface.

There are two versions with some limitations in the download and customer areas on the IBExpert web site:

- **Trial version:** limited to localhost and protocol for one session only. A copy of the IBExpertSQLMonitor is included in the IBExpert Developer Studio.
- **Customer version:** limited to local access (localhost) and protocol for unlimited sessions. Please refer to for further information.

Download the free Trial Version (part of the IBExpert Developer Studio): http://www.ibexpert.com/download/other_files, file name: `IBMonitor_setup_trial.exe`.

Download Version for IBExpert Customers: http://www.ibexpert.com/customer/IBExpertNetworkMonitorFull.zip.

IBExpertSQLMonitor features

- Access library independent SQL monitor, with plan retrieval and execution time logging;
- Filters SQL commands using an include/exclude template. For example, if you don't wish to log system (containing `RDB$`) traffic or want to log only `SALES` table related statements;
- Calculates traffic statistics (i/o bytes, statements count) by host names and sessions;
- Transforms all logs and stats into HTML form if needed, and uploads it on a selected ftp *;
- Filters logged statements by execution time, for example, if you wish to see only time-consuming commands *;
- Saves all logs into a selected database, if needed for further analysis **;
- Client/server traffic and performance runtime info presentation in tables and diagrams (common statistics, active connections, etc.);
- Separate services with flexible setup for specific functionalities;
- Single Control Center for all modules. Also used for runtime info monitoring.
- Basic firewall functionality for better server security.

* using separate StatToHtml service ** using separate StatToDB service

IBExpertSQLMonitor licenses

You wish to purchase the software for installation on a server with remote access?

- Limited to 10 active sessions logged, remote and local access: EUR 199.00
- Limited to 100 active sessions logged, remote and local access: EUR 499.00
- Unlimited active sessions logged, remote and local access: Price on request.

At the moment IBExpertSQLMonitor only works on Windows, no Linux version available yet.

All customers resident in Germany or other EU member nations may order directly by e-mail, fax or mail (please refer to contact for details). Please do not forget to include your invoice address and your VAT or sales tax ID number, along with the product description, quantity and registration information. Should you require an original invoice copy, please let us know.

If you wish to pay by credit card, or you are resident in a non-EU country, please order in our online shop.

Download and install IBExpertSQLMonitor

Both the Trial and the Customer versions can be downloaded from the IBExpert website:

- **Trial version:** limited to localhost and protocol for one session only.
- **Customer version:** limited to local access (localhost) and protocol for unlimited sessions.

The Trial version is incorporated in the IBExpert Developer Studio Trial Version which can be downloaded here: http://www.ibexpert.com/download/setup_trial.exe

The download version for registered IBExpert Customers is: http://www.ibexpert.com/customer/IBExpertNetworkMonitorFull.zip

Making the connection

For both versions, you need to take the following steps:

- Start the *HK Service Config Center* `hkscc.exe`.
- Bind port should be `3050`
- Bind IP should be `127.0.0.2`
- Map IP should be `127.0.0.1`
- Map port should be the InterBase default port `3050`.

You can change these values if needed, but the following description is based on these default values.

- Click left on SQL proxy.
- Open *Log Levels* and set the level you need for all operations.
- Set `log dir` to a directory where the log files should be saved (default `c:\temp\`).
- Set *StatsSaveInterval* to, for example, `15` (file is stored every 15 seconds).
- Click on *Save* button at the top of the form.
- Click on *Start* button at the top of the form.

Now the proxy should work, log files are saved every 15 seconds and stored in the directory `c:\temp\`.

Setting up the HTML Service

- Click left on *StatToHtml*.
- Set `log dir` to the same place as above in SQL proxy.
- Set `tmpDir` to a directory where HTML files should be stored.
- Check *StatsSaveInterval* default, for example, every 30 seconds.
- Check *WrapLineLength* (for example to 100 characters per line).
- If needed set up the FTP upload location.
- Click on *Save* button at the top of the form.
- Click on *Start* button at the top of the form.

Select the application

Start any database application with a changed server name. For example, when you typically use

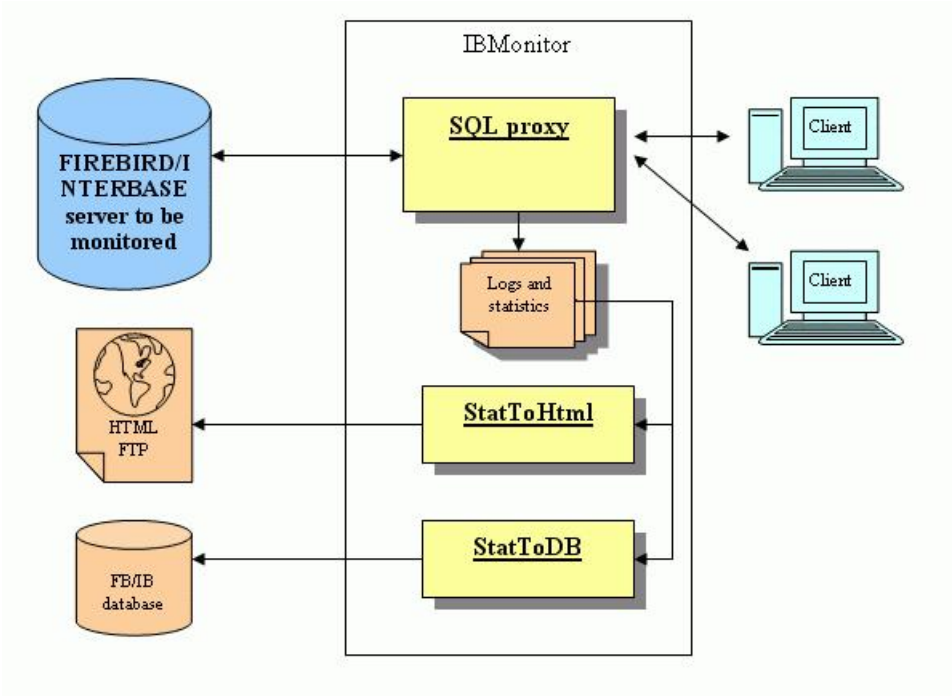
```
localhost:C:\path\file.fdb
```

now use

```
127.0.0.2:C:\path\file.fdb
```

After several seconds you can open the `index.html` file in `TmpDir` and see what has happened.

IBExpertSQLMonitor Workflow Scheme

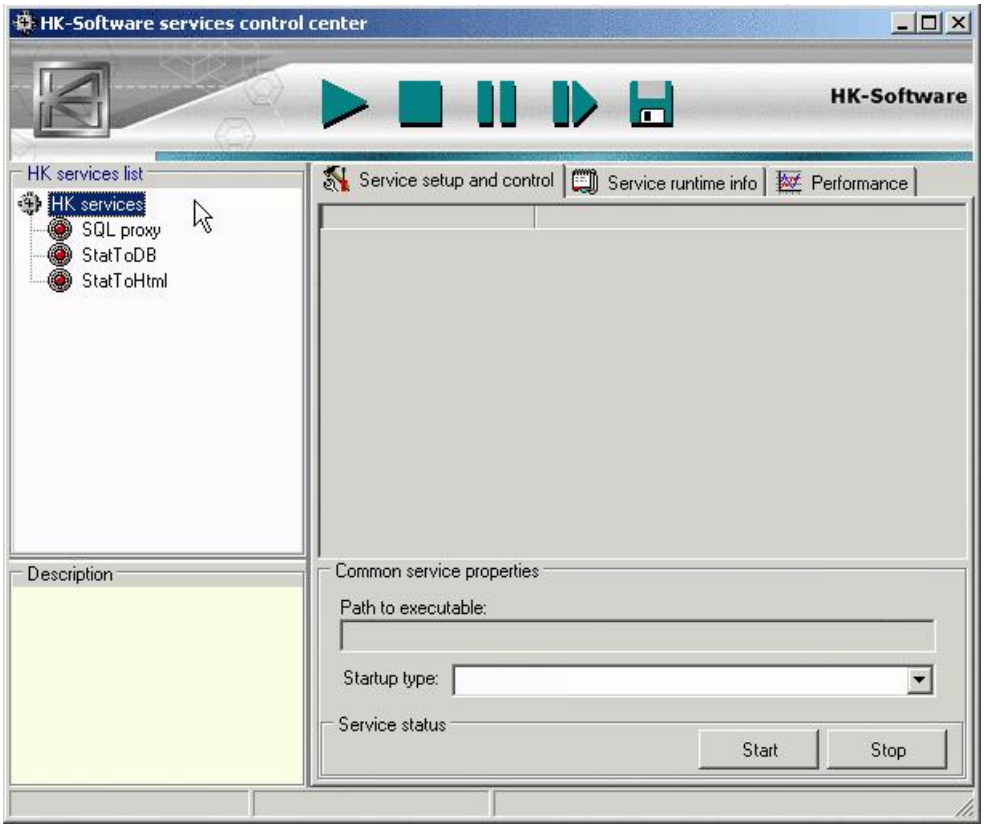


The Firebird/InterBase server listens to a specified IP and port (server IP and server port) and waits for client connections.

- SQL proxy listens to the TCP/IP protocol on other IP and port combinations (proxy IP and proxy port), logs all SQL, calculates traffic statistics, and then redirects traffic to the Firebird/InterBase server. It then gets a response from the server and redirects it to the client. This module also checks client validity and rejects a client connection request if that client is acknowledged to be invalid.
- StatToHtml gets log files and statistics collected by the SQL proxy and transforms them into HTML files, comfortable for end-user reading.
- StatToDB is used to store log files in a database, if the user wants to examine logs using SQL queries.

Services Control Center SCC

HK-Software Service Control Center (SCC) is intended to control all IBMonitor services using a single user interface. The SCC main window can be viewed below:



The upper panel is used to control services activity: *Run*, *Stop*, *Pause*, *estart* and *Save* buttons. The left side contains a list of installed services and the description of the service that is currently selected in the list.

The *Page* control, displayed as three tabs to the right of the services list, is used to set up the selected service and display its runtime information if the service is running.

- **First page:** *Service setup and control* – includes settings and properties of the selected service.
- **Second page:** contains the selected *Service Runtime Info*.
- **Third page:** *Performance* - contains module-specific performance diagrams.

More detailed description of the individual page contents can be found in the module descriptions.

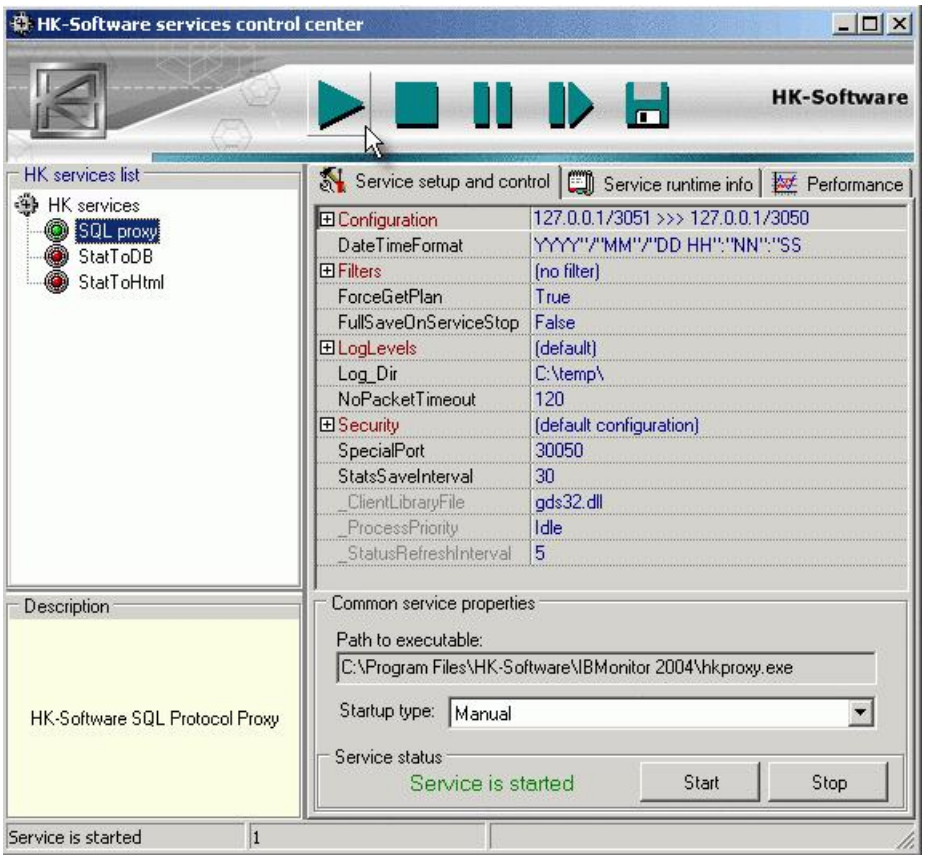
SQL Proxy: logging and security

SQL Monitoring and Logging: a quick start guide

Let's set up IBMonitor to do some simple logging while we're working in IBExpert with `employee.gdb` on localhost.

1. Start the SCC from the Windows *Start* menu. Select *SQL Proxy* in the services list.

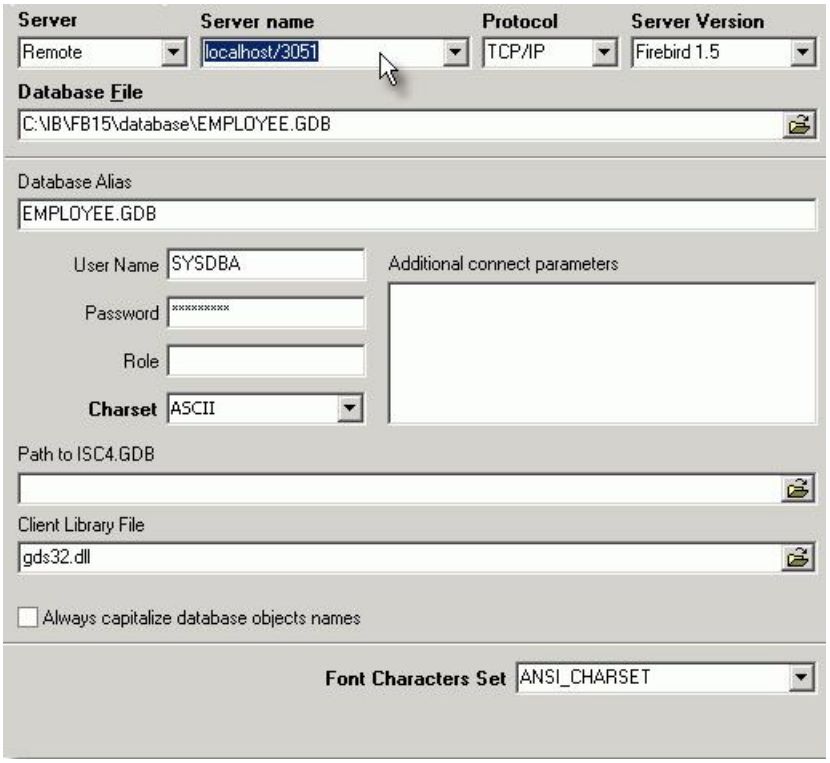
2. Set the proxy configuration settings to comply with those below, and then press the *Start* button at the top of the SCC form.



The *Service runtime info* and *Performance* pages display zeros at this stage, because we haven't had any traffic yet. So let's do it.

3. Start IBExpert and register `employee.gdb` on localhost. Use the *Test Connect* button to check that you've properly registered this database.

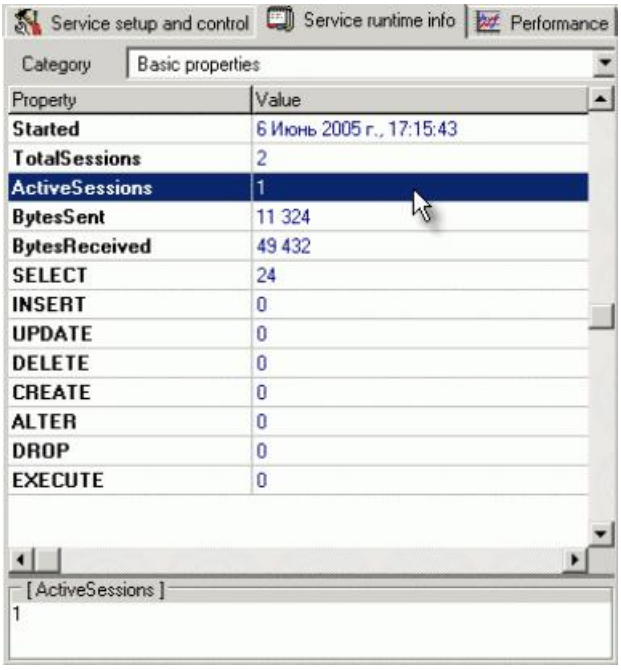
4. In the Database Properties window set the Server name to `localhost/3051`. The window should look something like this (the database file path may be different):



1. Connect from IBExpert to the `employee.gdb`, you have just configured.

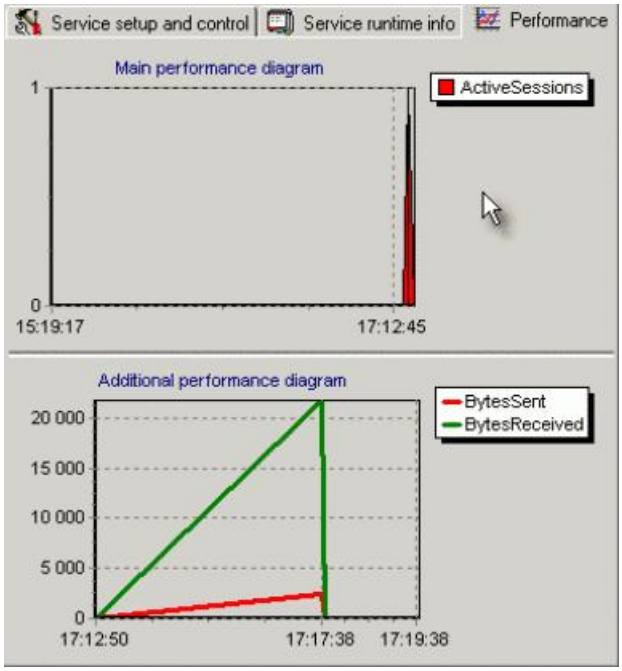
2. Now look at the Service *Runtime info* and *Performance* pages on the SCC. There you will see the traffic statistics made by IBExpert when you connected to `employee.gdb`.

The *Service runtime info* page contains the following information:



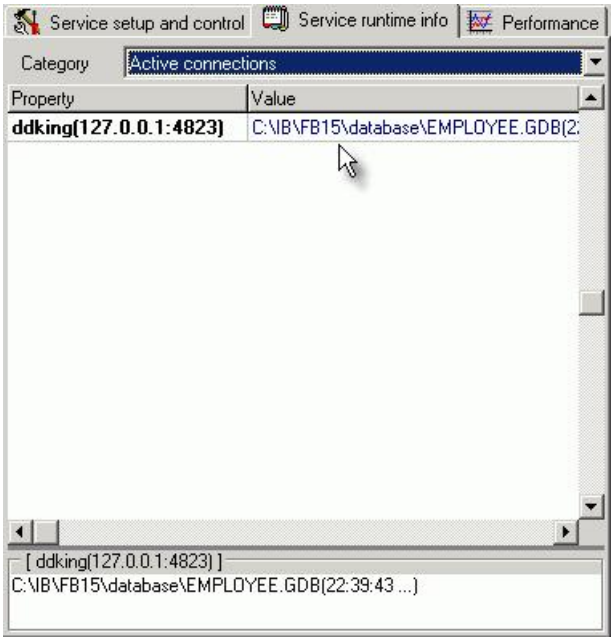
- **Total sessions count:** count of client/server sessions made via SQL proxy.
- **Active sessions count:** count of sessions currently opened via SQL proxy.
- **BytesSent, BytesReceived:** total volume of client/server traffic.
- **SELECT...EXECUTE:** total count of corresponding SQL statement calls.

The *Performance* page contains two charts showing *ActiveSessions*, *BytesSent* and *BytesReceived* values for a certain period of time.

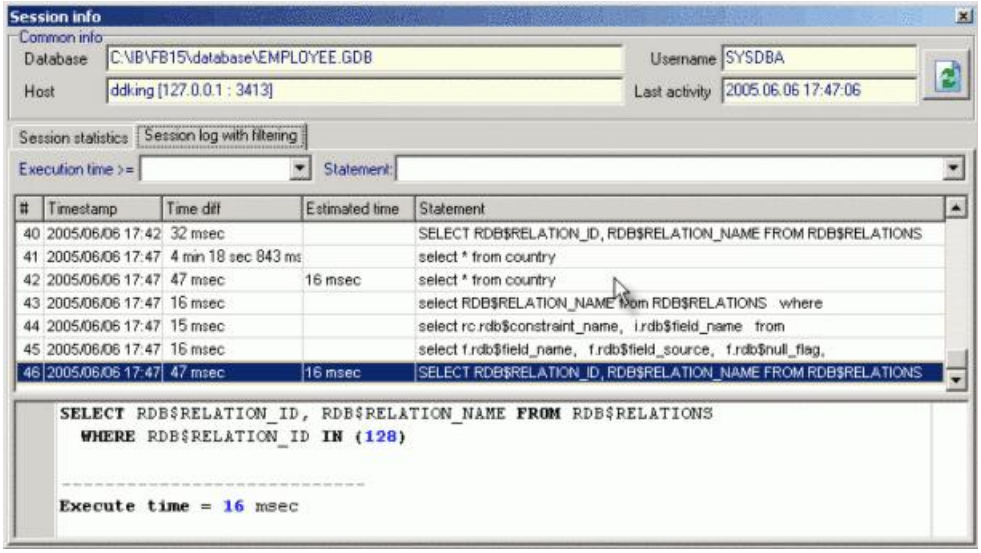


7. Execute a simple query from IBExpert. For example: `select * from country`. You can see that the information on the *Service runtime info* and *Performance* pages has changed.

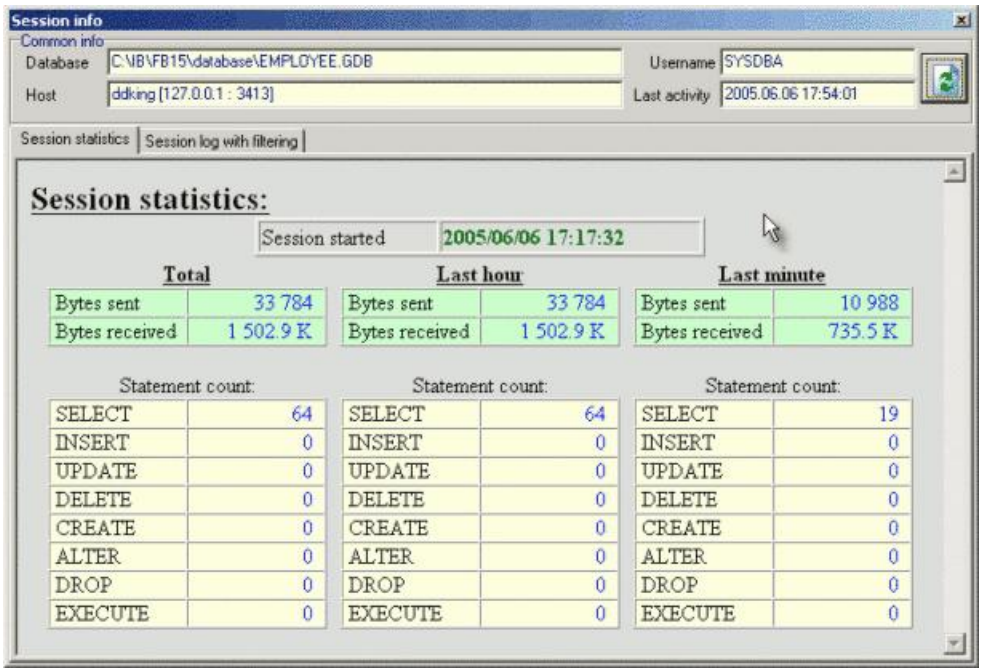
Now select the *Active connections* category on the *Service runtime info* page. There you should be able to see the `employee.gdb` connection made by IBExpert.



Double-click on any line with a connection description in the *Runtime info* table. After that you can see the *Session info* window, containing the log of executed SQL statements:



and traffic statistics for the selected connection:



If you can not see your connection in the *Active connections* category – maybe there has not been any traffic activity during the `NoPacketTimeout` time interval. In this case, select the *Timed out* connections category on the *Service runtime info* page.

Now let's look into the `Log_Dir` folder to find the log files we've just produced by our work in IBExpert. The default `log dir` folder is `C:\temp`. If you open it you should see a picture like this:

`_main_.log` is a single log file containing all notifications of clients' connect/disconnect attempts:

```
2005/05/11 22:22:50:734 [];127.0.0.1:1140;Connect to database: C:\IB\FB15\database\EMPLOYEE.GDB; User: SYSDBA
2005/05/11 22:22:51:109 [];127.0.0.1:1142;Connect to database: C:\IB\FB15\database\EMPLOYEE.GDB; User: SYSDBA
2005/05/11 22:23:30:296 [];127.0.0.1:1140;Disconnect database: C:\IB\FB15\database\EMPLOYEE.GDB; User: SYSDBA
```

Other `*.log` files are client/server sessions' logs. Our exercises with `employee.gdb` resulted in two log files: the first is produced by our connection and the second is produced by the additional IBExpert connection to the database.

SQL proxy settings

Configuration

Actually these properties are basic proxy settings: which IP and port the proxy should listen to and where it should redirect incoming requests.

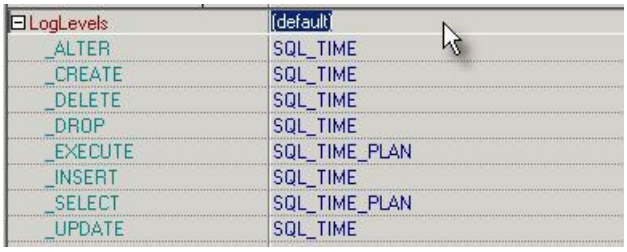
- **Proxy_IP and Proxy_Port:** IP and port that the proxy should listen to. You should use them as part of the server name in the database connection parameters to get your SQL traffic logged.
- **Server_IP and Server_Port:** IP and port of the Firebird/InterBase server to be monitored.

On the screenshot below you can see the default configuration: SQL proxy listens to port 3051 on localhost and redirects all requests to port 3050 (that is the default Firebird/InterBase server port) to localhost.

Log levels

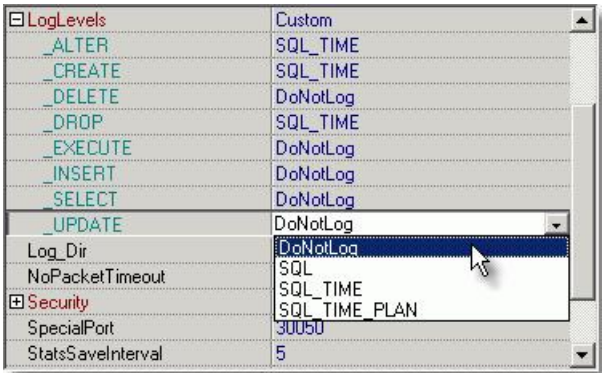
By default SQL proxy will log all SQL statements and their execution time. For SELECT and EXECUTE statements it will also log the statement execution plan (if the *ForceGetPlan* option is *True*).

Here is the default *LogLevels* options screenshot:



If you want to log only certain statement types (for example CREATE, ALTER and DROP), you can control SQL proxy behavior using the *LogLevels* property.

Here is the setup for our example:



After changing the properties in the SQL proxy setup as required, you should close the database connection, and then press the *Save* button in the SCC and answer *Yes* in this confirmation dialog:

Now if you connect to the Firebird/InterBase server via SQL proxy, it will log only CREATE, ALTER and DROP statements.

Filters

The other way to log only certain specified transactions is to set log filters. There are two kinds of filters in SQL proxy:

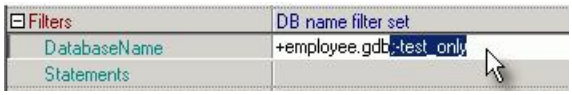
- Database name filter;
- SQL statement filter.

Both have the same simple syntax based on *Include* and *Exclude* templates.

The *Include* template should be started by the plus [+] sign, and the *Exclude* template by a minus [-] sign. Templates should be separated by semicolons.

For example, if you want to log only `employee.gdb`-related traffic you should set a corresponding *Include* template in the *DatabaseName* filter:

Now imagine that you have a lot of `employee.gdb` files placed in different folders. You want to log all of them, excluding `C:\test_only\employee.gdb`. In this case you should add an *Exclude* template to the *DatabaseName* filter:



The same logic is used when setting up the *Statements* filter. If you want to log only country-related statements set an *Include* template accordingly:

And if you wish to exclude update and alter statements from log files just add *Exclude* templates to the *Statements* filter:

Now let's see the filter working.

Execute or prepare SQL statements in IBExpert, such as, for example:

```
select * from country

select * from employee

update country set currency=currency

alter table country add test_field integer
```

Now double-click on a line with a connection description in the runtime info table (see below) to open the *Session info* window. In the table on the *Session log* page you can see only one statement:

```
select * from country
```

All other statements are excluded from the log by the statements filter.

So, if you can't achieve your required log configuration by setting *LogLevels* or just want to specify database name-based or statement-based log filters use the SQL proxy's *Filters* property.

Other settings

Section or parameter	Description
	Format of timestamps in log files. Default is `"YYYY"/"MM"/"DD HH":"NN":"SS"` `YYYY` - year `MM` - month `DD` - day `HH` - hour `NN` - minutes

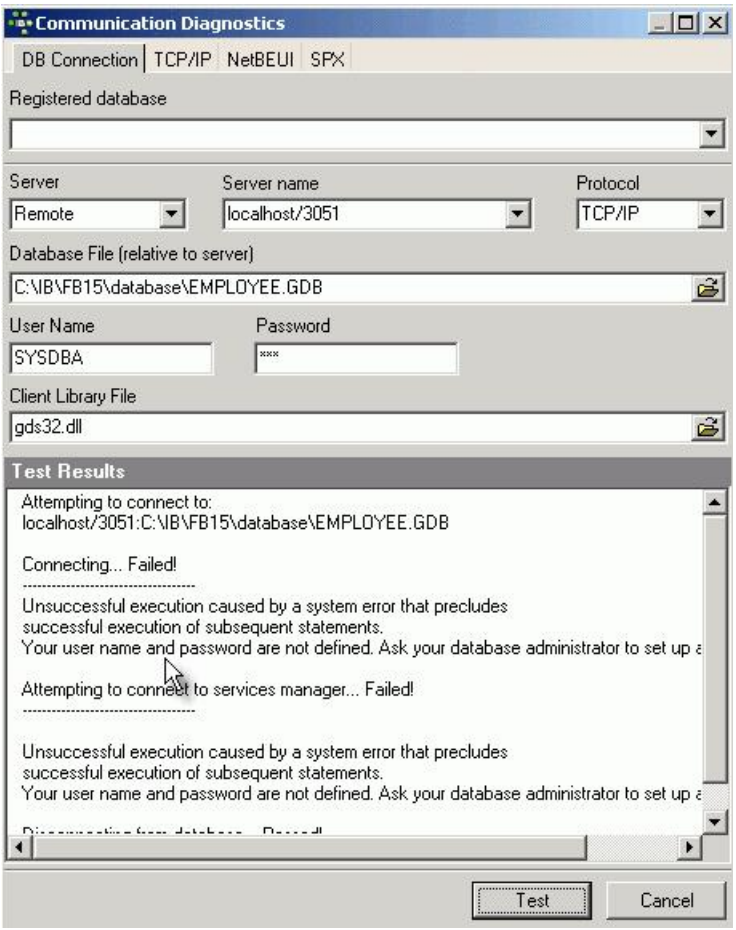
	ss - seconds Any characters in double quotes are constants.
`ForceGetPlan`	If *True* then SQL proxy will try to get an execution plan for every logged statement with help of additional connections to database. Default is *True*.
`SpecialPort`	The port used by SQL proxy for the force statement plan retrieval. Default is 3050.
`Log_Dir`	Path to the folder where SQL proxy will create the log files. Default is `c:\temp\`.
`NoPacketTimeout`	Connection timeout interval (in seconds). If no packets are passed through the client/server channel during this time, the connection is market as "timed out". Default is 120.
`FullSaveOnServiceStop`	If *False* then only changed connections statistics will be saved on service stop. Default is *False*.
`StatsSaveInterval`	Time interval (in seconds) defining the traffic statistics saving periodicity. Default is 5.
`_StatusRefreshInterval`	Time interval (in seconds) of the runtime info refresh. Default is 5. This means that every 5 seconds SQL proxy will send runtime info packets to the SCC.
`_ClientLibraryFile`	Firebird/InterBase client library file. SQL proxy may open additional connections to your databases for plan retrieval or checking user privileges (see the Security Features section). You can set which dll it should use as the client library. Default is `gds32.dll`.
`_ProcessPriority`	SQL proxy process priority (*Idle*, *Normal*). Default is *Idle*.

Security features

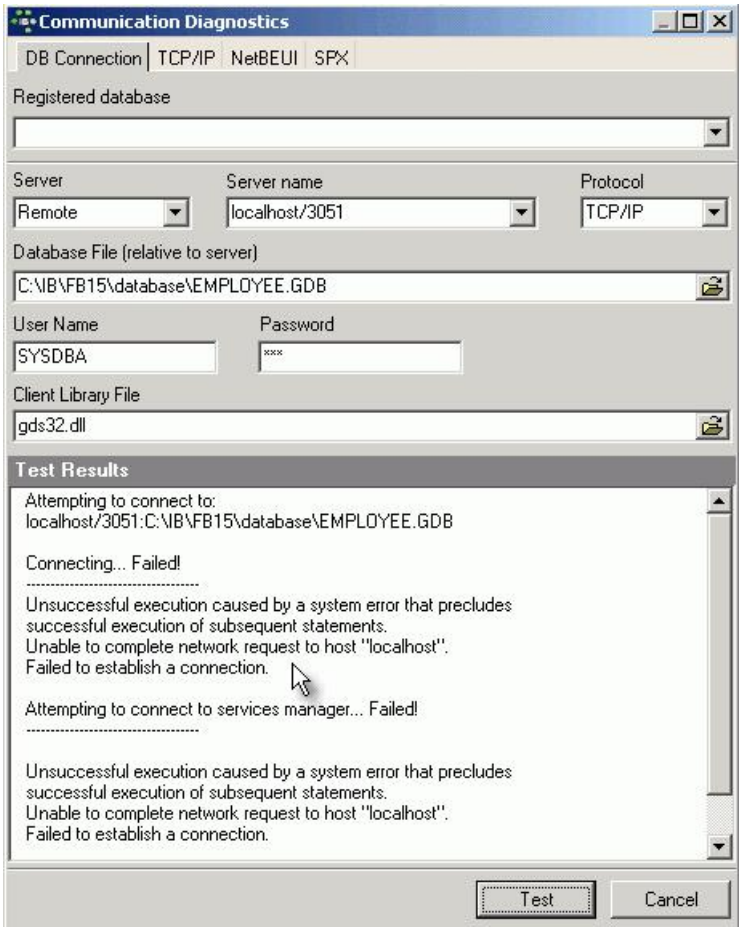
Bad password connections and BlockInterval

If you suspect that your Firebird/InterBase server may be subject to a brute force attack, this feature is useful. Using *BadPasswordAttemptCount* you can set the maximal count of invalid password connection attempts from one IP address. The default value of this property is 10. This means that when someone tries to connect to your Firebird/InterBase server via SQL proxy 10 times, their IP address will be blocked by SQL proxy for certain specified period of time. The block time interval in seconds is set by the *BlockInterval* property. The default value is 120 seconds.

Let's imitate such a situation by setting an invalid password in the `employee.gdb` connection we've made in IBExpert. The first few times we'll receive a *Your user name and password are not defined* message from the Firebird/InterBase server.

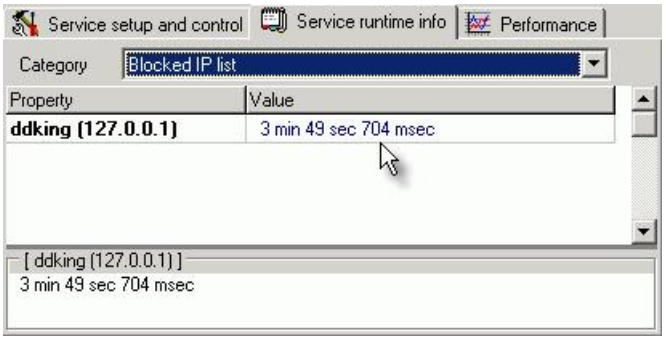


But if we click the *Test* button again a number of times we will see the following error message:



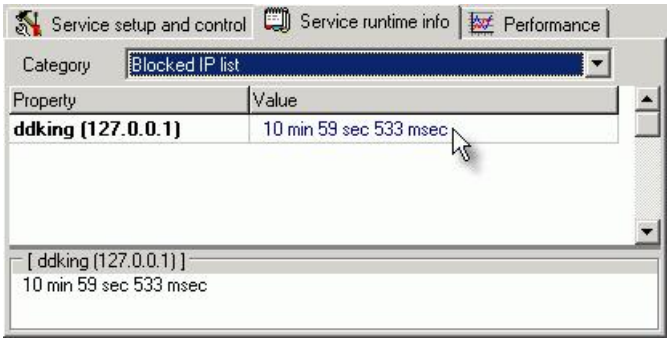
I.e. SQL proxy has marked our IP address as invalid and has blocked it for a certain amount of time.

A list of blocked IP addresses with blocking time can be seen in the SCC, on the *Service runtime info* page (category *Blocked IP list*).



During this blocking period any connection attempt from an invalid IP will be banned (even connections with valid username/password). "Bad guys" are blocked before any client/server packet exchange can take place. So no Firebird/InterBase server activity can be produced by such a fugitive client.

Each connection attempt during the blocking time will increase the blocking time. Here is the screenshot, made after some invalid password connections attempts:



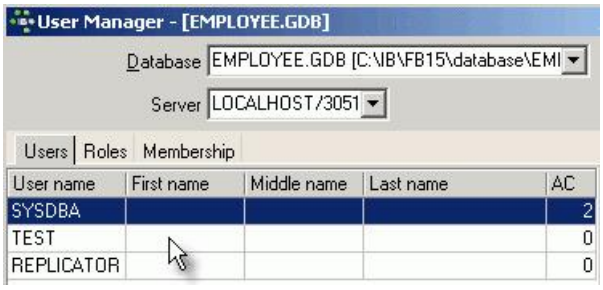
So, any persistent "bad guys" will be blocked for a very long time!

Check user privileges

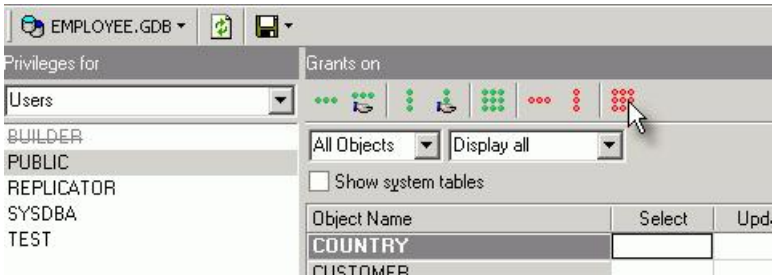
In some situations it may be useful to disable the database connection to users who haven't any privileges on database objects (tables, views, procedures, etc.). If you need such a functionality – you may use the *CheckUserPrivileges* option in SQL proxy. If this feature is switched on then SQL proxy will check if

the connecting user has any privileges (by querying the `RDB$USER_PRIVILEGES` table in the additional database connection). If the user has no privileges their connection request will be rejected and the client will receive the message: your user name and password are not defined.

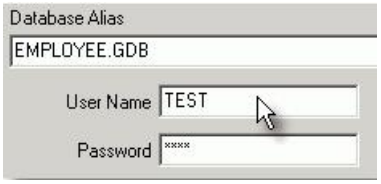
Let's demonstrate this function. Create a test user in IBExpert.



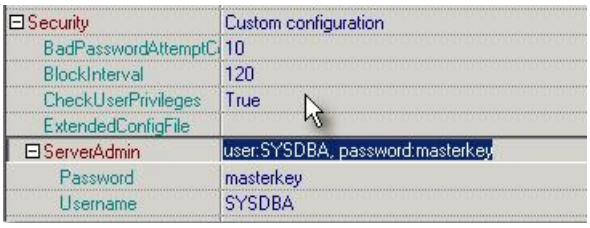
Now remove all privileges from `PUBLIC`:



Then disconnect from the database and change its registration info to make IBExpert connect to this database with our test user.



Now stop SQL proxy and activate the *CheckUserPrivileges* property:



Don't forget to specify the server admin login settings (actually it should be the user, who has a `SELECT` privilege on the table `RDB$USER_PRIVILEGES` in `employee.gdb`). This is necessary for SQL proxy to establish an additional connection to check client user privileges.

Now let's try to connect to our database using the unprivileged user. This should be the result:

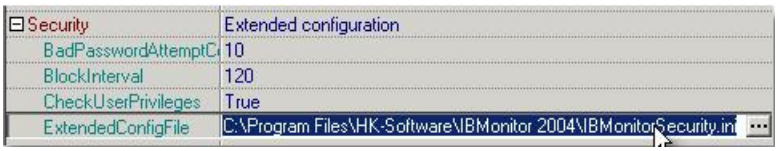
By default this function is deactivated.

Extended security configuration

The extended security configuration includes the following features:

- valid and invalid IP address lists,
- valid users list for certain specified databases,
- external application execution, when a connection request comes from certain specified IP addresses (for example to send an e-mail notification to the server administrator).

All this is configured by an `INI` file. If such a file already exists, all you need to do is to set its name in SQL proxy's *ExtendedConfigFile*, and then restart it.



Let's take a look at the extended security configuration file syntax, using the following example:

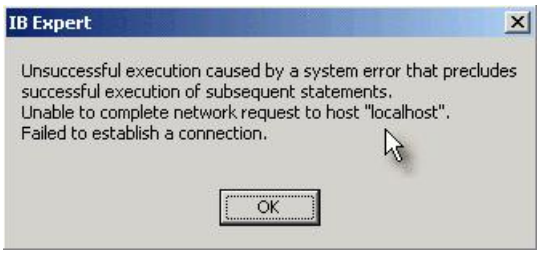
```
[DENY]
192.169.0.0-192.169.255.255
192.168.1.13

[ALLOW]
205.100.0.1-210.255.255.255
211.25.3.1
127.0.0.1

[ALARM]
127.0.0.1=net send ddking Hey! Somebody connected to me.
205.100.0.1-210.255.255.255=C:\my_dir\my_alarm_program.exe

[DBUSERS]
C:\path\db1.fdb=SYSDB1,DB1*3,
C:\IB\FB15\database\EMPLOYEE.GDB=*
```

DENY and ALLOW sections

Both sections contain IP addresses or IP address ranges. The client connection will be allowed if:

- the client's IP address is present in the ALLOW section (or the ALLOW section is empty)
- the client's IP address is absent in the DENY section (or the DENY section is empty)

For example, if you remove 127.0.0.1 from the ALLOW section, and then try to connect to our test database you should receive the following message:



The same result is achieved if you add 127.0.0.1 to both the DENY and ALLOW sections.

ALARM section

First make sure that 127.0.0.1 is valid, i.e. present in ALLOW and absent in DENY.

In the previous example shown in the DENY and ALLOW sections, the net send system command will be executed when SQL proxy receives a connection request from IP 127.0.0.1 (*ddking* is here the user or computer name where a message will be sent. You will need to replace it by your Windows user name, and *Hey! Somebody connected to me.* is just a message text).

Now, if you try to make a connection, you should receive the following message:

This means that the net send ... command line was executed by SQL proxy.

DBUSERS section

If you want to control Firebird/InterBase user access to certain databases, you may use this section.

In the previous example (ALARM section) you can see an example of the configuration of this section.

```
[DBUSERS]
C:\path\db1.fdb=SYSDB1,DB1*3,
C:\IB\FB15\database\EMPLOYEE.GDB=*
```

This means that users SYSDB1 and DB1*3 (where * is a wildcard) are allowed to connect to the database C:\path\db1.fdb and any users can connect to the C:\IB\FB15\database\EMPLOYEE.GDB database. User names in the user list may contain the * character and should be separated by commas.

Let's change the `C:\IB\FB15\database\EMPLOYEE.GDB` user list to see this feature working:

```
[DBUSERS]
C:\IB\FB15\database\EMPLOYEE.GDB=SYSDBA
```

Make sure that this user is configured in the `employee.gdb` *Registration Info* in IBExpert and then connect to this database. Everything should be ok – you're connected and can work.

Now close connection and change `DBUSERS` section this way:

```
DBUSERS]
C:\IB\FB15\database\EMPLOYEE.GDB=TEST
```

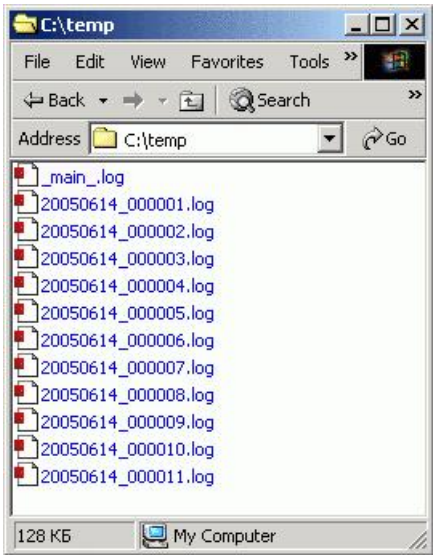
Then restart SQL proxy and try to connect to `employee.gdb`. You should receive this error message:

StatToHtml: logging to HTML and FTP

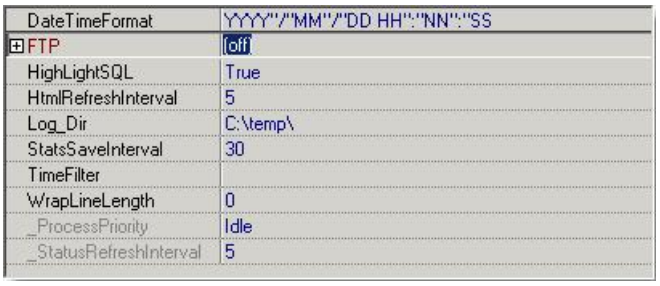
This module allows you to view the log files made by SQL proxy in HTML format. All you need is to start a service. This service can also upload all your logs and statistics to a selected FTP server, to let you view Firebird/InterBase server activity remotely, using just a simple web browser. To enable this feature you should set up the FTP server properties in the *StatToHtml* service configuration. We'd like to illustrate these features using an example. The open source project Filezilla (http://filezilla.sourceforge.net/) is used as a test FTP server in the following section, *Log to HTML transform*.

Log to HTML transform

Following the SQL proxy testing described earlier, we now have a few log files in our log directory:

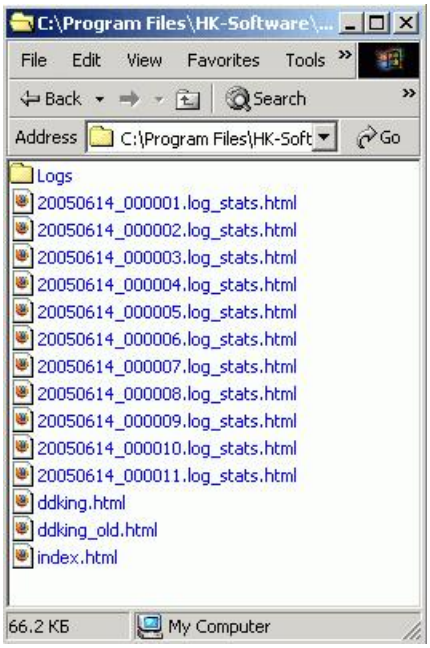


Now we are going to see *StatToHtml* working with these files. On the screenshot below you can view the default *StatToHtml* properties:

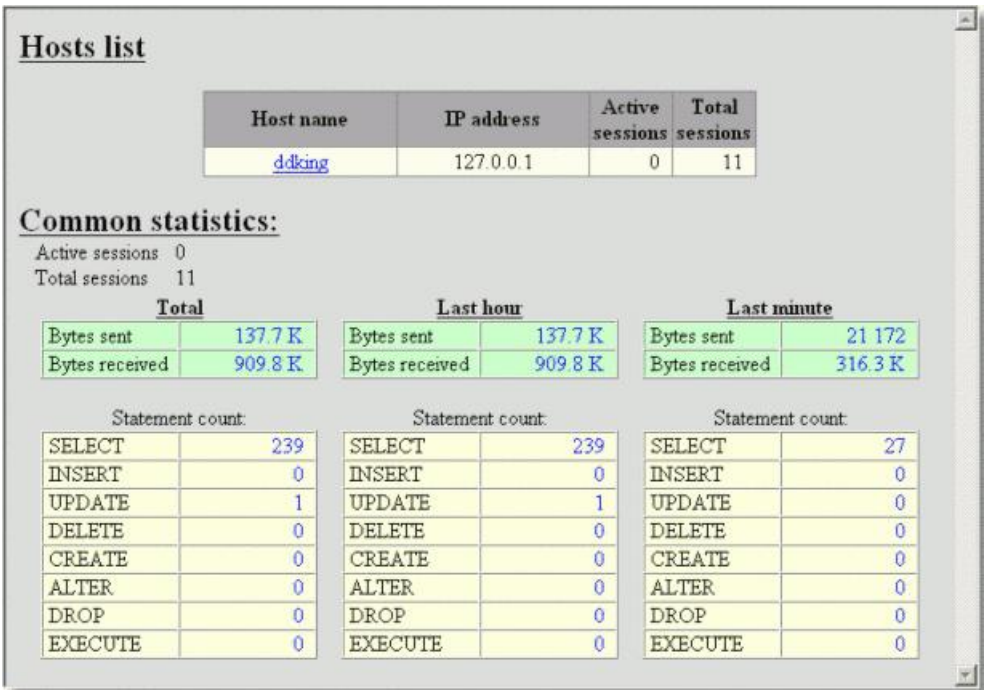


DateTimeFormat	YYYY"/"MM"/"DD HH":"NN":"SS
⊞ FTP	(off)
HighLightSQL	True
HtmlRefreshInterval	5
Log_Dir	C:\temp\
StatsSaveInterval	30
TimeFilter	
WrapLineLength	0
_ProcessPriority	Idle
_StatusRefreshInterval	5

HTML log file production occurs by a timer when the service is working, and once again when service stops. To view the HTML files produced just start the *StatToHtml* service by clicking on the corresponding button in the SCC and then stop the service. You can see that an HTML folder has been created in the IBExpertSQLMonitor installation directory and there are a few HTML files (see screenshot below) corresponding to the SQL proxy log files.



To navigate the log files simply open the `index.html` file. You should see something like this:



Hosts list

Host name	IP address	Active sessions	Total sessions
ddking	127.0.0.1	0	11

Common statistics:

Active sessions 0
Total sessions 11

Total		Last hour		Last minute	
Bytes sent	137.7 K	Bytes sent	137.7 K	Bytes sent	21 172
Bytes received	909.8 K	Bytes received	909.8 K	Bytes received	316.3 K

Statement count:		Statement count:		Statement count:	
SELECT	239	SELECT	239	SELECT	27
INSERT	0	INSERT	0	INSERT	0
UPDATE	1	UPDATE	1	UPDATE	0
DELETE	0	DELETE	0	DELETE	0
CREATE	0	CREATE	0	CREATE	0
ALTER	0	ALTER	0	ALTER	0
DROP	0	DROP	0	DROP	0
EXECUTE	0	EXECUTE	0	EXECUTE	0

This is a sample screenshot of the browser window after opening the `index.html` file. Here you can see a list of hosts being connected to the Firebird/ InterBase server and common traffic statistics:

- sessions count
- I/O volumes and
- statements count

If you wish to see the statistics and sessions of a separate host – just click on respective host name in the table at the top of `index.html`.

You should then see the following:

Last 5 sessions:

Started at	Finished at	Database name	Statistics	Logs
2005/06/14 20:05:53	2005/06/14 20:05:53	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log
2005/06/14 20:05:53	2005/06/14 20:07:11	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log
2005/06/14 20:05:51	2005/06/14 20:07:14	C:\IB\FB15\help\help.fdb	statistics	Log
2005/06/14 20:05:20	2005/06/14 20:05:20	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log
2005/06/14 20:05:20	2005/06/14 20:05:42	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log

Obsolete sessions

Host statistics:

Active sessions 0
Total sessions 11

	Total	Last hour	Last minute
Bytes sent	137.7 K	137.7 K	21 172
Bytes received	909 8 K	909 8 K	316.3 K

Statement count:	Total	Last hour	Last minute
SELECT	239	239	27
INSERT	0	0	0
UPDATE	1	1	0
DELETE	0	0	0
CREATE	0	0	0
ALTER	0	0	0
DROP	0	0	0
EXECUTE	0	0	0

On this screen you can view selected host traffic statistics and short descriptions of the last five client/server sessions produced by this host. Older sessions can be viewed by clicking on the *Obsolete sessions* link, below the last sessions table:

Obsolete sessions:

Started at	Finished at	Database name	Statistics	Logs
2005/06/14 20:04:52	2005/06/14 20:05:15	C:\IB\FB15\help\help.fdb	statistics	Log
2005/06/14 20:04:45	2005/06/14 20:04:45	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log
2005/06/14 20:04:45	2005/06/14 20:05:18	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log
2005/06/14 20:04:37	2005/06/14 20:04:51	C:\IB\FB15\help\help.fdb	statistics	Log
2005/06/14 20:02:53	2005/06/14 20:02:53	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log
2005/06/14 20:02:53	2005/06/14 20:03:21	C:\IB\FB15\database\EMPLOYEE.GDB	statistics	Log

Separate session statistics can be viewed by clicking the *Statistics* link on the respective session row in the session's table. There you can view the session start and end time, the session duration time and the session statistics.

View log

Session statistics:

Session started	2005/06/14 20:05:53
Session finished	2005/06/14 20:07:11
Session length	1 min 18 sec 31 msec

	Total	Last hour	Last minute
Bytes sent	44 092	44 092	32 832
Bytes received	479.7 K	479.7 K	380.1 K

Statement count:	Total	Last hour	Last minute
SELECT	66	66	41
INSERT	0	0	0

To view the selected session log click on the *Log* link in the session table of the host statistics window or click the *View log* link in the session statistics window.

Date/Time	Time diff	Execution time	
2005/06/14 20:05:53:468			Connect to database: C:\IB\FB15\database\EMPLOYEE.GDB; User
2005/06/14 20:05:53:468	15 msec	16 msec	SELECT D.RDB$CHARACTER_SET_NAME, C.RDB$DEFAULT_COLLATE_NAMI FROM RDB$DATABASE D LEFT JOIN RDB$CHARACTER_SETS C ON (D.RDB$CHARACTER_SET_NAMI
2005/06/14 20:05:53:468	15 msec		PLAN JOIN (D NATURAL,C INDEX (RDB$INDEX_19))
2005/06/14	16	16 msec	SELECT RDB$FIELD_NAME FROM RDB$RELATION_FIELDS

By default you will see all statements as they were logged, without any filtering or wrapping. Should you wish, for example, to view only time consuming SQL statements, simply set the *TimeFilter* property. For example, if you want to see only those statements in the HTML files whose execution time is more then 10 msec, you should set *TimeFilter = 10 msec*.

DateTimeFormat	YYYY"/"MM"/"DD HH":"NN":"SS
⊞FTP	(off)
HighLightSQL	True
HtmlRefreshInterval	5
Log_Dir	C:\temp\
StatsSaveInterval	30
TimeFilter	10 msec
WrapLineLength	0
_ProcessPriority	Idle
_StatusRefreshInterval	5

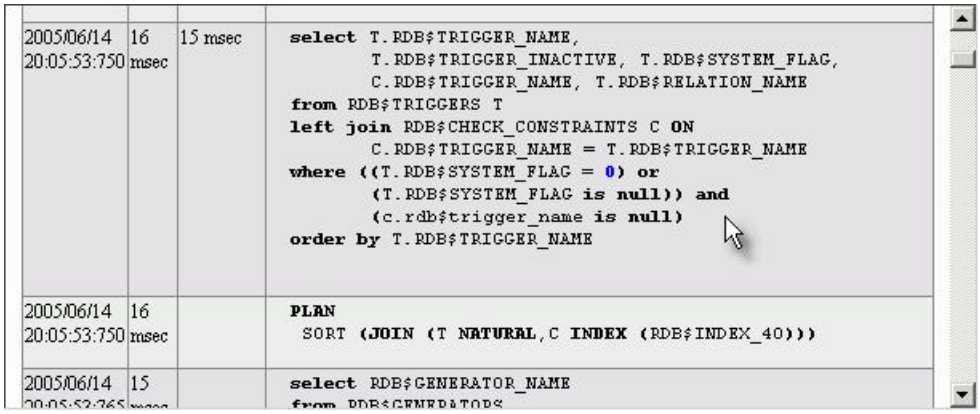
Save the new specifications by clicking the *Save* button in the SCC and then start and stop the *StatToHtml* service, to enable it to recreate the HTML log files. Now, when you open any session's log file, you will see only statements with an execution time >= 10 msec.

2005/06/14 20:05:53:750	16 msec	15 msec	select T.RDB$TRIGGER_NAME, T.RDB$TRIGGER_INACTIVE, T.RDB$S from RDB$TRIGGERS T left join RDB$CHECK_CONSTRAINTS C ON C.RDB$TRIGGER_NAME = T where ((T.RDB$SYSTEM_FLAG = 0) or (T.RDB$SYSTEM_FLAG is nul order by T.RDB$TRIGGER_NAME
2005/06/14 20:05:53:750	16 msec		PLAN SORT (JOIN (T NATURAL,C INDEX (RDB$INDEX_40)))
2005/06/14	16	16 msec	select RDB$INDEX_NAME, RDB$RELATION_NAME, RDB$SYSTEM_FLAG

In the screenshot above you can see that some statements are very long; these can be read using the horizontal scroller. If you want *StatToHtml* to make these statements easier to read, you can set up the statement wrapping by setting the *WrapLineLength* property. For example set it to 50:

DateTimeFormat	YYYY"/"MM"/"DD HH":"NN":"SS
⊞FTP	(off)
HighLightSQL	True
HtmlRefreshInterval	5
Log_Dir	C:\temp\
StatsSaveInterval	30
TimeFilter	
WrapLineLength	50
_ProcessPriority	Idle
_StatusRefreshInterval	5

Then save and start/stop the service. After opening any HTML log file you can see that now all statements are smartly wrapped and became much easier to read:



2005/06/14 20:05:53:750	16 msec	15 msec	select T.RDB$TRIGGER_NAME, T.RDB$TRIGGER_INACTIVE, T.RDB$SYSTEM_FLAG, C.RDB$TRIGGER_NAME, T.RDB$RELATION_NAME from RDB$TRIGGERS T left join RDB$CHECK_CONSTRAINTS C ON C.RDB$TRIGGER_NAME = T.RDB$TRIGGER_NAME where ((T.RDB$SYSTEM_FLAG = 0) or (T.RDB$SYSTEM_FLAG is null)) and (c.rdb$trigger_name is null) order by T.RDB$TRIGGER_NAME
2005/06/14 20:05:53:750	16 msec		PLAN SORT (JOIN (T NATURAL,C INDEX (RDB$INDEX_40)))
2005/06/14 20:05:53:765	15 msec		select RDB$GENERATOR_NAME from RDB$GENERATORS

StatToHtml: HTML-related properties

Now let's review HTML related properties of the *StatToHtml* service:

Section or parameter	Description
`DateTimeFormat`	Format of timestamps in HTML files `"YYYY"/"MM"/"DD HH":"NN":"SS"` `YYYY` - year `MM` - month `DD` - day `HH` - hour `NN` - minutes `SS` - seconds Any characters in double quotes are constants.
`HighLightSQL`	If True then SQL statements in HTML files will be highlighted for more readability. Default is True.
`HtmlRefreshInterval`	Time (in seconds) to be used as a refresh meta tag value in HTML files while the *StatToHtml* service is working. When the service is stopped, it will rewrite all HTML log files without the refresh meta tag. Default is 5.
`Log_Dir`	Path to the folder where SQL proxy log files are placed. Default is `C:\temp\`.
`StatsSaveInterval`	Time interval (in seconds) defining HTML files production periodicity. Default is 30.
`TimeFilter`	If you want to see only time-consuming statements iin HTML form you can use this property. Here you may set the statement execution time filter to make *StatToHtml* remove all statements with an execution time less than the *TimeFilter* value from the HTML files. The format of this filter is the same as the format of the Execution time column of the HTML log files: `… Days … hours … min … sec … msec` where each "…" is some integer value Here are filter value examples: - 1 min - 30 sec 10 msec - 1 hours 30 min 20 sec 10 msec - 5 Days Default is empty.
`WrapLineLength`	If you want to wrap long SQL statements to make them more easily readable, you may set this property. Default is 0 – i.e. no wrapping.
`_StatusRefreshInterval`	Time interval (in seconds) of runtime info refresh. Default is 5.
`_ProcessPriority`	*StatToHtml* process priority (*Idle*, *Normal*). Default is *Idle*.

FTP upload

StatToHtml enables you to load HTML log files onto selected FTP server so that you can view them remotely. To enable this function, you should set up the *FTP properties* of the *StatToHtml* service. By default this FTP functionality is disabled:

⊟FTP	(off)
FTPUpload	Disabled
Host	
⊟LoginParams	user:(empty), password:(empty)
Password	
Username	
Port	21
TmpDir	C:\temp\FtpTmp

To activate it, set up the *Host property* to your FTP server address, also specify the *FTP user login information*.

Then switch the *FTPUpload* property to *Enabled*. We will now illustrate this feature on localhost (using the Filezilla FTP server) with two users created for our test:

- **ftp_user** (without password)
- **anonymous**

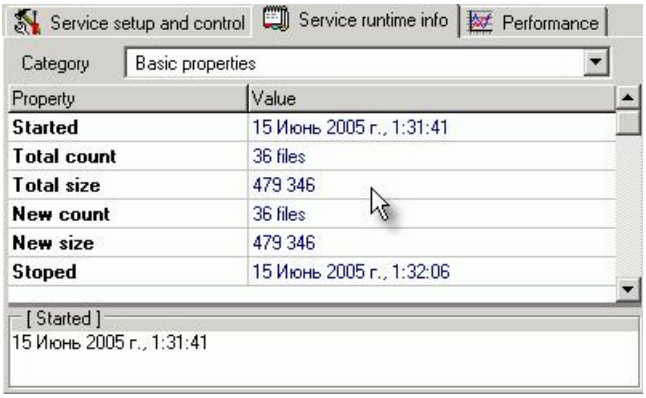
Both should have the same home directory (`C:\local_FTP_home` in this example).

The `TmpDir` folder is used by the FTP upload algorithm to store uploaded files and then detect any newly created files which need to be uploaded.

Configure *StatToHtml* to work with FTP on localhost, using the user `ftp_user`. On the screenshot below you can see the corresponding configuration:

Now save the configuration and start/stop the service. In the Filezilla server window you should see a lot of client activity, made by the *StatToHtml* service:

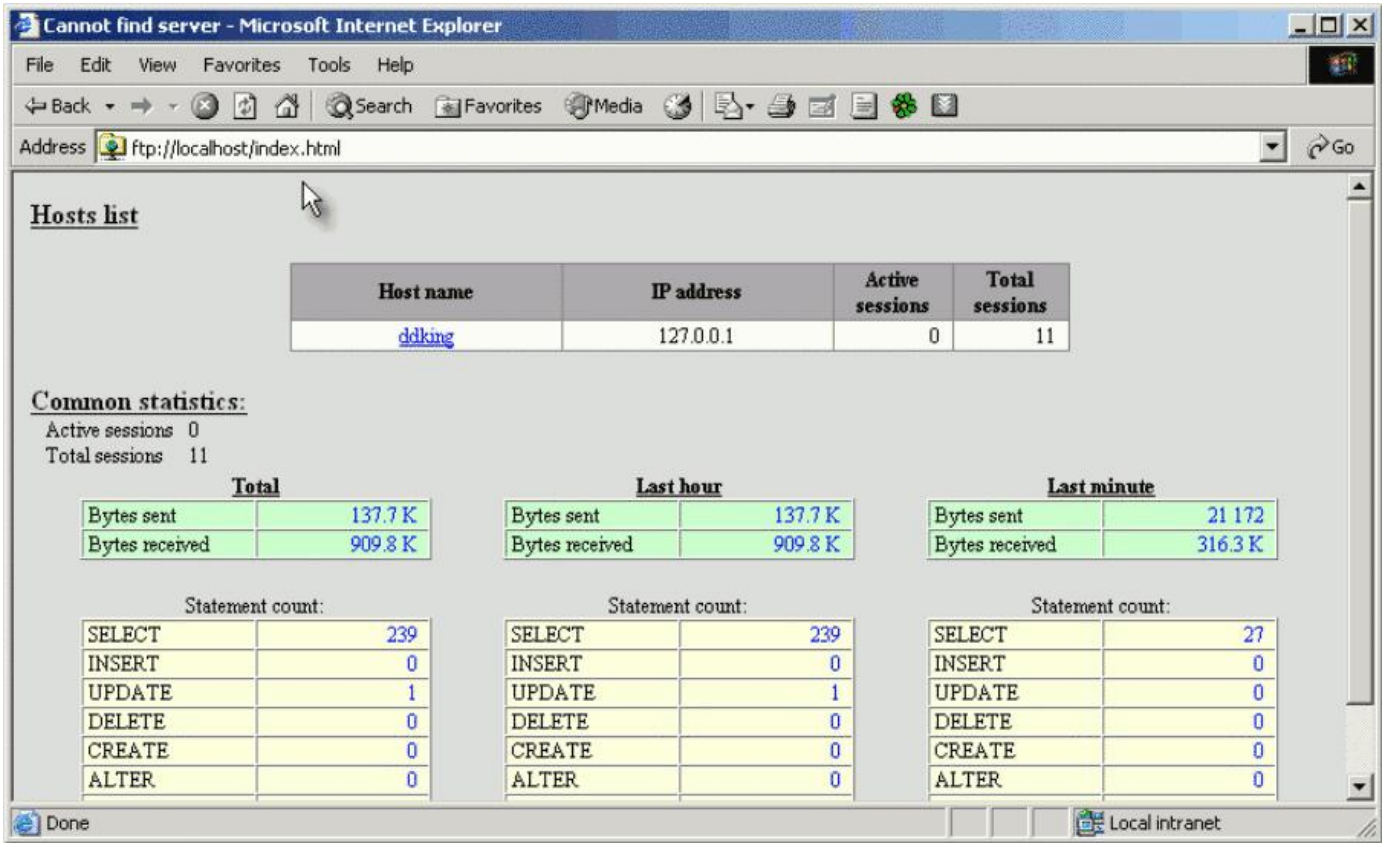
You can also view the *StatToHtml* FTP activity report on the *Service runtime info* page in the SCC window:



Here you can see the total count and size of files uploaded by the service. Also you can see the count and size of newly uploaded files, i.e. files changed after the last upload session. The same statistics can be viewed in the charts on the *Performance* page.

Now let's try to open our HTML material through FTP by opening a link in the browser `ftp://localhost/index.html`.

Here is the screenshot you should see:

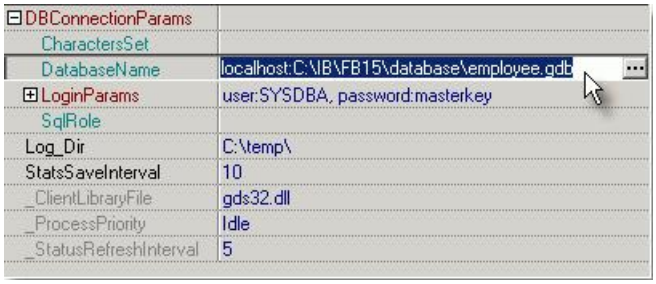


StatToHtml has uploaded HTML log files onto the FTP server and you can now view them remotely.

StatToDB

This module is needed if you want to store your log files in a Firebird/InterBase database and analyze them using SQL. StatToDB, like StatToHtml described earlier, takes log files made by SQL Proxy and puts them, as they are, in a specified database. All database objects (two tables: IBE$MONITOR_SESSIONS and IBE$MONITOR_EVENTS, generators, triggers and indices) needed to store the log files are created by the service itself if necessary.

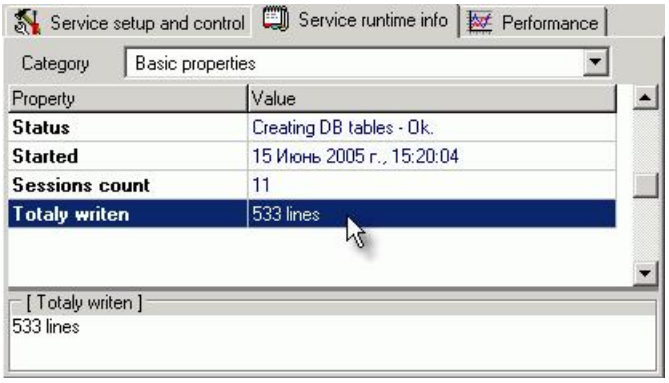
Let's see the service working. Before starting it you should specify the database where the service is to store the log files. Actually it may be the same database (`employee.gdb`) used for the previous tests:



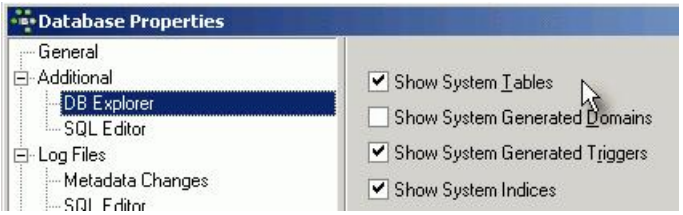
Now start the service, and select the *Service runtime info* page in the SCC to view service activity. If this database has not previously been used by the service, it will first create the necessary database objects. You should see a corresponding report line in the runtime properties table:

Status Creating DB tables - Ok.

Then, after the *StatsSaveInterval* time period has elapsed (default – 10 sec), you will see the log files uploading report information to the database:



Now let's see what has been done in the selected database by the *StatToDB* service. Start IBExpert and register a database connection to `employee.gdb`. In the Database Properties window set IBExpert to *show system objects in the DB Explorer*:



Then connect to the database.

You should see two new tables under *System Tables* node:

IBE$MONITOR_SESSIONS

This table contains information about all sessions logged by SQL proxy. Here is the table structure:

#	FK	PK	Field Name	Field Type	Domain	Size	S
1			SESSION_ID	CHAR		32	
2			DATABASE_NAME	VARCHAR		512	
3			USER_NAME	VARCHAR		31	
4			IP	VARCHAR		15	
5			PORT	INTEGER			
6			LOG_FILENAME	VARCHAR		20	
7			LAST_SAVED_LINE	INTEGER			

Field name	Description
SESSON_ID	Session identifier. Should be used for joins with IBE$MONITOR_EVENTS table.
DATABASE_NAME	Name of database, used in the session.
USER_NAME	Name of user, connected to the database in the session.
IP	Client's IP address.
PORT	Client's port.
LOG_FILENAME	Name of session's log file.
LAST_SAVED_LINE	Used internally by the *StatToDB* service.

Now, for example, if you want to check if a host has been working with your Firebird/InterBase server, you should execute the respective SQL query on this table, for example:

```
SELECT * FROM IBE$MONITOR_SESSIONS WHERE IP = '11.22.33.44'
```

This enables you to view a list of all sessions made by the specified host.

If you want to analyze client activity by statements or statement plans – you should query the IBE$MONITOR_EVENTS table.

IBE$MONITOR_EVENTS

This table contains lines from all log files for all sessions. Here is the table structure:

#	FK	PK	Field Name	Field Type	Domain	Size	Sc
1			EVENT_ID	INTEGER			
2			SESSION_ID	CHAR		32	
3			TME	VARCHAR		25	
4			TXT	BLOB		100	

Field name	Description
EVENT_ID	Just an identifier.
SESSION_ID	Session identifier. Should be used for joins with the IBE$MONITOR_SESSIONS table to get log lines for separate sessions.
TME	Log line appearance time. Timestamp string.
TXT	Event text. May be one of the following: . connect/disconnect message . SQL statement . PLAN statement.

For example, if you want see all SELECT statements, which are not related to system tables, you may use, for example, such a query:

```
select * from ibe$monitor events
where
      TXT containing 'SELECT'
      and TXT not containing 'RDB$'
```

Then, if you are working with logs made during SQL proxy testing, you should get a list of statements, which you've executed from IBExpert earlier:

EVENT_ID	SESSION_ID	TME	TXT
434	a44259261fd72ea94a	2005/06/14 20:06:02:031	select * from country/*CRLF*/
436	a44259261fd72ea94a	2005/06/14 20:06:04:562	select * from employee/*CRLF*/
459	a44259261fd72ea94a	2005/06/14 20:06:16:953	select * from EMPLOYEE/*CRLF*/order by EMP_NO/*CRLF*/
485	a44259261fd72ea94a	2005/06/14 20:06:37:515	select * from ORG_CHART/*CRLF*/
494	a44259261fd72ea94a	2005/06/14 20:06:41:765	SELECT * FROM ORG_CHART/*CRLF*/
518	a44259261fd72ea94a	2005/06/14 20:06:58:062	select * from SALARY_HISTORY/*CRLF*/order by

/*CRLF*/ in SQL statements is used to replace character returns, so you may restore the source SQL statement view if needed.

For example, you may use the following query to view all statement plans:

```
select * from ibe$monitor_events
where
      TXT starting with 'PLAN'
```

Or all client connect/disconnect messages:

```
select * from ibe$monitor_events
where
      TXT starting with 'Connect'
      or
      TXT starting with 'Disconnect'
```

Or view a list of active connections (no disconnect message in log) by joining both tables in such a query:

```
select * from ibe$monitor_sessions s
              left join ibe$monitor_events e
              on
                (e.session_id=s.session_id
                  and
                 e.txt starting with 'Disconnect')
where e.event_id is null
```

Actually, using SQL you can perform extremely complex log file analysis by simply querying the tables made by the *StatToDB* service.

IBExpertSQLMonitor Help

The complete IBExpertSQLMonitor help files (beta version) are available directly online: http://ibexpert.net/ibe/pmwiki.php?n=Doc.IBExpertSQLMonitor/.

The first view displays the documentation structure. If you are looking for help about a specific subject use the Search function.

Should you not be able to find a solution to your problem here, please use the IBExpertSQLMonitor newsgroup: news://ibexpert.info/ibmonitor.general.en or send us an email to support@ibexpert.com.

Should you have any comments or queries directly regarding the documentation, or wish to contribute you own articles, please contact documentation@ibexpert.com.

What's New?

IBExpertSQLMonitor v. 2004.10.03.1

1. Session Info:

- Simply double-click any active connection in the Control Center.

2. Security Setting in the SQL Proxy:

- IP address blocking after numerous bad password connection attempts.
- List of blocked IPs with blocking time can be seen in hkSCC, on the service *Runtime info page* (category - "Blocked IP list"). During blocking time the client will be banned any connection attempt (even with a valid user name and password). "Bad Guys" are blocked before any client/server packet exchange. So no IB server activity will be produced by such a fugitive client. He will simply receive a message such as the following (taken from IBExpert connection test):

```
"Unable to complete network request to host " ......".
Failed to establish a connection.
Unknown Win32 error 10060.
```

3. New installer, bugfixes and small improvements...

IBExpertSQLMonitor v. 2004.04.18.1

Bugfixes and small improvements...

IBExpertSQLMonitor v. 2004.04.12.1

New Features:

1. For use with InterBase: IP 127.0.0.2 is usable:

Typical config for using IBExpertSQLMonitor with InterBase:

```
bind_ip=127.0.0.2
bind_port=3050
map_ip=127.0.0.1
map_port=3050
```

Typical connection string in an InterBase environment:

```
127.0.0.2:C:\path\db.ib
```

IBExpertSQLMonitor should work with all InterBase versions and all Firebird versions.

2. Installer now comes with necessary dll file

3. Bugfixes and small improvements...

(:keywords :)

IBExpert Documentation

Click here to view the complete IBExpert documentation: http://ibexpert.net/ibe/pmwiki.php?n=Doc.IBExpert.

FAQs

FAQs

Here we will attempt to answer some of the more frequently asked questions regarding IBExpertSQLMonitor. Should you not be able to find a solution to your problem here or elsewhere within the IBExpertSQLMonitor Documentation, please contact our newsgroup: news://ibexpert.info/IBMonitor.general.en (English language) or send an email to support@ibexpert.com.

How much load does IBExpertSQLMonitor add to the server?

I would like to run IBExpertSQLMonitor on a heavily loaded live server with up to over one hundred concurrent users.

A: *So far even customers with extremely large installations have reported almost no loss in performance at all.*

What figures does the `Time Diff` column show in the `log.html`?

The meaning of the figures displayed in the `Time Diff` column is unfortunately not clear to me.

A: *This displays the difference from one statement to the next statement, because in some cases (for example, with a `FETCH ALL`), you do not see the correct time with the first calling statement.*

... coming soon.

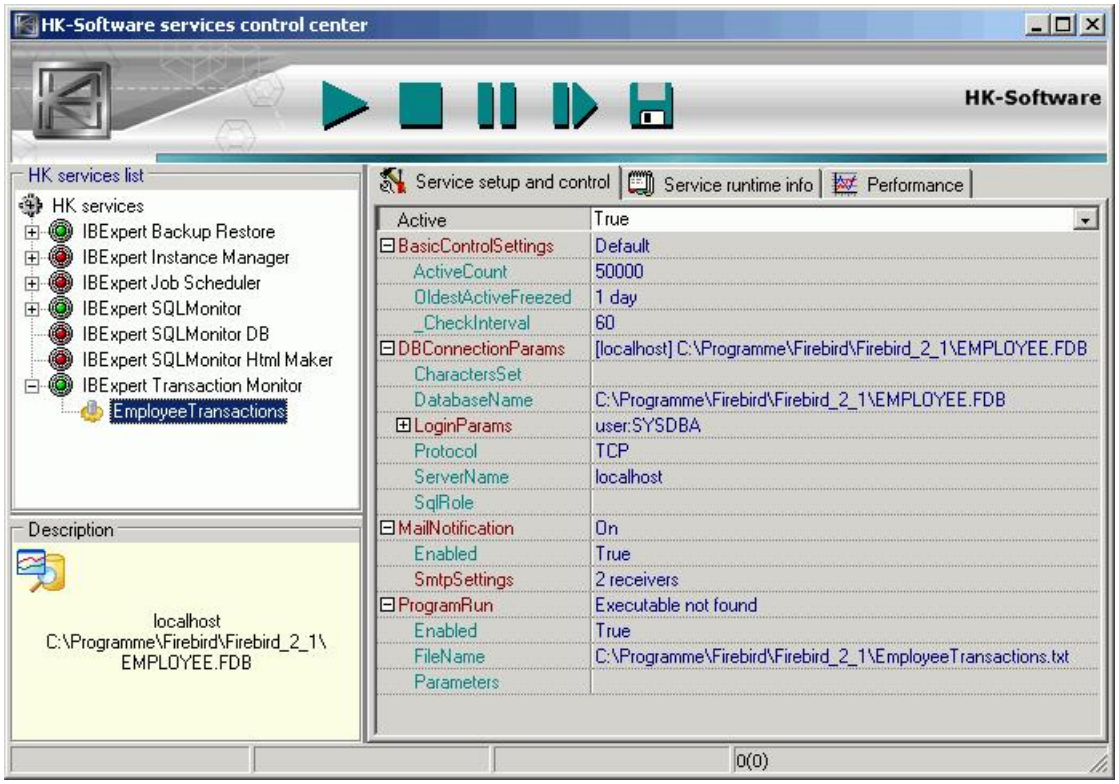
- What is IBExpertTransactionMonitor?
- Setup and usage

IBExpertTransactionMonitor

... currently in work.

IBExpertTransactionMonitor is a new module in the HK-Software Control Center.

The IBExpert Junior VAR license or the VAR license entitles you to distribute the IBExpertTransactionMonitor with your application.



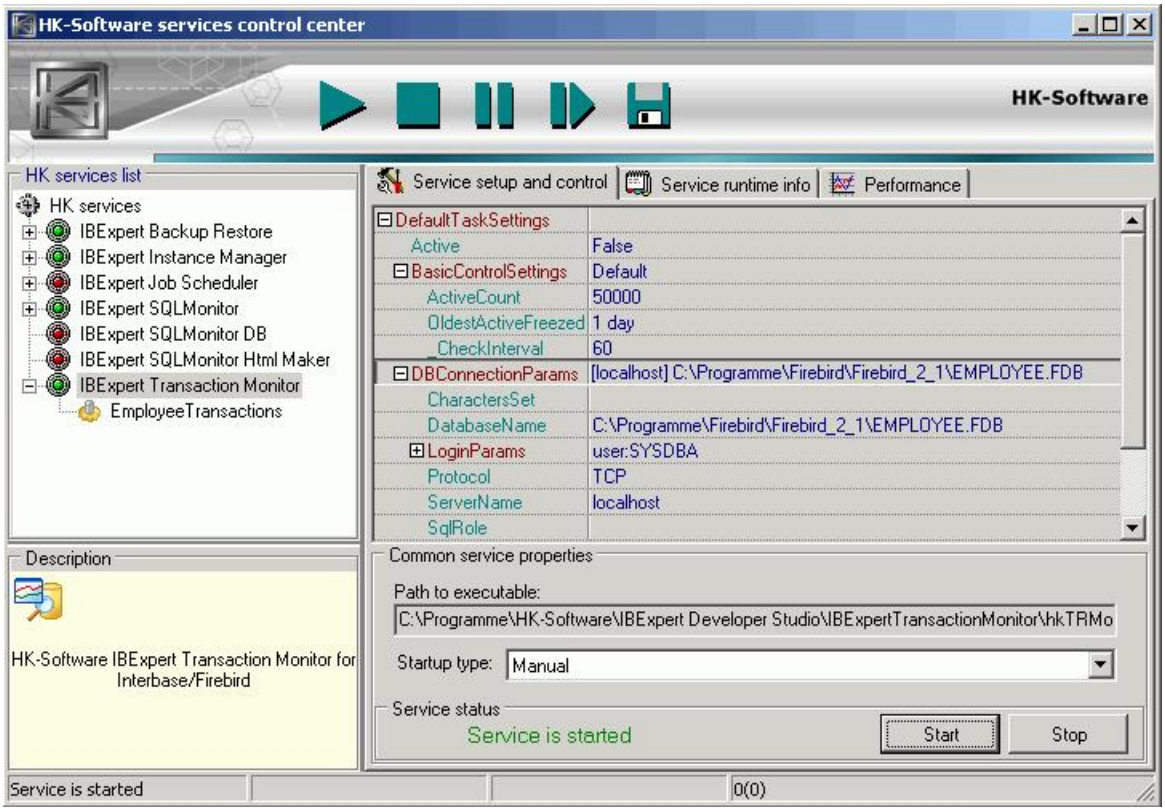
...currently in work.

Setup and usage

Start the HK-Software Services Control Center, found in the IBExpert Services menu, and select *IBExpert Transaction Monitor* in the *HK services list*.

We now need to configure the default task settings. As some parameters will remain the same for all further tasks (for example: SMTP settings), these should be configured first.

Expand the *DefaultTaskSettings* item on the *Service setup and control* page.



The following lists the various default settings and options available:

- Active
- Basic control settings
- Database connection configuration
- Mail notification
- Schedule
- Program run

After configuring the default task settings, all new tasks will have this configuration when created. It is of course possible to alter specific options for individual tasks.

Default task settings

Active

When `True` then the task just created will be active (see illustration above).

Basic control settings

Parameters include: *ActiveCount*, *OldestActiveFreezed* and *_CheckInterval*.

⊟ BasicControlSettings	Default
ActiveCount	50000
OldestActiveFreezed	1 day
_CheckInterval	60

Database connection configuration

The next step is to establish the database connection. All necessary properties can be configured in the *DBConnectionParams* section:

⊟ DBConnectionParams	[localhost] C:\Programme\Firebird\Firebird_2_1\EMPLOYEE.FDB
CharactersSet	
DatabaseName	C:\Programme\Firebird\Firebird_2_1\EMPLOYEE.FDB
⊟ LoginParams	user:SYSDBA
UserName	SYSDBA
UserPassword	********
Protocol	TCP
ServerName	localhost
SqlRole	
Log	True

This is fairly self-explanatory; although should you require detailed information regarding Firebird/InterBase database connection parameters, please refer to the online IBExpert documentation.

Mail notification

The mail notification feature sends reports concerning the IBExpertTransactionMonitor activity. The service sends an e-mail message with log files attached when the job is completed.

To use this feature, set the *Enabled* parameter in the *MailNotification* section to *True*.

⊟ MailNotification	On
Enabled	True
SmtpSettings	2 receivers

The IBExpertTransactionMonitor uses a built-in SMTP client to send e-mails, so you need to set up the SMTP parameters in the task configuration to enable this to work properly. Simply double-click on the *SmtpSettings* option, to open the configuration dialog window.

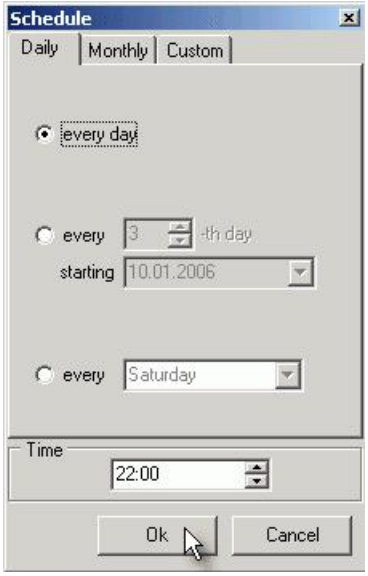
In this dialog you should set up the *Sender*, *SMTP server configuration* and one or more recipients.

Schedule

Schedule	Every day at 22:00

Double-click on the *Schedule* option to open the schedule configuration dialog window:

Daily schedule:

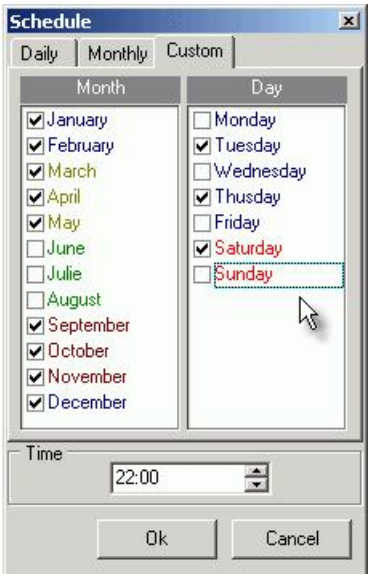


- every day at the specified time.
- every *n*th day, starting from date.
- every given day of week.

Monthly schedule:

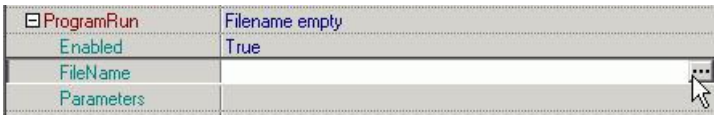
Every *n*th day of the selected months at the given time.

Custom schedule:



Selected days of every week of selected months at given time.

Program run



Here you can activate the *ProgramRun* by altering the *Enable* parameter to *True*. Then simply specify the file name and add parameters if required.

_ProcessPriority

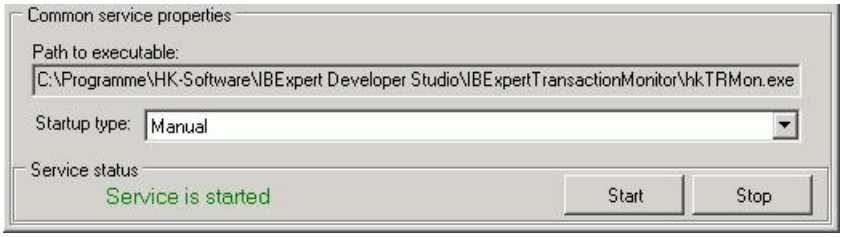
This parameter can be set to *Idle*, *Normal* or *High* (the default is *Idle*).

_StatusRefreshInterval

Here the refresh interval in seconds can be specified (default value is 5).

Common service properties

The path to the executable file, `hkTRMon.exe` is displayed. You can specify the *Startup type* selecting an option from the drop-down list (options: *Manual*, *Automatic* or *Disabled*).



The *Service Status* can be viewed at the bottom of the window, and the *Start* and *Stop* buttons used to manually start or stop the service.

When you are happy with your specifications, they can be saved using the disk icon in the toolbar. After configuring the default task settings, all new tasks will have the same configuration when created. You can of course alter specific options for individual tasks if wished.

Preparing a task

To create individual job schedules, you now need to create a task. Right-click on the *IBExpert Transaction Monitor* service's item in the SCC. Then click *Add task* in the popup menu. After that you will see the new task item (Task 0) under the *Transaction Monitor* service's item. You may rename it by clicking on the name simultaneously holding the [Ctrl] key down.

Alter your default settings if necessary. Then you can simply run the service.

IBExpertDemoDB

Use the IBExpertDemoDB for benchmark testing. The UDFs and SQLs necessary to generate the demo database can be found in the IBExpert Developer Studio's `/IBExpertDemoDB` directory. This documentation lists the simple steps needed to generate a demo database to the size of your choice.

The IBExpert Benchmarks article illustrates in detail how to utilize this valuable function as a sample web shop.

Database Technology Articles

This section offers a more in-depth view of the InterBase/Firebird database and how it functions.

- Firebird Classic server versus SuperServer
- Database design and database normalization
- Enterprise-wide data model
- Space management in InterBase
- Multi-generational architecture and record versioning
- Multi-version concurrency control
- Using IBExpert and Delphi applications in a Linux environment, accessing Firebird
- Bidirectional replication for InterBase and Firebird
- Database Corruption
- Firebird for the Database Expert: Episode 1 - Indexes
- Firebird for the Database Expert: Episode 2 - Page Types
- Firebird for the Database Expert: Episode 3 - On Disk Consistency
- Firebird for the Database Expert: Episode 4 - OAT, OIT and Sweep
- Firebird for the Database Expert: Episode 5 - Locking and Record Versions
- Firebird for the Database Expert - Episode 6: Why can't I shrink my databases
- Structure of a header page
- Structure of a data page
- Garbage Collectors
- Record versions as an undo log
- Where do data pages come from?
- Optimize database cache utilization to improve database performance
- Selecting the right datatype to improve database performance

Classic server versus SuperServer

Many thanks to Paul Beach of http://www.IBPhoenix.com for this article.

InterBase SuperServer architecture

SuperServer is a multi-client, multi-threaded implementation of the InterBase server process. This implementation replaces the "Classic" implementation used for previous versions of InterBase.

SuperServer serves many clients at the same time using threads instead of separate server processes for each client. Multiple threads share access to a single server process. The benefits of SuperServer architecture include:

Having a single server process eliminates bottlenecks resulting from arbitration for shared database pages and reduces the overhead required for multiple process startups and database queries. SuperServer improves message interaction performance because a shared library call is always faster than an interprocess communication request to a server process.

SuperServer improves database integrity because only one server process has write access to the database, rather than one process for each client. All database engine functionality is encapsulated into a unified, protected subsystem that is isolated from user application error.

SuperServer allows for the collection of database statistics and user information that InterBase's tools can use for performance monitoring and administrative tasks.

SuperServer is more cost-effective than the Classic architecture. All operating systems have limits on the number of OS processes that can run concurrently. SuperServer allows for a fixed number of database threads to be multiplexed over a potentially large number of concurrent database connections. Since these threads are not hard-wired to any specific database connection, SuperServer can support a larger number of users with minimum resources use.

InterBase Classic architecture

Classic architecture, the design in InterBase 4.0 and earlier, was process-based. For every client connection, a separate server process was started to execute the database engine, and each server process had a dedicated database cache. The server processes contended for access to the database, so a *Lock Manager* subsystem was required to arbitrate and synchronize concurrent page access among the processes.

Invoking the Classic Server

The InterBase Classic server runs on demand as multiple processes. When a client attempts to connect to an InterBase database, one instance of the `gds_inet_server` executable runs and remains dedicated to that client connection for the duration of the connection.

The initiator of `gds_inet_server` is `inetd`, the UNIX service turnkey process. It has a configuration file, `/etc/inetd.conf`, which associates services with the executable that is to receive the connection. When `inetd` receives a connection request for a given service, it looks up the appropriate program in `/etc/inetd.conf`, executes it, and transfers the network connection to the service program.

When the client chooses to disconnect, `gds_inet_server` closes its connection to the database and any other files, and then exits. When there are no clients connected to any database, there should be no invocations of `gds_inet_server` running.

Lock management

Lock management is taken care of by another process, `gds_lock_mgr`. This program is started when the second client attaches to a given database. The job of the lock manager is to serve (metaphorically) as a traffic cop. It grants locks on database resources to clients. It also requests that clients relinquish locks on a resource when that resource is in demand by other clients. The `gds_lock_mgr` remains running even after the last client disconnects. The next time a client connects, it can avoid the slight overhead of starting the lock manager process. For further information regarding locking, refer to *Firebird for the database expert: Episode 5 - Locking and Record Versions*.

Use of Posix signals

The `gds_lock_mgr` process communicates with each client process by using a shared memory area, and a signaling mechanism using the POSIX signals `SIGUSR1` and `SIGUSR2`. Signals are caught in signal handling routines in `libgdslib.a`, and for this reason user applications should not perform signal handling or any modification to the signal mask. Applications which need to use POSIX signals must compile with an alternate InterBase library, `libgds.a`. This library functions identically to `libgdslib.a`, but it handles signals sent by the lock manager in a child process called `gds_pipe`. All client applications compiled with `libgds.a` automatically run with this child process. No changes to application code are needed, only a different linking option.

Resource use

Each instance of `gds_inet_server` keeps a cache of database pages in its memory space, which is likely to result in some duplication of cached data across the system. While the resource use per client is greater than in SuperServer, Classic uses less overall resources when the number of concurrent connections is low.

Local access method

The Classic architecture permits application processes to perform I/O on database files directly, whereas the SuperServer architecture requires applications to request the IBServer I/O operations by proxy, using a network method. The local access method is faster than the network access method, but is only usable by applications which run on the same host as the database.

Monitoring

The database information call for active connections always reports exactly one connection on a Classic server, no matter how many clients are connected to databases on that server. The reason for this is that every client connection has its own `gds_inet_server` process on the server, and each instance of that program knows only about its own connection. Only in SuperServer does the server process have the ability to report all client connections on the server.

Security

In order for InterBase Classic to work with a mixture of local and remote clients running as different user ID's, the server executables `gds_inet_server` and `gds_lock_mgr` must run as root.

The processes must run with a real `uid` of root to set their effective `uid` to that of the client `uid`. The lock manager must have the superuser privilege to send signals to the processes. In some IT environments, the presence of executables with `setuid` bits turned on raises concerns about security. Nevertheless, do not change the runtime configuration of the InterBase server. The `setuid` root configuration of the Classic software is important to its function.

Because applications can run as any `uid`, database files must be writable by all `uid`s that access the databases. To simplify maintenance, database files are created writable by the whole world.

With care, you can restrict these file permissions, so that the database files are safe from accidental or deliberate damage. Make sure you understand file permissions completely before attempting this, because all local and remote clients need write access to the database, even if they intend only to read data.

Classic versus SuperServer

Invoking SuperServer

SuperServer runs as a single process, an invocation of the `ibserver` executable. `ibserver` is started once by the system administrator or by a system boot script. This process runs always, waiting for connection requests. Even when no client is connected to a database on the server, `ibserver` continues to run quietly.

The SuperServer process is not dependant on `inetd`; it waits for connection requests to the `gds_db` service itself.

The SuperServer process is a multi-threaded application. Different threads within the process are dedicated to different tasks. For instance, one thread waits on the `gds_db` service port for incoming connection requests. Other threads are analogous to individual `gds_inet_server` processes in the Classic model, serving client queries. Another thread serves as the lock manager, replacing the `gds_lock_mgr` process from the Classic model.

Lock management

The lock manager in SuperServer is implemented as a thread in the `ibserver` executable. Therefore InterBase does not use the `gds_lock_mgr` process. Likewise, POSIX signals are not used by the lock manager thread in SuperServer; interthread communication mechanisms are used.

Resource use

The SuperServer implementation has less overhead and uses fewer system resources per client connection than the Classic model. SuperServer has one cache space for all client attachments, allowing more efficient use of cache memory. For these and other reasons, SuperServer has demonstrated an ability to efficiently serve a higher number of concurrent clients.

Threaded server & UDFs

User-Defined Functions (UDFs) are libraries of functions that you can add to extend the set of functions that the InterBase server supports. The functions in your UDF library execute within the process context of the InterBase server. Due to the threaded implementation of SuperServer, there are issues with UDFs that require that you write UDF functions more carefully than when writing UDFs for a Classic server.

You must design UDFs for SuperServer as thread-safe functions. You cannot use global variables in your UDF library, because if two clients run the UDF simultaneously, they conflict in their use of the global variables.

Do not use thread-local global variables to simulate global variables. SuperServer implements a sort of thread pooling mechanism, to share threads among all the client connections. It is likely that if a given client executes a UDF twice, that each execution is not executed in the context of the same thread. Therefore, you cannot depend on thread-local variables keeping values from one execution of the UDF to the next for a given client.

UDFs that allocate memory dynamically run the risk of creating a memory leak. Because SuperServer is supposed to stay up and running indefinitely, not just for the duration of the client connection, memory leaks can be more damaging in SuperServer than in Classic. If your UDFs return dynamically allocated objects, then you must use `malloc()` to allocate the memory for these objects (on Win32, you must use `ib_util_malloc()` or the `malloc()` that is part of the Microsoft Visual C++ runtime library). Do not use new or `globalalloc()` or the Borland `malloc()`.

Finally, such functions must be declared in databases with the `FREE_IT` option of the `DECLARE EXTERNAL FUNCTION` statement.

By contrast, in Classic, there is a separate process for each client connection, so the UDFs are guaranteed not to conflict. Global variables are safe to use. Also, memory leaks are not as dangerous, because any leaked memory is released when the client disconnects. InterBase recommends that you design UDFs for SuperServer, the more restrictive model, even if you use a version of InterBase implemented with the Classic model. Eventually InterBase will be implemented with SuperServer on the platform you use. If you design UDFs with this assumption, you can upgrade to a later version of InterBase without the risk that your UDFs must be redesigned to work with SuperServer.

Security

SuperServer can be configured to run as a non-root `uid`, for enhanced security. In SuperServer, you can restrict the permissions on database files to allow only the InterBase server `uid` to access the database.

Why two implementations?

The Classic implementation predates the SuperServer implementation, and the SuperServer implementation is the future of InterBase. Classic configuration is used on operating systems that currently don't have the technology for threaded applications, which is required for SuperServer. InterBase also distributes the Classic version on platforms that have threading technology, but which benefit from the low-profile implementation.

SuperServer has a greater ability to meet the demands of a growing multi-user system, while retaining good performance and efficiency. SuperServer is implemented in InterBase product on all platforms where it is technically practical. It is the intention that SuperServer is the future direction of InterBase on all platforms.

Changing server to solve undefined crashes

September 2004. Many thanks to Gerhard Behnke at dpa (Deutsche Presse Agentur) for this contribution.

We managed to solve our problem with undefined Firebird crashes in the following way:

W2003/Superserver

It is essential to check Firebird's memory requirements using the *Task Manager*. If the requirements are approaching 2 GB, there is a danger of Firebird crashing, e.g. if more than 2 GB is required when submitting a long and detailed query.

Solution

1. Equip your server with at least 3 GB, and ensure the 3GB switch is set in the `Boot.ini`. In order to handle this 3 GB address space, it is necessary to use the appropriate Firebird version (when the normal Firebird version is only linked with a different link flag). I think we may be the only company to currently be in possession of such a Firebird version (Paul Reeves performed the linking for us).
2. The best solution is however to change to the Firebird Classic Server, together with sufficient RAM and more that one CPU. This certainly puts life back into the database!

Database design

A good database design is vital for a client/server application. It is important to think about the design of the tables among each other to optimize data storage, i.e. in which table should each quantity of information be placed, and how this table should be linked to the information in other tables. The normalization process helps here as it avoids double data storage as well as unnecessary wastage of space; data access becomes considerably more efficient, at the same time improving database performance and data integrity. Special business problems in the database can be solved with the aid of database design; for example, they enable typical relationships between master and detail tables.

Relational databases work best when data is broken up into different tables that are joined together on common columns. This design results in narrower, longer tables, where the primary key is used to access the data, and indices are used to speed this process.

Database models are generally designed to solve specific business problems: they allow typical business data relationships to be represented. This is particularly important, for example, when many detail rows need to be joined to one master row. This is most often done by splitting the data into two or more tables and joining them on a shared column. When data is represented in this way, some duplication is unavoidable. There are always columns that must appear in each table in order to actually create the join. However database models allow you to minimize unnecessary duplication.

These models also ensure that if a value is updated in one table, the matching values are updated in related tables, known as referential integrity.

The IBExpert Database Designer is an ideal tool for data modeling and design, whether creating a model of an existing database for analysis, or designing a new database.

Database normalization

The goal of normalization is to reduce redundant information. In other words, only store one piece of information one time. A table is said to have repeating groups and to be un-normalized if:

1. it contains many repetitions of the same piece of information in the same column
2. more than one column contains almost the same type of information
3. a column consists of complex information that should be broken into several smaller pieces.

Tables without repetitive values are described as normalized. The transition from one design to the other is called normalization.

Five forms of normalization can be differentiated. The first four normalization forms will be described very briefly here, the fifth being an extremely theoretical demand on tables. There is a wide range of specialist literature available on this subject, for those requiring more in-depth information.

Rule zero

The relational theory requires, as a rule, a unique key in each table, in order to identify information clearly. This is composed from the three following:

- The table, in which the data is stored,
- The field in this table, which needs to be accessed,
- The value of the primary key for this data set.

It is clear that the primary key is important for the identification of a data set. At the same time InterBase/Firebird automatically creates an index via the primary key, so that searches in multi-table queries are much quicker than those without an index.

A table has only one primary key, although the primary key can consist of several columns. So, a simple rule for normalizing databases is - always key your tables!

First normal norm

The first rule of database design states: eliminate repetitive groups. For each group of related columns, make a separate table and give that table a primary key.

A table is said to be in first normal form if all columns contain atomic (i.e. indivisible) values only. This is another way of saying that there are no repeating groups.

First normal form problems

`INSERT` anomalies (e.g. certain master data cannot be recorded until an order or sale is placed), `UPDATE` anomalies (it is too easy to miss certain entries when updating) and `DELETE` anomalies (whole records disappear from the database, including master data).

Second normal norm

The second rule of database design is: If a table column is only dependent upon part of a multicolumn key, this column should be removed to a separate table.

For a table in the second normal form, it must already be in the first normal form, and all non-key-column contents must be dependent upon the complete primary key. The second normal form avoids double storage of information. Tables become narrower, the more the database is normalized, with less duplication of wide column values. Where duplication is unavoidable, it can be made as small as possible by using an ID number.

Second normal form problems

There are no repetitive groups, and all columns are dependent on their table's primary key. However some irregularities can still be found; from the relational viewpoint, certain fields may have no relationship to each other, e.g. a customer telephone number has nothing to do with an order number. It is a customer feature, not an order feature, and leads to storage of redundant data. For this reason, it makes sense to remove this information to a separate table.

Third normal norm

The third normal form is tantamount to the second normal form, as it is also aimed to avoid `UDATE, DELETE` and `INSERT` problems. It is mainly concerned with relationships in tables with a single column primary key.

The rule can be defined: when column contents have no connection to the table's primary key, they should be removed to a separate table.

A table is in the third normal form, when each column describes data corresponding to the primary key.

Most operations are carried out on key fields, ensuring a high performance. Details are maintained in their own tables, secure from `UPDATE`, `DELETE`, and `INSERT` anomalies.

Fourth normal norm

The majority of applications need go no further than the third normal form. There are however certain situations, in which the data segmentation needs to be refined. For example, each sales team order needs to be assigned to the sales person responsible, for a planned monthly sales per person summary. Where should this information be stored? A simple solution is to expand the relevant table to include the field `SalesContact`.

The problem becomes clear, when it is considered that often more than one call was necessary to result in one sale. The fourth normal form rule is: isolate independent multiple relationships.

There are one or more calls leading to each order. The order position information has nothing to do with the telephone calls made. Therefore the call information is removed to its own table, to ensure that, here also, the independence of information in each table is warranted.

Enterprise-wide data model

New technologies are not a universal remedy: ways to achieve an enterprise-wide data model

Today almost all enterprises are fighting against a profusion of data, simultaneously suffering from a lack of useful information. Applications have grown isolated and exist in their own more or less well-documented data and file world. An important task of information management is to convert the multitude of data into a manageable amount of significant information. "Information as a resource" has integrated itself in the series of terms that have become common knowledge for data users. This keyword is commonly used and everyone now considers information to be of equal importance to the classical production factors capital, human resources and plant. Information management is an old hat which has finally been recognized and allocated its own organizational unit.

The persons appointed the responsibility for this information management are those who have so far been responsible for information systems: the DP or Organizational Manager. As an additional admonition, these managers are then required by general management to also consider old data as a new resource, and treat it with the corresponding diligence.

This viewpoint may be exaggerated, however the impression is given in many enterprises that by appointing an Information Manager, enough has been done to keep up with the new trend, and it is now possible to return to day-to-day business with responsibilities for:

- Hardware and software selection and implementation,
- Design of a hardware and software architecture for centralized and decentralized applications,
- Provision of the infrastructure for information users in the various enterprise sectors,
- Maintenance of standards and procedures.

But is that really all that information management needs to do? It is indisputable that the strategic direction of Information Technology is a considerable complex task of information management, the tasks mentioned above having become considerably more complex than they ever were.

Information management has lost its way in the data-processing jungle. The technical range, with its overabundance of possibilities, has not just become more extensive and complex, but has also brought with it compatibility and integration problems due to the lack of standardization; just consider the range of different network types, communication technologies, CIM products.

It's no wonder that information management can these days easily err in the data-processing jungle. But let's assume that the IT-technical world was different: strategically concise, tidier, clearly structured and without any technical problems.

What would then stop the enterprise from finally being able to fully utilize the longed-for possibilities to exchange all information as desired?

Everyone could then:

- within the realms of his authentication, independently
- use and alter others' information, create new information and make it available to others?

What is stopping them? This picture might be enticing, but unfortunately extremely deceptive. Because even the most perfect technology cannot hide the fact that, although bits and bytes can be distributed as wished, their information content could still continue to be unknown, or at least be misinterpretable.

By now it should be clear, that today's information management insufficiently fulfils the fundamental tasks of tomorrow:

- Information planning and information strategy
- Design of an application architecture
- Planning software applications

These three fields of responsibility are closely linked together, as an expedient planning strategy of individual software applications needs to be based on a previously compiled applications architecture, designed for the future.

The application architecture itself will need to be based on the results of the information plan and strategy, so that this task can be regarded as, in the long-term, the central logical basis.

The following remarks will therefore be confined to this basic function. There are two aspects to information planning. It demands firstly that you deal with the information itself - specifically and in detail. And it needs the managerial functions that create and process the information. However the lynchpin remains the information itself.

We are still confused – but on a global scale

So, initially the information is in the foreground. Information cannot be classified as such, until the data has been complemented by its semantic content, i.e. its meaning, thus becoming interpretable. However the current situation in most enterprises still predominantly mirrors the conventional picture of data processing and not that of targeted information processing. Applications systems that have grown isolated exist in their own world, where no one system is aware of the other, and which, at best, are only able to communicate via elaborate interfaces.

Data communication demands a common data appreciation though. However homonyms (terms with the same name but a different meaning) and synonyms (terms with different names but the same meaning) have become the order of the day in both application systems as well as in individual departments.

Applications, whose job it is to compile summaries and analyses, composed from base data from different operative systems for planning purposes or even as a tool to support enterprise decision making, find it extremely difficult to deliver reliable results. Reliability can only be achieved, when it can be assured that the base data do not just have the same name, but also the same meaning.

As clear definitions and descriptions for the data meaning are still missing in many enterprises, it is right to doubt the informational value of many an analysis or report. This situation cannot however be improved by implementation of new technology, which serves no other purpose than to distribute the dubious data more quickly.

New technologies alone may even make this problem worse, by ingeniously helping to expand localized chaotic situations into global ones, based on the principle, "We are still confused, but on a global scale".

Structuring data comprehensively and usefully

One of the most important tasks of information management is therefore to transform the multitude of existing data into a manageable quantity of meaningful information, in a structure that is both comprehensible and therefore usable for all information users.

This structure is the well-known data model. A data model is an illustration of the enterprise's information (or parts thereof) and their interrelations from a purely managerial point of view, independent of how they might be realized in the data-processing world. These days the importance of such an enterprise-wide data model is almost indisputable and its design and maintenance should be a task for data management, which is an integral constituent of information management.

Unfortunately in reality, surprisingly few enterprises dare to venture the construction of such a model. One the reasons for this appears to be fear of the word "enterprise-wide", as it gives the impression of an impossibly huge and insurmountable task.

But there are in fact realistic and viable ways by which "enterprise-wide" can be approached step by step, without having the rug pulled out from under your feet. One of these methods leads to what should here be called "enterprise-wide data model", the other leads to the resulting "enterprise data model".

The construction of both models is based on the same theoretically established and empirically tested method, that of the data model, which however will not be gone into detail here. Both models differ in their aim and, more than anything else, in their level of detail. Both models should enable information planning and information utilization globally across all projects, nevertheless each with a somewhat different specificity.

The enterprise-wide data model

The enterprise-wide data model corresponds to today's current established data model, and has the certainly extremely ambitious aim to achieve the following:

- A complete base of all information that the enterprise has to offer (including a professional data catalog), which is able to serve both as a detailed fundament of information and communication between departments, and aid with data processing.
- To provide a specification from which database structures can then be derived.
- To keep project interfaces small.

How is it possible to meet these high demands? Such a detailed data model cannot realistically be achieved in one simple step, but needs to be constructed from many small sub- data models. Each single partial model results from a project, which applies methodical data analysis. Each project creates a project-related data model, confined to its own informational area. The terms and concepts used in this data model however need to be clearly defined and be valid for the total enterprise.

The enterprise-wide data model evolves from the bottom up, arising from the union of the single project results into one consolidated structure.

Practice shows that this method has the following advantages:

- Each project recognizes the benefits of data analysis itself as an aid.
- The resulting "project data model" can be utilized immediately.
- The project result has been achieved to the great level of detail required, yet with a manageable amount of effort.

Problems arise however with this method when consolidating the partial models. It often becomes apparent at the interface of two projects, that the supposed enterprise-wide denomination and definition of the data is only actually fully valid from the limited project viewpoint, and now needs to be synchronized with the other projects. Information streaming increases project effectiveness.

This fine-tuning can be an elaborate process, which also in addition needs to take into account the human factor, namely the danger of those involved mistaking their own contributions and efforts as their property.

The process is also elaborate, because alteration to names and structures could have an effect on the results of other projects (e.g. functional flow descriptions), and other projects may need to adapt their results accordingly.

It is only possible to minimize this project-related annoyance if:

- Each sub-project is adequately informed of the enterprise's strategy with regard to mutual information, and feels sufficiently obliged to comply.
- Each project is kept informed right from the beginning of the results of previous or progress of current projects, and is able to use these actively, thereby saving expenditure, and even more importantly, effort.

This method produces immediate results, as even the initial results of the first project are a step towards information organization, without which information management is powerless in the long-term.

However the enterprise-wide data model cannot be used as a basis for information planning until at least two years later, as it takes this long for the results of the individual projects to be delivered, quality-controlled and synchronized with each other.

The enterprise data model however demonstrates its benefits rapidly, because it is constructed as an independent assignment, detached from other projects and with a different target: that of the enterprise data model.

The enterprise data model primarily follows a different target direction to the enterprise-wide data model. It does not aim to achieve a detailed data catalog and will never represent a complete base of all data in the enterprise.

In contrast it should:

- Provide an summary of the enterprise's information at top level,
- Recognize information supply areas ("theme" databases), according to which this information can grouped and summarized,
- Be a decision aid, enabling projects to be defined more precisely at their initiation, and last but not least,
- Produce a result with which enterprise management can demonstrate to all information users that they take the term "information" seriously.

These goals of a collective consolidated structural summary of the enterprise cannot be attained by joining the individual project results, bottom-up, and then integrating them upwards. An enterprise data model can only be developed as an independent and self-contained project, with participation of all management levels, as summary and not detail information is required here. By management we mean the specialist departments and sectors, as it is only here that data can be defined from a managerial point-of-view.

Data collation is achieved through interviews relevant for the level concerned and related professionally and technically.

Its goes without saying that the definition and description of this data will have a different quality to that of the enterprise-wide data model. In the enterprise data model relationships are identified clearly and precisely, always with the aim of simplification and rough abstraction. (Keep in mind that the objective here should not be a constant information refinement down to the last detail (i.e. an endless top-down process), but the construction of an approximate summary model that can continue to be maintained at this crude level.)

Project model with clear task definition

As the enterprise model is a self-contained investment, without direct implementation into a data application, its initial construction should be completed within a maximum 6 month time frame. The model can then be used immediately at project initiation.

The enterprise data model can be made immediately available and passed on to projects, along with a clear definition of which of the subject areas described belong to the project objectives. Project boundaries can be defined and established in relationship to each other right from the start, and subproject intersections can at least be fixed at a rough level.

This anticipatory description of intersections considerably reduces later investment for project adjustments, as each project knows its own "data limits", and is also aware from the start which other information management project partners he will need to confer with for the fine-tuning phase.

The results of this project refinement are not included in the enterprise data model, but grow together to form their own independent enterprise-wide data model.

The enterprise-wide data model and enterprise data model are therefore not "either-or" models, but are, in the true sense of the word, "as-well-as" complements. But what do both these models have to do with future-oriented information planning and information strategy, are they not managed by data administration?

The problem today lies in the term administration, which has little to do with management and consequently with strategy and planning. Many enterprises have started constructing an enterprise-wide data model, but its use is still mainly limited to the unification of terms and information correlations. Compared to the alternative of prevailing data chaos this limitation is however still a huge step forward, which itself is worth a certain amount of effort.

However data and information planning require more, and open up in the long-term other perspectives. Information planning should achieve the goal of comparing the enterprise's current information supply situation with future visions and to assess them, in order to recognize supply deficits and to be able to simulate and optimize future supply situations. This however assumes that not only the enterprise's information is known, but also the functions that are connected to the use of this information. Principally only the comparison of a data model with the functions or functional areas that are necessary for the improvement or extensions of strategically important business areas, allows informational gaps and superfluous informational ballast to be detected. But what does this mean, in view of both the above-mentioned types of data model?

The enterprise-wide data model is only suitable for detail planning aspects, because of its level of detail; that is, when dealing with the concrete definition for contained single subject areas. In contrast, this model is unsuitable for planning or strategic aspects. And yet it is the only model, in many cases, that is (at least in part) used by information management.

Foundation for theme databases

But who begins with strategic management tasks in other enterprise areas at the operative level, such as the Internal Revenue? Information management is often degraded to the level of dealing with nothing other than the daily business (enterprise-wide data model), instead of setting the basis for management tasks: the enterprise data model.

The enterprise data model offers management (and not just information management), for the first time and within a short period of time, a defined basis for their own information and informational areas (theme databases) which is comprehensible to all.

This model allows a meaningful and clear classification of information for the enterprise's functional areas and organizational units. It supports the construction of a more comprehensive model, with which a conscious design and simulation of future information management is made possible.

And there is one more advantage: the enterprise data model includes business management, in its target-oriented way of thinking, in information and informational correlations right from the beginning. Data model and information management become accessible to all concerned, becoming today's obligation to go on the "search for tomorrow's information".

Space Management in InterBase

By Ann Harrison, IBPhoenix.

An InterBase database consists of a set of fixed length pages of different types. Ten page types are currently defined:

- Header page (HDR)
- Data page (DPG)
- Blob page (BLP)
- Transaction Inventory Page (TIP)
- Page Inventory Page (PIP)
- Pointer page (PTR)
- Index Root page (IRT)
- B-tree page (BTR)
- Write-Ahead Log page (LIP)
- Generator page (GEN)

Two of these, page inventory and pointer are used for space management. For those not familiar with InterBase's on-disk structure, the next article, *Page Types*, includes a brief description of each of the page types.

Page types

All page types include a header that holds generic page information.

```
typedef struct pag {
    SCHAR pag_type;
    SCHAR pag_flags;
    USHORT pag_checksum;
    ULONG pag_generation;
    ULONG pag_seqno; /* WAL seqno of last update */
    ULONG pag_offset; /* WAL offset of last update */
} *PAG;
```

Each specific page type adds more structural information. The first page in every database is its header (HDR) page. Secondary database files also have header pages. Data pages (DPG) contain data; blob pages (BLP) contain blob data for those blobs that don't fit on the data page with their parent record. Any data page contains data for only one table. Any blob page contains data for only one blob. Transaction inventory pages (TIP) contain an array of bits, two per transaction, that indicate the state of the transaction. A transaction id is an index into this array. Every page in the database is represented by one bit in a page inventory page (PIP). The bit indicates whether the page is currently in use. Page inventory pages (PIP) occur at fixed intervals in the database - the interval is determined by the page size. A pointer (PTR) page is the top-level locator for data pages. It contains an array of page numbers for the data pages of the table and a corresponding array of bits that indicate whether the page is full. No pointer page entry is made for blob pages or pages that contain only the second or subsequent pages of data from a fragmented record. Index (IRT) root and b-tree (BTR) pages are what they appear to be. The only odd thing is that each table can have only one index root page. For that reason, you can put more indexes on a table when you use a large page size. The log information pages (LIP) for the write-ahead log are not currently used, though code to use them is included conditionally. Generator pages (GEN) contain arrays of 32 or 64 bit integers, depending on the dialect.

Basic page allocation

Page allocation is handled by the routine `PAG_allocate` in `PAG.C`. When some routine needs a new page, it calls `PAG_allocate`. `PAG_allocate` gets the page control block from the database block to find the first page information page that has free space. If necessary, it reads that pointer page from disk. It then scans the page, looking for the first free bit, and assigns that page number to the new page. The page image is created in the cache manager (CCH), which give it the appropriate page type. The cache manager then returns the buffer pointer to the routine that requested the new page. When the page is marked for write, the page I/O module (PIO) writes it to the appropriate offset in the database file. *Housekeeping Note:* To keep the database on disk consistent, the pointer page must be written before any page that is allocated from it to avoid doubly allocated pages. Under ordinary circumstances, the shared cache or page locks keep this from happening. If, however, the machine were to crash in mid-operation, the order of page writes can prevent corruption.

Advanced page allocation

If the system does not find space on the first PIP it examines, it reads the next, and so on until it searches the last PIP. If the last unallocated page is the last bit on the last PIP, the routine allocates that page number as the next new PIP, formats it, marks the new PIP as needing to be written and the old PIP as dependent on it. Finally, `PAG_allocate` calls itself to allocate the page that was requested originally, using the first bit on the new page inventory page. If the database is defined to hold multiple files, when page allocation reaches the end of the first file, it creates a new file, gives it a new header, and resumes allocating pages.

Additional page allocation steps for data pages

A data page is recorded as being in use both in the PIP and in a pointer page for that table. Once the new data page has been marked for write, its page number is written into the first free slot one in the current pointer page or the first free slot on any pointer page. The order of writes is: PIP, data page, pointer-page.

Additional steps for interesting pages

Information about interesting pages is stored in a system table called RDB$PAGES. When an index root page, a transaction inventory page, a generator page or a pointer page is created, a new row is stored in RDB$PAGES. This operation can cause a new page, a new pointer page, a new page inventory page or even a new file to be allocated.

Releasing pages

The header page is never released. Generator pages and transaction inventory pages are not released either. In theory, they could be, but that would complicate (slightly) some sensitive bookkeeping for (relatively) little gain. Nor are page inventory pages released. Once a database has grown to some size, the only way to shrink it is to recreate it from a backup. When a page is empty, it is put back in the "free space pool" by clearing its bit on the appropriate page inventory page. B-tree pages are released when the index is deleted, deactivated, or rebalanced. Blob pages are released when the blob is released, because the record that owns it is deleted or because the blob itself was modified. Data pages created to hold the trailing part of a fragmented row are released when the row - or at least that version of the row - is removed.

Releasing data pages

When the last row on a normal (non-overflow) data page is deleted, the page is returned to free space in a two-part operation. First, the page is removed from its pointer page, which is the page that associates it with its table. If that empties the pointer page, then the pointer page is also marked as released on its page inventory page. Releasing a pointer page requires changing a system table called RDB$PAGES. RDB$PAGES contains one row for each "interesting" page in the database. Pointer pages, index root pages, generator pages, and transaction inventory pages are considered "interesting". Releasing an index root page also requires deleting a row from RDB$PAGES. This process can recurse, just as the allocation process recurses, except that neither files nor page inventory pages are released.

Elementary allocation on page

For most of the page types, allocation of space on page is not difficult. Generator pages, transaction inventory pages, page inventory pages, and pointer pages are just [[Field Definitions][#Array | arrays. When one page fills, another one is allocated. (Theoretic rather than actual in the case of generator pages, but the principle holds). Routines in the module PAG.C manage header pages - they are essentially simple structures followed by a byte array that holds the filenames for secondary files. Space on generator pages and transaction inventory pages is never reused, so there is no reason to look for space on any page of those types except the last. Space on page inventory pages is reused. When a page is released - no longer needed for whatever purpose it had - its entry is cleared. For that reason, the page number of the lowest PIP with space is carried in the database control block. That number is not considered reliable, but a good starting point.

Finding space for data

Each table carries with it a vector of its pointer page numbers, and two high-water marks, one for the first pointer page with data space, and one for the first pointer page with space for a new data page. When storing a record that compresses to less than the page size, DPM looks first for a pointer page with data pages that have free space, then at the header of the pointer page to find the first slot pointing to a page with space.

Now, just a bit more about data pages. Every data page has a header like this:

```
typedef struct dpg {
    struct pag dpg_header;
    SLONG dpg_sequence; /* Sequence number in relation */
    USHORT dpg_relation; /* Relation id */
    USHORT dpg_count; /* Number of record segments on page */
    struct dpg_repeat
    {
    USHORT dpg_offset; /* Offset of record fragment */
    USHORT dpg_length; /* Length of record fragment */
    }  dpg_rpt [1];
} *DPG;
```

The repeating offset/length is an array of pointers to data on the page. These pointers are called line index entries, at least by me. The actual data starts at the bottom of the page and works up. When there is no longer enough space for another line index entry and another minimal sized record, plus whatever space is reserved for future expansion (that's another topic), the page is marked full, both in its header and on the pointer page.

DPM goes through the line index, adding up the space on page. If there's enough for the compressed record, alignment overhead, and a line index entry, it's got a winner. However, the space may not be contiguous. In that case, DPM shuffles all the data down to the bottom of the page. Obviously, it doesn't compress the line index entries, though it does correct the offset for data that has moved. Next step is to create a new line index entry and shoot the data onto the page. Final step is to see if the page's fullness quotient has changed and make appropriate changes if so.

If there is space on page, but not enough for the current compressed record, DPM marches on through the pointer page, checking plausible candidates, then on through other pointer pages until there are no more allocated data pages.

OK, now it's time to allocate a new data page. First, find a free page in the current PIP, or the next PIPs, or create a new PIP. Next, create the page in a buffer. Now, starting with the first pointer page that has space to hold a new data page pointer, or create a new pointer page for the table. That's it. At least that's all I can explain at the moment.

See also:
Firebird for the database expert: Episode 2 - Page Types

Multi-generational architecture (MGA) and record versioning

InterBase introduced multi-generational architecture (MGA) as the term for its implementation of multiversion concurrency control.

Multiversion concurrency control (abbreviated MCC or MVCC) is the method used to prevent two or more users changing a single data set at the same time. It provides each user connected to the database with a "snapshot" of the database for that person to work with. Any changes made will not be seen by other users of the database until the transaction has been committed.

Firebird and InterBase implement this architecture using record versions. For example in dBase when a data set is altered, dBase overwrites the old version of the data set with the new in the database file. The old version of the data set is lost for ever. The Firebird server processes the data manipulation differently: when a data set is updated, Firebird creates a new data set, recording the differences between the original data set in its original state and the new updated content. And when a data set is deleted, Firebird also creates a new data set (flagged as deleted)! For the simple reason, if a mistake has been made and the transaction needs to be rolled back, the data set fully recovered.

These record versions are maintained by Firebird - parallel to the original data sets - until a `COMMIT` or `ROLLBACK` has been executed or until the server is restarted (when Firebird restarts it rolls back all active transactions).

But not just the *active* transactions are stored. For example: User A checks the bank balance ($1,500) makes a bank account withdrawal of $1,000. Just then the great-looking guy from the office next door rings and asks if she's free for lunch. User A drops everything and rushes out to lunch, forgetting to commit her transaction, thus leaving it open. In the meantime User B checks the bank balance (still $1,500) and withdraws $800, not forgetting to commit his transaction before he goes to lunch. User C likes to work through lunch, and whilst User A and B are out, he withdraws (bank balance now $700) respectively, $100, $200 and $300.

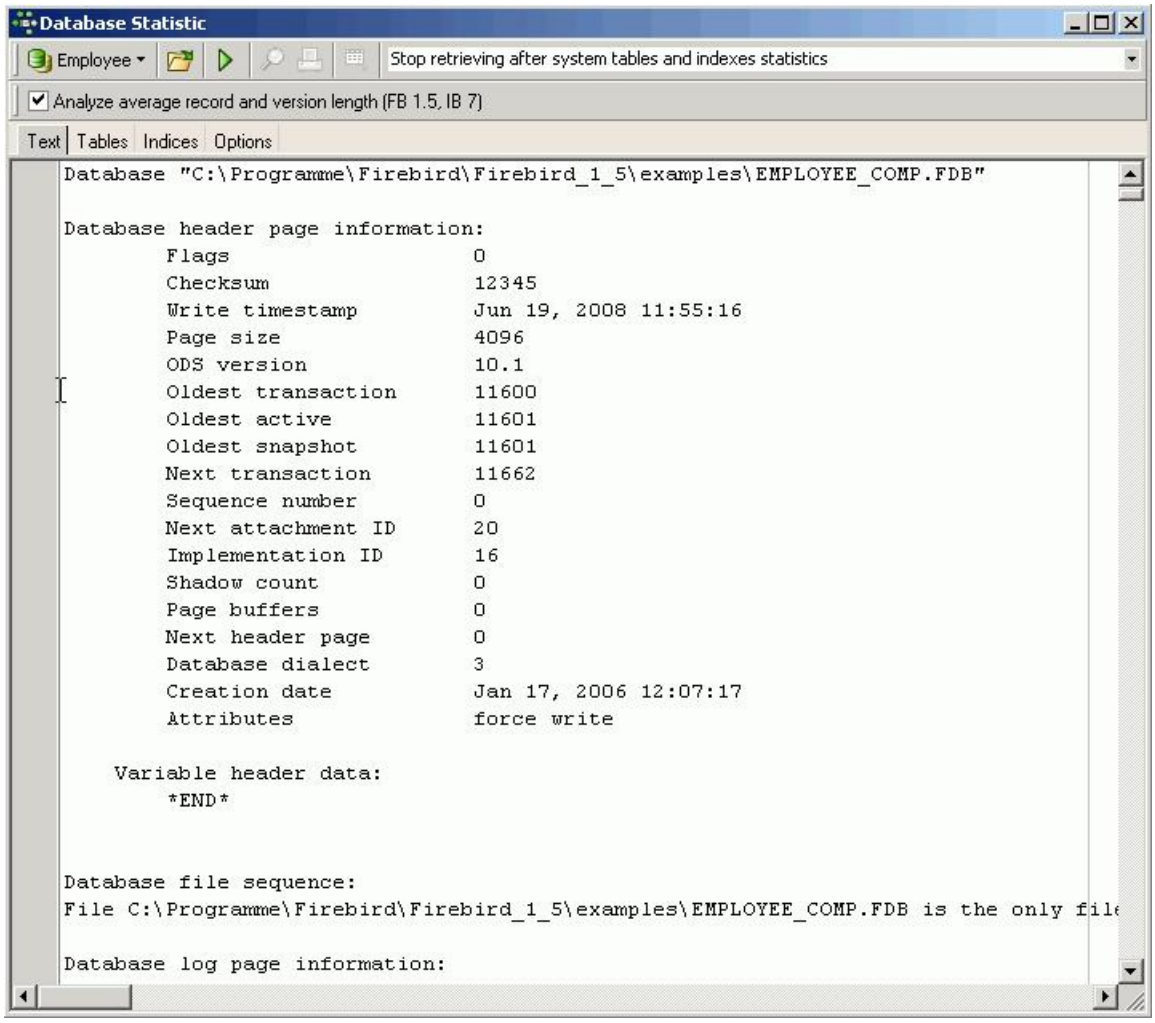
Not only is the record version for User A's active transaction stored. The 4 transactions made by Users B and C also have to be stored, because they were made after User A's transaction. In fact, all transactions which follow User A's cannot be completed and garbage collected until she has committed or rolled back her transaction. What if she and the "good-looker" fall so madly in love, they spontaneously decide to elope and never return to the office? It quite simply means that *all* record versions from this date on will remain on the database file as record versions, which will obviously soon start to slow performance considerably, unless someone finds her active transaction and rolls it back, or the server is restarted.

Database Statistics

Poor or degrading database performance is practically always to do with poor programming and/or poor transaction handling.

Database Statistics are an invaluable insight to what is actually happening on the server. Firebird statistics should be evaluated regularly and kept, because when things do go wrong, it's immensely helpful to be able to see what they looked like when thing went right.

The Database Statistics display the following information for all tables in the database, both as a log script and in tabular form: table name, location, pages, size (bytes), slots, fill (%), DP usage (%) and fill distribution (an optimal page fill is around 80%). For each table the indices statistics include: depth, leaf buckets, nodes, average data length and fill distribution. Further information regarding these statistics can be found in the IBExpert Services menu item, Database Statistics.



Analyzing transactions

Under the *oldest transaction* we can see the oldest transaction number that cannot yet be garbage collected. To ensure efficient performance, the difference between this number and the *next transaction* number should be kept as small as possible. This depends of course on the number of users and database activity. For example, if you have 160 users working on one database, a difference of 3,000-5,000 is probably perfectly acceptable. However if there are only 2 users working on the database, you should be concerned if the difference between the oldest and next transaction is in the range of 3,000-5,000.

The fault can usually be found in the programming. For example a select query that's never committed or rolled back. One secure way of ensuring active transactions are rolled back is to temporarily disconnect any user, that has not actively used the application for the last half hour. There are great components on the market for this, e.g. FIBPlus and IBObjects.

By the way: the *next transaction* value may not exceed 1.4 billion. At the very latest at this stage you will need to do a backup and restore, as the restore sets all transactions back to zero. However, at an average rate of one transaction per second, it would take 130 years to reach this number, and even if 10 transactions a second are performed, it will take 13 years!

It's important to observe the degradation when things slow down. For example, running a select every second, and watching the prepare and execute time can be a good indicator. When this begins to slow, it's a premptive that something is wrong, and you will find within a few hours that the database will begin to slow, unless you find the source of the problem quick.

In daily usage, the oldest active transaction should not stay on a specific value for a long time, when the next transaction is constantly increasing.

If the oldest transaction is lower that the oldest active, use `GFIX` or any other tool for that matter, to sweep the database.

```
SQL> gfix
CON> Expected end of statement, encountered EOF

C:\>gfix
please retry, specifying an option
plausible options are:
        -activate        activate shadow file for database usage
        -attach          shutdown new database attachments
        -buffers         set page buffers <n>
        -commit          commit transaction <tr / all>
        -cache           shutdown cache manager
        -full            validate record fragments (-v)
        -force           force database shutdown
        -housekeeping    set sweep interval <n>
        -ignore          ignore checksum errors
        -kill            kill all unavailable shadow files
        -list            show limbo transactions
        -mend            prepare corrupt database for backup
        -mode            read_only or read_write
        -no_update       read-only validation (-v)
        -online          database online <single / multi / normal>
        -prompt          prompt for commit/rollback (-l)
        -password        default password
        -rollback        rollback transaction <tr / all>
        -sql_dialect     set database dialect n
        -sweep           force garbage collection
        -shut            shutdown <full / single / multi>
        -two_phase       perform automated two-phase recovery
        -tran            shutdown transaction startup
        -use             use full or reserve space for versions
        -user            default user name
        -validate        validate database structure
        -write           write synchronously or asynchronously
        -z               print software version number

    qualifiers show the major option in parenthesis
```

See also:
Record versions as an undo log
Firebird for the database expert: Episode 2 - Page types
Firebird for the database expert: Episode 4 - OAT, OIT and Sweep
IBExpert Database Statistics
Transaction
GFIX

Multi-version concurrency control

A not-so-very technical discussion of Multi-Version Concurrency Control

Origins of conflict

In February 2002 Oracle published a "Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance" white paper where they claimed to have better architecture in Oracle 9i compared to IBM DB2 UDB V7.2. In August 2002 IBM published "A Technical Discussion of Multi-Version Read Consistency" white paper claiming that Oracle multi-version concurrency is not better than the approach used in IBM DB2, but requires many workarounds to achieve needed results.

Traditionally, the problem of concurrency is solved using locking. If A needs access to resource N, it locks it; after use the lock is released. If B wants to access resource N while A is using it, it must wait. It is clear that such approach may give very poor results when the locks are applied at a very high level – consider the example of two editors editing different chapters in a big MS Word document. MS Word blocks access to the document file at the file system level. While the first editor is able to modify the document, the second must wait until the first one finishes editing. And this is correct, since the second editor does not know what changes were made by the first one in general. However, MS Word gives an option to open the document in read-only mode, allowing the second editor to read the chapter, and plan what to change on the "secondary storage", read "using a pen and a sheet of paper". When the first editor finishes editing, the second editor re-opens the latest version of the document in a read-write mode and "applies" the changes noted on the paper.

In its white paper Oracle claims that IBM DB2 UDB V7.2 EEE, which uses locking as in the example above, has poor concurrency, citing the "Oracle to DB2 Porting Guide": "As a result of different concurrency controls in Oracle and DB2 UDB, an application ported directly from Oracle to DB2 UDB may experience deadlocks that it did not have previously. As DB2 UDB acquires a share lock for readers, updaters may be blocked where that was not the case using Oracle. A deadlock occurs when two or more applications are waiting for each other but neither can proceed because each has locks that are required by others. The only way to resolve a deadlock is to roll back one of the applications."[1]. In response, IBM claims that Oracle's multi-version architecture does not solve the problem, since now the database engine has to do much more I/O to access needed record versions and the disk space for record versions is limited, and, when it is filled completely, transactions are rolled back with a ORA-1555 "Snapshot too old" message. IBM also claims that approach used in Oracle gives incorrect results under some conditions and additional programming is needed to solve the issue.

Firebird case

InterBase, the predecessor of Firebird, was among the first commercial databases to implement multi-version concurrency control (MVCC)[2]. This makes the behavior of Firebird close to Oracle, however with a notable difference – Firebird is naturally multi-versioned, while Oracle acquired this feature in Oracle 7.x. Until than it had an architecture similar to IBM DB2. Firebird simply does not have the negative issues emphasized in the both white papers, while using all advantages of MVCC.

Concept

So how does it work? The main idea was already presented when we talked about MS Word opening a file in read-only mode, but there are some important details. As the name implies, each record in the system might have multiple versions visible to different transactions. When a transaction modifies a record, a new version is written to the database, and a previous version, representing only the difference between the version of the record that was read by the transaction and the new value of the record, is written as a back version of that record.

How does the system know which version is visible to which transaction? When a transaction starts, it receives a singly incrementing number. This number uniquely identifies the transaction within the system during the lifespan of the database since the last restore. Every change that is done in the database is "signed" by the transaction number. When a record is read on behalf of some transaction, the database system compares the "signature" of the record with a transaction number. If the "signature" belongs to a transaction that was committed when the current transaction started, that version is returned to the application. Otherwise, the database engine computes the needed version using the current record state and the back versions of that record without regard to the locks that the writing transaction has.

This is very simplified description of what happens in Firebird, for more technical details please read the Firebird for the Database Expert: Episode 4 - OAT,OIT & Sweep article. Ann W. Harrison provides an excellent description with examples that illustrate the whole complexity of this issue.

Similarities and differences

The description above should be enough to see that Firebird functions similarly to Oracle 9i.

- Multi-generational architecture allows different transactions to avoid conflicts between readers and writers. The reading transaction can always see a consistent view of the database regardless of the write operations that are happening concurrently. IBM DB2 can provide such level of concurrency only sacrificing the database consistency and using dirty reads.
- The mechanism of back versions in Firebird is similar to the rollback segments used in Oracle for the same purposes. Both systems are optimistic, in other words, they assume that, in most cases, an application will not need previous versions of the records. The optimization is performed to give the best performance to the most likely case.

But unlike Oracle, Firebird cannot produce anything similar to the ORA-1555 "Snapshot too old". There is no need to estimate the size of the rollback segments as described in the IBM white paper, since all information needed for rollback operations and computing previous record versions is stored inside the database itself and the database file grows automatically if more space is needed.

However, the approach used in Firebird has its price. What Oracle solves by rolling the rollback segments over, and which finally leads to the ORA-1555 "Snapshot too old" error, Firebird must handle differently.

The first issue is long record version chains. Oracle drops rollback segments when they get too large. Firebird never drops a back version if it could be seen by any running transaction. As a result, a long-lived transaction blocks the removal of back versions of all records, causing the database to grow and performance to deteriorate. The performance cost is due both to the decreased density of valid data and to the cost of checking whether any back versions of records can be deleted.

A second issue is the cost of removing back versions. Oracle's back versions are in a separate segment. Firebird's back versions are in the database, so they must be removed one at a time, as they are encountered by subsequent transactions.

A third issue is the cost of a rollback. When a transaction inserts, updates, or deletes a record, Firebird changes the database immediately, relying on the back versions as an undo log. A failed transaction's work remains in the database and must be removed when it is found.

Firebird successfully handles these cases without user intervention. Its behavior is controlled by a few parameters, like "sweep interval". However detailed discussion is out of the scope of this paper: please see the Firebird documentation for more details.

It is worth mentioning one very nice "consequence" of the fact that there is no recovery log. Firebird has to take additional care to keep the database file in a consistent state – if a crash happens, there is no other place where information can be recovered except the database file itself. This is achieved using the careful write technique – Firebird writes data onto disk in such a manner that, at every single moment, the database file is consistent. The careful writes feature is something that really makes the life of the end-user easier. In addition to automated database housekeeping, Firebird has also automated crash recovery – a truly DBA-free database engine.

The next critique of Oracle's versioning mechanism is what IBM calls an ability to see current data. The example on Illustration 1 is used to demonstrate the weakness of Oracle 9i.

Time	Transaction 1	Transaction 2
1.	Begin transaction.	
2.		Begin transaction.
3.	Select available seats on flight ABC111. See seat 23F is the last seat available reserve this seat.	
4.		Select available seats on flight ABC111. Also sees 23F as Oracle will go to the rollback segment to get the old version of that block.
5.	Commit transaction.	
6.		Reserve this seat.
7.		Commit transaction. Successful but now the flight is oversold.

Illustration 1: Example IBM used to show incorrect logic in Oracle 9i version control.

So, how does it apply to Firebird? It will not work. Firebird reports an error on step 6. The logic is quite simple in this case. At the beginning of the operation, both transactions saw a record version signed by a transaction, let's say, 120. When transaction 1 committed on step 5, the new record version was signed with a number of transaction 1, lets say, 125. Now, if transaction 2 tries to update the same record, it will find that the version of the record is no longer 120, but 125, and will report an error to the application. The update operation will not succeed.

Furthermore, the same error will be reported if step 6 happens before step 5, but after step 3. It is also possible to tell transaction 2 to wait until transaction 1 finishes and then decide the outcome of the operation. If transaction 2 is lucky and transaction 1 is rolled back (for example, the customer booking a seat in transaction 1 changed his mind), it will successfully book the seat for the second customer. In case of IBM DB2, the lock conflict would have happened already in step 4, since transaction 2 would try to lock a record that had already been modified by transaction 1. The change of mind by the first customer does not help the second one. The application has to re-read the table and check for a new seat for the booking.

Conclusion

From the above it is clear that multi-version concurrency control, if implemented correctly, provides a superior concurrency in cases when update conflicts are rare compared to traditional pessimistic locking schemes. It is also clear that there are cases when pessimistic locking will perform better. However, the claim made by IBM that multi-version concurrency control is not used in most database systems is no longer true since Microsoft has decided to switch to MVCC in the next version of SQL Server (code name Yukon). Now two of three biggest commercial database vendors use MVCC. In fact, the versioning mechanism used in Yukon is almost an exact copy of the mechanism used in Firebird. It took almost 20 years for other software vendors to find out that MVCC is great approach to handle concurrent access to the database.

Acknowledgments

The author is grateful to Ann W. Harrison and Helen Borrie for their comments and help during the preparation of this paper.

References

A Technical Discussion of Multi Version Read Consistency, By IBM Software Group, Toronto Laboratory August 2002, ftp://ftp.software.ibm.com/software/data/pubs/papers/readconsistency.pdf.

Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance, An Oracle White Paper, February 2002.

About the Author

Roman Rokytskyy is one of the Firebird Project members, leader of the JayBird subproject, the JCA/JDBC driver for Firebird.

[1]Oracle to DB2 Porting Guide, page 47, http://www.db2udb.net/guide/program/text/oraclev3.pdf

[2]According to Ann W. Harrison, first was Rdb/ELN released in 1984 by DEC, second was InterBase, both designed by Jim Starkey. Later DEC decided to push Rdb/VMS, which had the same API, but was implemented completely different, so InterBase can be considered the first database using MVCC that survived to our days.

Using IBExpert and Delphi applications in a Linux environment accessing Firebird

By Luiz "RedDevil" Stefanski.
(Revisions by IBExpert KG)

Initial Topics

Before writing this article I performed tests in a complete installation of the Conectiva Linux version 10 without the 4th CD (the Update CD), so I had no problems with dependencies, except with WineTools. I therefore installed the packet/libraries `"gtk+-devel"`, `X-dialog` and `glibc`. The version of Wines used for executing IBExpert is 0.9.5. I did not test Delphi applications with this version of Wine. The Delphi application executed fine with Wine version 20041019, but when using Wine version 0.9.5 I detected the application's screens were not displayed correctly, or maybe it depends on some extra configuration I have not yet discovered.

Introduction

Linux is being used in homes and enterprise plants all over the world; Linux is gaining space and growing all the time, not only in server installations where it is sacred, but in desktop installations too. Therefore it is unavoidable that developers will have to have contact with the Penguin at some time.

When this happens a good developer will discover that Linux is not a monster, principally because Linux has made a lot of transformations and now it is much friendlier for the end user.

Talking about Firebird: they have a native version for Linux, but we can also manage and develop applications to run in Windows accessing Firebird in Linux. One example of this management tool is IBExpert, and executing IBExpert in Linux. We can do this with a lot of applications developed in Delphi (Windows).

There is a lot of software emulators for Linux, like DosEMU and WABI, this software make DOS applications and Windows 3.1 applications execute on a Linux platform. To execute IBExpert we will use one of the best ways to do this: WINE (Wine Is Not an Emulator). The own name affirms that WINE is not only an emulator. It maps and converts calls in the Windows API to the Linux API, this way the Windows programs are actually deceived, because they "think" they are being executed in Windows, but are actually executing in a virtual Windows called Fake Windows created by WINE to execute in the specified Windows directory (`~/.wine`).

Download and installation of Wine

In this article we use the version 0.9.5 of Wine, the last version is the version 1.0 (you can see this in the WINEHQ (http://www.winehq.org/), the official site of WINE, the version 0.9.5 is however sufficient for us. Maybe the early versions of Wine execute the Delphi applications better, but I tested IBExpert with version 0.9.5 of Wine and it works fine. This is also the nearest version to Wine 0.9 – Beta, the preferred version when using WineTools 0.9, and for the front-end we will install some Microsoft Windows components in our fake windows, - very important in order for the software to work smoothly in Wine.

If you are an advanced Wine user you may prefer to use the early versions of Wine or install the Wine version for your Linux distribution. If you want to do this, use the link for Sourceforge and track the steps beginning the installation from the `archive.tar.bz2`. You can download this file from a multitude of mirrors (see table below). This is not of interest for newcomers.

Download Wine 0.9.x

Mirror	Action
New York, New York	Download
Atlanta, GA	Download
Phoenix, AZ	Download
Sydney, Australia	Download
Dublin, Ireland	Download
Paris, France	Download
Kent, UK	Download

Table 1: Mirrors for downloading Wine 0.9.5.

Important: When I wrote this article the latest version of Wine was version 0.9.5. You can however now find newer versions. Wine offers retrocompatibility, and this can work the same way. By only changing the names you can adapt the scripts to new versions.

After downloading the `wine-0.9.5.tar.bz2` archive, we need to compile and install the Wine. So you need to log in again as root user and move the downloaded file to directory `/root`. Open a new shell and type the comands:

```
[root@hades root]# tar -jxvf wine-0.9.5.tar.bz2 [Enter]
.
. (list of unpacked files)
.
[root@hades root]# cd wine-0.9.5 [Enter)]

[root@hades wine-0.9.5]# ./configure [Enter]
```

The last comand maybe delayed some moments.

Following completion type in the same shell:

```
[root@hades wine-0.9.5]# make depend && make [Enter]
```

Now, the next instruction will be delayed... This depends on the CPU used. I suggest you go drink a coffee, or read a good book until the compilation finishes! After this we need to install Wine by typing the instruction:

```
[root@hades wine-0.9.5]# make install [Enter]
```

If all works fine, Wine is now installed and ready for use.

Do not run IBExpert (if you already have it installed) before carrying out the next steps. If you have done, you may need to delete `.wine`.

Download and installation of WineTools

WineTools is the front-end used to manage our fake windows; this tool will install the software and components necessary for Windows programs run inside Linux.

As I already mentioned, we will use the version winetools-0.9, the download can be done in 2 ways:

- If you prefer the installation in a `RPM` pack, select it
- If you prefer to track all steps and use the `archive.tar.gz` can download it from the link below.
  Download `winetools-0.9 em RPM`
  Download `winetools-0.9 em .tar.gz`

After downloading the file `winetools-0.9jo-III.tar.gz`, and logging in as root, move the file to the directory `/root` and open a new shell. Inside the shell, type these instructions:

```
[root@hades root]# tar -xzvf winetools-0.9jo-III.tar.gz [Enter]
.
. (list of files beeing unpacked)
.
[root@hades root]# cd winetools-0.9jo-III [Enter]

[root@hades winetools-0.9jo-III]# ./install [Enter)]
```

Ready! The installation is finished. Looking at the illustration below you can see some strange messages (this can happen ;-), like `command not found`, but this does not affect the performance of WineTools.

```
[root@hades winetools-0.9jo-III]# ./install
Version: winetools-0.9
Release: 0.9
You do not have gettext installed on your system. WineTools will not run in your native language
. Rather it will use english.
Installing WineTools to /usr/local/winetools...
Installing translations...
./install: line 58: msgfmt: command not found
Installed translations for de_DE@euro to /usr/local/share/locale/de_DE@euro/LC_MESSAGES/wt0.9.mo
./install: line 58: msgfmt: command not found
Installed translations for es to /usr/local/share/locale/es/LC_MESSAGES/wt0.9.mo
./install: line 58: msgfmt: command not found
Installed translations for fr to /usr/local/share/locale/fr/LC_MESSAGES/wt0.9.mo
./install: line 58: msgfmt: command not found
Installed translations for nb to /usr/local/share/locale/nb/LC_MESSAGES/wt0.9.mo
./install: line 58: msgfmt: command not found
Installed translations for pt_BR to /usr/local/share/locale/pt_BR/LC_MESSAGES/wt0.9.mo
Start WineTools as *normal* user with "wt". Don't use as root!
[root@hades winetools-0.9jo-III]#
```

Illustration 1: Installation of WineTools

To close the installation, still logged in as root, follow the procedures listed below to download a script to use the WineTools. Open a shell and type the instructions:

```
[root@hades root]# cd /usr/local/bin [Enter]

[root@hades bin]# wget -vc http://www.reddevil.eti.br/gettext.sh [Enter]
.
. (receiving the file)
.
[root@hades bin]# chmod 777 gettext.sh [Enter]

[root@hades bin]# exit [Enter]
```

Creating and managing the Fake Windows (installations)

Now we will create our Fake Windows (we will call it Fake). This is necessary to run Windows applications inside Linux.

Caution! The root should never be used to work with Wine, so to this end we will use a common user, with rare exceptions, our user needing to belong to a group firebird (and while I wait our Firebird is installed and running in Linux). If you have not yet installed a Firebird server, you should do it now (http://www.firebirdsql.org) before continuing to read this article. Please refer to Download and Install Firebird for installation details.

In this article I will be using a linux user called `reddevil`, so when I refer to the directory `~/.wine`, I will be referring to the directory `/home/reddevil/.wine`, OK?

Another important detail we need to determine is that Firebird is running in the localhost, or be situated on the same computer on which we are working, and running (loaded) in the default directory (`/opt/firebird`). If someone needs to use a remote Firebird the results will be the same, but it is necessary of course to make certain alterations correspondingly.

So, let's go! Firstly make sure you are logged in as a common user – that user will execute the windows programs. Open a shell and type the instruction. The result is presented in the illustration below.

```
[reddevil@hades reddevil]$ wine [Enter]
```

```
[reddevil@hades reddevil]$ wine
wine: creating configuration directory '/home/reddevil/.wine'...
fixme:midi:OSS_MidiInit Synthesizer support MIDI in. Not supported yet (pleas
e report)
fixme:midi:OSS_MidiInit Synthesizer support MIDI in. Not supported yet (pleas
e report)
wine: '/home/reddevil/.wine' created successfully.
Wine 0.9.5
Usage: wine PROGRAM [ARGUMENTS...]   Run the specified program
       wine --help                   Display this help and exit
       wine --version                Output version information and exit
[reddevil@hades reddevil]$
```

Illustration 2: Initialization of Wine

This instruction starts Wine and creates the Fake, to prepare the environment to run the WineTools.

Now, we will access the WineTools for the first time. If you are using KDE press [Alt] + [F2] or use the `fbrun` in the Fluxbox, or open a shell and type `winetools` (in lowercase). Click the *OK* button in the 3 initial screens and done, you are in the *Main Form* of WineTools, and now you have a very intuitive interface, as in illustration 3.

Illustration 3: WineTools Main Form

So, let´s begin by selecting the option *Base setup*, the first option in the WineTool's main form, this option should be already highlighted by default, if not, select it! Then click the *OK* button to access the menu *Base Setup* of WineTools, as in illustration 4.

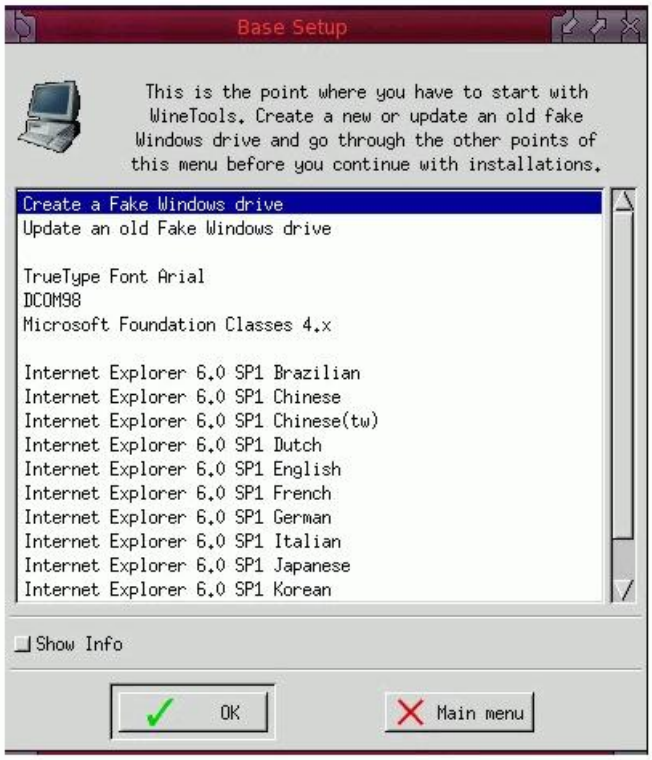


Illustration 4: WineTools Menu Base Setup

Select the option *Create a fake Windows drive* and confirm, answer *YES* to the question *Remove existing Wine configuration?*. Click the *OK* button to confirm the path of the CD ROM in `/etc/fstab` (or change it, if this isn't correct) and to answer the questions *What's your username?* and *What's your organization?* you can fill in some information, like, `user` and `home` (if necessary the information can be changed in the file `/~wine/system.reg`). Wait for the confirmation Fake Windows drive created in `~/.wine`. Click in the *OK* button again and wait for the *Base Setup* menu of WineTools.

The last procedure has created our Fake, with certain changes created when Wine is started. These changes may be altered using `WineCfg` and `regedit`. Some changes may be, for example, certain file entries in `~/.wine/*.reg` and some files in the directory `~/.wine/drive_c/windows/system32`. Our Fake is (or should be!) prepared to be a Windows 98, the best to run our applications.

Now we will install some programs for the Fake to work without any problems.

The first step is to install DCOM98. In the WineToolsBase *Setup* menu, select the option *DCOM98* and click the *OK* button. You should see the message *Downloading*.

When the download is completed, proceed answer YES to the question: *OK to install DCOM98 for Windows 98?*, YES to the licensing terms (only if you agree of course!) and wait for the installation to finish.

In the next step we will install the Microsoft foundation Classes 4.x. In the same menu (Base Setup), select the option Microsoft Foundation Classes 4.x, confirm, and wait for the download to finish. Two DLL files will be installed and then you should return to the *Base Setup* menu.

To continue we now need to perform a more complex installation, namely the Internet Explorer. In the *Base Setup* menu, select the option *Internet Explorer 6.0 SP1 English* and click the *OK* button. You will see the information displayed in Illustration 5. Click *OK* to begin downloading the installer. During the installation process the download screen may be displayed too.

Illustration 5: Information during the Internet Explorer Installation

The download of the installer is fast, and when finished the installation starts automatically and begins to download the components. This process may take a long time, depending upon the power of your internet link (as much as 30 minutes or more). The following will be downloaded: Internet Explorer, Outlook Express, Windows Media Player, Macromedia Flash Player and the codecs, support for images files and VB Scripts. During the installation process of the Internet Explorer messages, such as the one displayed in illustration 6, may occur. If this happens simply click the *OK* button to continue.

Illustration 6: Information during the Internet Explorer Installation

The downloaded components are subsequently installed automatically. When finished, the installation displays a message informing you that WineTools has copied some scripts into the directory bin. Click the *OK* button, and answer *NO* to the next question: *Do you want to save the downloaded files for later?*

Following this you are returned to the *Base Setup* menu.

We have now finished the installation in the *Base Setup* menu. So return to the *Main* Menu, the principal WineTools menu, and select the option *Install Windows system software* to access the menu *System Software*, displayed in Illustration 7:

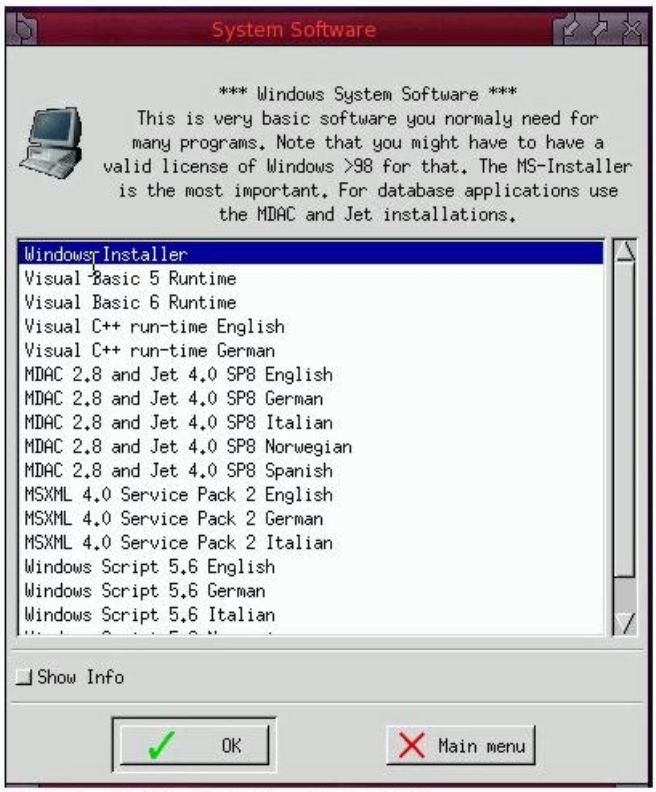


Illustration 7: WineTools Menu System Software

Note: You need some previous know-how regarding the installation process of WineTools. When you are in the *System Software* menu, select the following options and follow the instructions displayed adjacently:

Windows Installer: Click the *OK* button, and then *YES* in the `***WARNING***` message displayed and continue. When the message confirming completion of the installation process appears, click the *OK* button and wait for the WineTools *System Software* menu to be displayed.

Visual Basic 6 Runtime: Click the *OK* button, wait for the download and when the message *Would you like to install the Visual Basic 6.0 run time files?* appears, confirm (*YES*) and wait for the installation process to complete and then return to the *System Software* menu.

Visual C++ run-time English: Click the *OK* button, wait for the download to complete, accept the license terms and confirm *YES* to the question: *Do you want to restart your computer now?*. Wait for Wine to reboot, and the *System Software* menu to be displayed.

MDAC 2.8 and Jet 4.0 SP8 English: Click *OK* for the *Hint* exposed before the installation of MDAC 2.8, and wait for the download to complete. When the installation starts accept the license and proceed through the installation, selecting *Next*, *Next* and *Finish*. After the installation of Jet 4.0 SP8, follow the same steps for the installation of the MDAC 2.8.

Now let's install the Microsoft TrueType core fonts. Go to the WineTools *Base Setup* menu, select the option *True Type Font Arial*, click *OK*, wait for the download to complete, accept the license terms and finish the installation of this font family. After this, select the option *Install Microsoft TrueType core fonts*, the third option in the WineTools *Main* Menu, and proceed with the installation of all available fonts in the same way.

Ready! The installation is complete!

Preparing the Desktop

Now we will prepare our Desktop, "casting" IBExpert and the programs in our Fake, and the Firebird´s Client DLLs too, preparing our database in the Linux environment. You must have a disk partition with Windows and IBExpert installed, as well as the Firebird client on the same computer.

Before proceeding further we still have a detail to solve, namely the Wine date format used for programs. When I first began to "play" with Wine I found it hard to understand, because Wine manages date fields with the format `d/m/aaaa`. To use the date format `dd/mm/aaaa` we need to make some changes; this information was kindly provided by my friend Hamacker.

Logged in as a common user (this detail is very important) – the user you were logged-in as during the installation, open a shell and type the command:

```
[reddevil@hades reddevil]$ wine regedit [Enter]
```

This opens the *Windows Register Editor*. Search for the contents in the `HKEY_CURRENT_USER\Control Panel\International key`, and change the value of `sShortDate` from `d/M/yyyy` to `dd/MM/yyyy`.

Our fake windows (`~/.wine`) could be copied at this stage to another computer,
where, after the permissions have been changed, another user could use it, without having to reinstall all the software.

Note: The next steps need be done by a user with root rights. I prefer the KDE environment, because of the copy and paste facility for the instructions in *Konsole* with [Shift] + [Insert].

For the next example, I have Windows 98 installed in a FAT32 partition in the same hard drive on the same computer and I will mount this partition in the Windows flavor in the `/winhd` directory. This way I can show how to proceed with copying the necessary files from that Windows partition to the Linux partition where we are preparing the Wine Desktop. Open a shell and type the instructions (if necessary, change the instructions to adapt to your Windows):

```
 [reddevil@hades reddevil]$ su [Enter]
 Password: [root´s password] [Enter]

 [root@hades reddevil]# mount /dev/hda1 /winhd [Enter]

 [root@hades reddevil]# mkdir /home/reddevil/.wine/drive_c/programs [Enter]

 [root@hades reddevil]# mkdir /home/dados [Enter]
```

Now we will copy the necessary files to Linux. To make the Firebird connection we need a library to connect. If you are not sure about the software or component necessary, use only the `fbclient.dll` and copy this file. If you are still not sure, copy `gds32.dll` too to Wine's `system32` directory. Be careful with lower-case: all names and extensions must be lower-case. Hamacker says Wine does not accept any mismatches.

The list 1 (below) displays the sequence of instructions used for copying the files and creating the rights:

```
 [reddevil@hades reddevil]$ su [Enter]
 Password: [root´s password] [Enter]

 [root@hades reddevil]# mount /dev/hda1 /winhd [Enter]

 [root@hades reddevil]# mkdir /home/reddevil/.wine/drive_c/programs [Enter]

 [root@hades reddevil]# mkdir /home/dados [Enter]

 [root@hades reddevil]# cp -r /winhd/Arquivos\ de\ programas/HK-Software/IBExpert/
/home/reddevil/.wine/drive_c/ [Enter]

 [root@hades reddevil]# cp /winhd/Arquivos\ de\ programas/Firebird/Firebird_1_5/bin/fbclient.dll /home/reddevil/.wine/drive_c/windows
/system/fbclient.dll [Enter]

 [root@hades reddevil]# cp /winhd/Arquivos\ de\ programas/Firebird/Firebird_1_5/bin/fbclient.dll /home/reddevil/.wine/drive_c/windows
/system/gds32.dll [Enter]

 [root@hades reddevil]# cp -ax /winhd/myhome/dragonegg.fbk /home/dados/ [Enter]

 [root@hades reddevil]# cp -ax /winhd/myhome/*.exe /home/reddevil/.wine/drive_c/programs/ [Enter]

 [root@hades reddevil]# cd /home/ [Enter]

 [root@hades home]# chown -R firebird.firebird dados/ [Enter]

 [root@hades home]# cd /home/reddevil/.wine/drive_c/ [Enter]

 [root@hades drive_c]# chown -R reddevil.firebird IBExpert/ [Enter]

 [root@hades drive_c]# chown -R reddevil.firebird programs/ [Enter]

 [root@hades drive_c]# chown reddevil.firebird windows/system/fbclient.dll [Enter]

 [root@hades drive_c]# chown reddevil.firebird windows/system/gds32.dll [Enter]

 [root@hades drive_c]# umount /winhd/ [Enter]

 [root@hades drive_c]# exit [Enter]
```

List 1: Instructions for copying and granting permissions (rights)

The exit command (above) closes our root `Access` and turn us into a common user in the group `firebird`. And this user will restore the back up of a database from the Windows partition in Linux. In the example, the database name is `dragonegg.fdb`:

```
[reddevil@hades reddevil]$ cd /opt/firebird/bin/ [Enter]

 [reddevil@hades bin]$ ./gbak -user sysdba -password senha_do_sysdba -C -V -Z -R -P
4096 /home/dados/dragonegg.fbk /home/dados/dragonegg.fdb [Enter]
 .
 .[instructions for restore]
 .
 gbak: finishing, closing, and going home
 [reddevil@hades bin]$ exit [Enter]
```

If you haven't installed IBExpert yet, then run it now.

Now let´s configure IBExpert to run in Wine, principally using the Multiple Document Interface (MDI):

So proceed with the following steps used to run WineTools and call the `Winecfg`, or, in the KDE environment, press [Alt] + [F2], or use the `fbrun` in the fluxbox, or open a shell and type `winecfg` (in lowercase).

In the *Wine Configuration*, as seen in Illustration 8, on the *Application* page, select the option *Add application*, and in then in the form which opens, *Select a Windows executable file*, select the path of wine's virtualized windows environment until you reach the IBExpert folder, select `ibexpert.exe` and click *Open*. Then go to the *Graphics* page and uncheck the option *Allow* the windows manager to control the windows, check the option *Emulate a virtual desktop*, specify the Desktop size: `795 x 550`, as seen in Illustration 9, and confirm with *Apply* and *OK*.

Alternatively type:

```
$ env WINEPREFIX="/home/reddevil/.win" wine "C:\Program Files\HK-Software\IBExpert\ibexpert.exe"
```

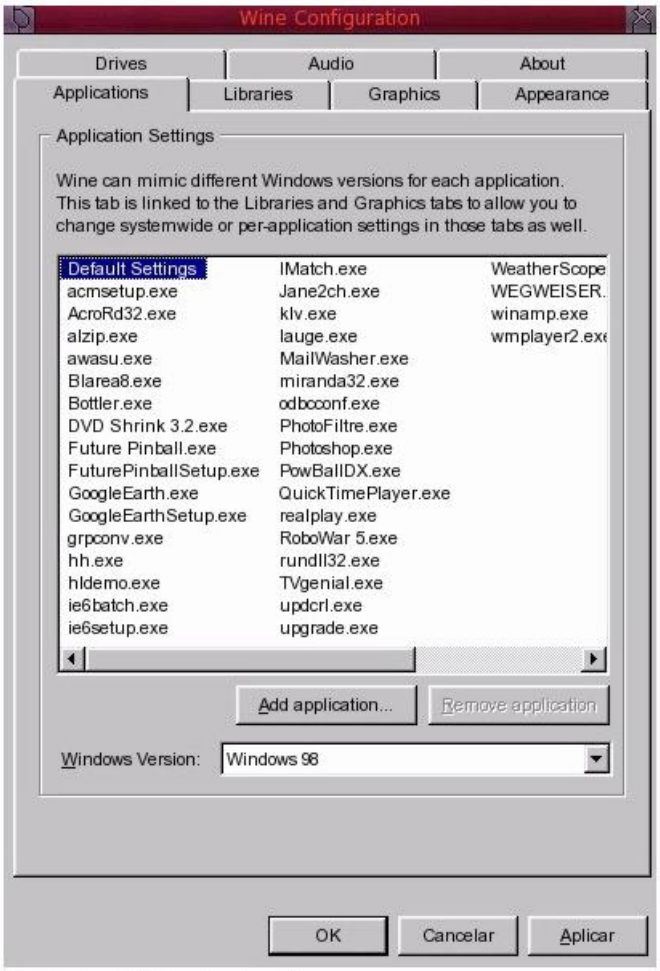


Illustration 8: Wine Configuration

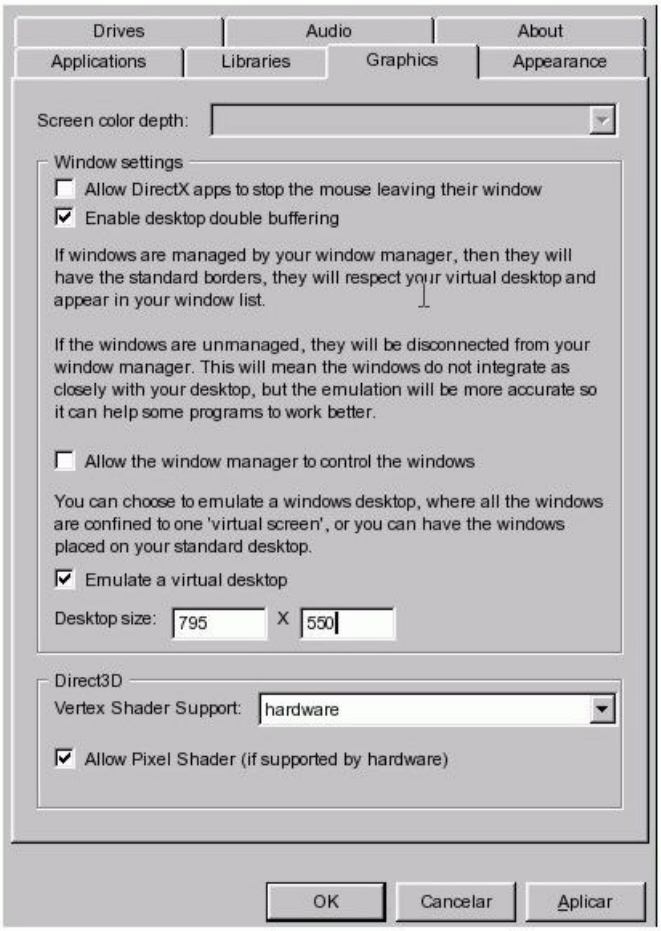


Illustration 9: Adding an application in WineCfg

The same procedure can also be used for applications developed in Delphi.

With screens with resolutions of 800x600 I got a good result for IBExpert with Desktop in 795x550. Try different configurations with your screen width.

Now open a shell and type:

```
[reddevil@hades reddevil]$ wine ~/.wine/drive_c/IBExpert/ibexpert.exe [Enter]
```

This command runs IBExpert in the virtualized Windows. The use of IBExpert will not be reported in this article. It is extremely easy and intuitive. However Wine implements an "exception" case: database access must always be made using TCP/IP. So it is important to use the TCP/IP protocol and localhost to identify the database. An example of IBExpert running in Linux KDE is displayed in Illustration 10.

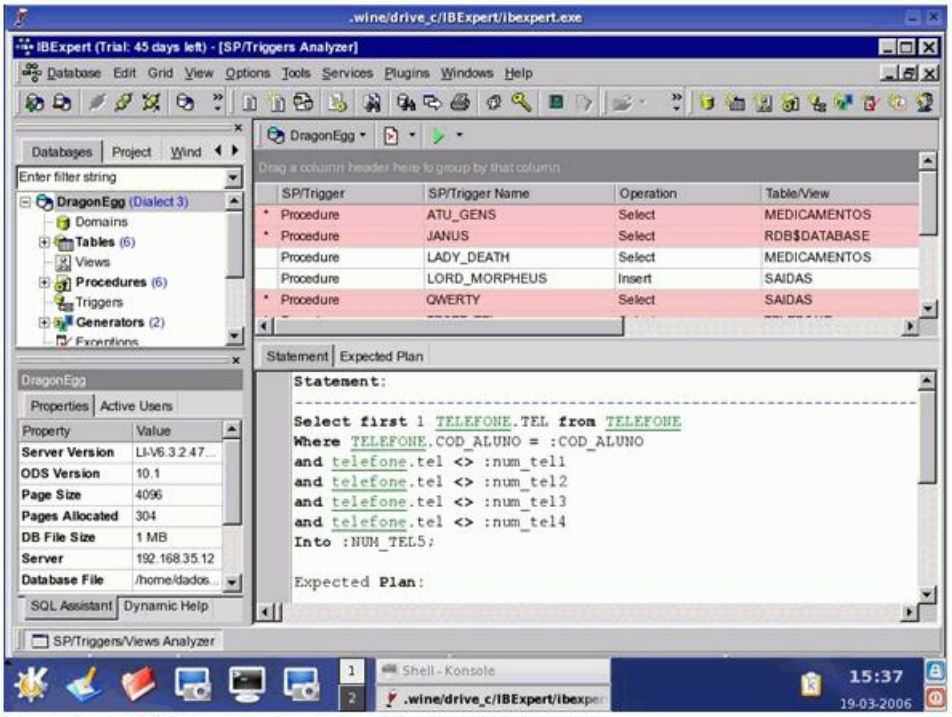


Illustration 10: IBExpert running in the virtualized Windows

In the case of Delphi applications, IBObjects users have an advantage, because they simply need to configure the component `TIB_Connection` and then compile the applications to run in Fake. Applications using dbExpress must have the DLLs exported for Wine.

Using IBObjects, I configure `TIB_Connection` changing the properties `CharSet`, `DatabaseName` (including the IP and path of the database, for example: `192.168.35.12:/home/dados/dragonegg.fdb`), `Username` and `Password`.

When we install the WineTools MDAC 2.8 and Jet 4.0, there are other components installed in Fake, such as ADO, support for MSSQL Server, etc. So if you use something other than IBObjects to access Firebird you need to configure it in Wine.

And yet another problem has been detected. Some applications developed in Delphi display problems with the position of *Buttons* and *Panels* (position and sizes), as can be seen in `192.168.35.12:/home/dados/dragonegg.fdb`), `Username` and `Password`.

`[reddevil@hades reddevil]$ wine ~/.wine/drive_c/programs/ohades.exe` [Enter]

Illustration 11: Form with Font MS Sans Serif

After a lot of hard work, in a moment of insanity (I live constantly on the borders of insanity and geniality :-)), I discovered the font used in the forms is *MS Sans Serif* (default for Delphi programs), and by changing the font to *Arial* the problem was solved, as displayed in Illustration 12. There is a program called `ohades.exe`, which you can not see in the illustration:

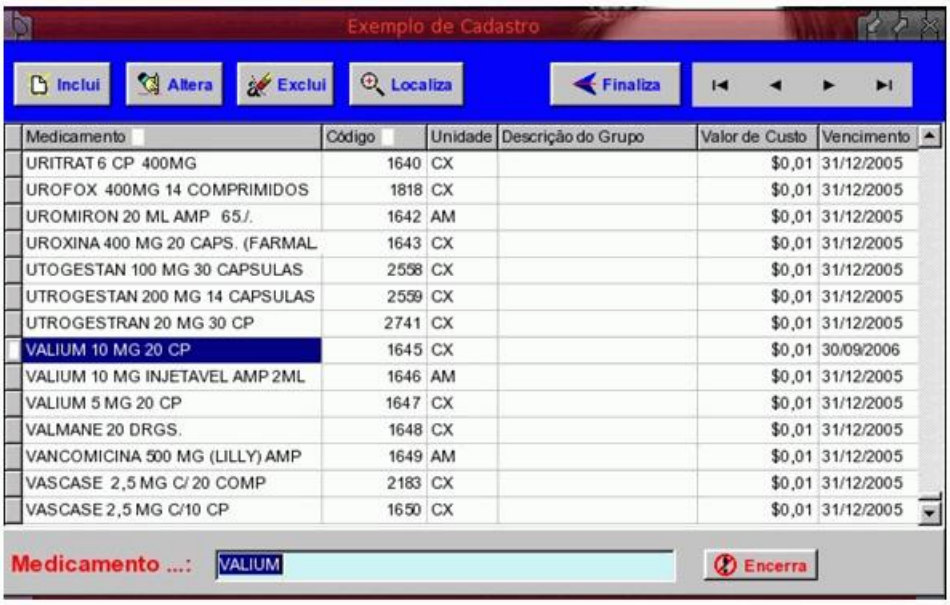


Illustration 12: Form with font Arial

As you can see, we can call the application from the command line in a shell. I did it because someone asked about the compatibility with Linux, and still using the Fluxbox like the Windows Manager, it uses only 700KB of RAM and not the 280MB used by KDE and Gnome. As it is normal to call applications from the Graphic Interface of a Windows Manager, it is necessary to create a link for the application in the KDE or call the application from an icon in the Fluxbox, after installing iDesk. The instruction to be used in the shortcut is displayed below, but be sure the `hades.exe` program is in the folder `~/.wine/drive_c/programs` of the Fake. There are 3 ways to call applications:

- `wine /home/reddevil/.wine/drive_c/programs/hades.exe` (UNIX mode)
- `wine C:\\programs\\hades.exe` (Wine mode)
- `wine "C:\programs\hades.exe"` ("DOS" mode)

Note: when using the last mode, the path of the program must be specified with quotation marks (double quotations).

Acknowledgements

Author's acknowledgements: I would like to give special thanks to Hamacker, who helped me too much and always gave me support I needed to enable me to finish this job.

We would like to thank the author, Luiz Paulo de O. Santos, for providing us with the English translation of this article, which was originally published in Linux World.

See also:
Installing on Linux

Bidirectional replication for InterBase and Firebird

The open source database server, Firebird, and its commercial partner, Borland InterBase, have long been established as a proven and stable platform for all sorts of database applications. Because of the common ancestry in the form of the InterBase 6 source code, many solutions can be implemented on both platforms without any problems. However Version 2 of the Firebird Server has recently set new standards, introducing many helpful functions that are unfortunately missing in InterBase 7.5. However a replication facility is not included in either platform. This article illustrates how a replication can be created with the aid of IBExpert.

What is replication?

The German-language Wikipedia offers a concise definition: "Replication is a duplication of data. The data base of the replicated data is, as a rule, identical with the original."

We need to distinguish between synchronous replication and asynchronous replication. Whilst the synchronous replication ensures that in the case of a fault or error, the database server can be immediately replaced by the replicated backup server and users can continue work without any disruption, an asynchronous replication makes sense when the databases and their servers are not always in the same network. An asynchronous replication is typically used for field staff and their laptops, or when branches of a company are not always connected to the main server by a dedicated line.

There are many further applications for replicated data, for example, a cluster can be constructed, by which multiple database servers can be interconnected to distribute the burden. Although there are various commercial suppliers offering replication solutions in the Firebird and InterBase world, a customized implementation has the advantage that it is possible to fulfil considerably more individual needs and wishes, at the same time saving license fees.

Fundamentals

The basis for a replicable database should always be a consequently constructed data model. The author's preferred solution is based on a primary key ID field, datatype `BIGINT` in every table and a consequent naming convention of foreign key fields in the form: `TABELLE_ID`. All primary keys are always created from a single generator.

This may initially appear somewhat unusual, but it does offer distinct advantages for replication and for any other subsequent extensions. Should existing databases need to be made replicable, existing tables can optionally be supplemented by a replication ID field or parallel tables filled using triggers. The mechanisms presented here are based on the preferred ID model with a common generator. All SQL commands are accommodated on the freely available Firebird Server. Necessary alterations for deployment on the InterBase server are explicitly mentioned.

In order to construct a replication, it is initially vital that absolutely all data alterations in the database are logged securely. Whilst other, supposedly transaction-safe database systems clearly produced gaps when rolling back, the Firebird and InterBase server are always transaction safe even in the case of trigger operations. Therefore corresponding triggers are created for existing tables, which log all insert, update and delete operations on each table.

The log is written in the following table:

```
CREATE TABLE IBE$LOG (
    ID   BIGINT NOT NULL PRIMARY KEY,
    USR  VARCHAR(30) default current_user,
    TS   TIMESTAMP default current_timestamp,
    SQL  VARCHAR(32000),
    IDX  BIGINT,
    DAT  BLOB SUB_TYPE 0 SEGMENT SIZE 16384
);
```

Although it is not always recommendable to use very large `VARCHAR` fields, this simplifies the model presented here. An autoincrement trigger can be created using IBExpert for the ID field, the value of which should be fetched from a generator, called `ID`. `USR` and `TS` are automatically filled in with the user name and timestamp. The complete SQL source code is stored in the SQL field, which will execute the identical insert, update and delete operations. This will be later exchanged between the databases concerned as part of the replication, and executed on the replicated system. The `IDX` field is designed to be an auxiliary field for the associated primary key. This can later be used to easily ascertain the history of a data set with the ID 123. Altered blob data is stored by means of special triggers for the replication in the `DAT` field.

To avoid global conflict of allocated primary keys, all ID generators are set at different start values on all servers concerned; Server A starts at 1 billion, Server B at 2 billion etc. As generators return a 64 Bit value, 16 billion participating replication servers could each generate 1 billion globally unique IDs without any conflict. Alternatively the offset between the IDs on each server can of course be increased accordingly by reducing the number of replication servers involved. The author considers the popular alternative method based on GUIDs disadvantageous, because the ID method can also be used for other solutions, for example, that data may only be altered on the server where it was created.

Transaction Log

It is wise to automate trigger creation, so as to be armed for later data model alterations. Due to the commands available in Firebird, it is possible to do this within a stored procedure. The absence of the `EXECUTE STATEMENT` command in InterBase means that the source code needs to be executed using IBExpert's IBEBlock technology, as this method enables the InterBase server to handle such language elements.

The `INITLOG` procedure initially begins with a loop, extracting all table names from the system table, `RDB$RELATIONS`, which do not contain the dollar sign:

```
select f_rtrim(rdb$relation_name) from rdb$relations
where rdb$relation_name  not containing '$'
  INTO :V$RELATION_NAME
```

Then the source code for the first AFTER INSERT trigger for the first table found begins in the following statement:

```
sql='RECREATE TRIGGER IBE$'||V$RELATION_NAME||'_AI FOR '||V$RELATION_NAME||' '||f_crlf()||
    'ACTIVE AFTER INSERT POSITION 32000 '||f_crlf()||
    'AS '||f_crlf()||
    'declare variable sql varchar(32000); '||f_crlf()||
    'begin '||f_crlf()||
    '  SQL=''INSERT INTO '||V$RELATION_NAME||'(';
```

Using the `f_crlf` UDF, from the FreeAdhocUDF library, a line feed is inserted into the trigger source code, without which the trigger would function, but nevertheless be extremely confusing.

In the following loop all fields in the current table are selected from the `RDB$RELATION_FIELDS` and `RDB$FIELDS` tables, whose type does not equal 261. Type 261 is for blob fields, which need to be treated separately later on.

```
  komma='';
  for select f_rtrim(rdb$relation_fields.rdb$field_name)
  from rdb$relation_fields
  join rdb$fields on   rdb$relation_fields.rdb$field_source=rdb$fields.rdb$field_name and

 rdb$fields.rdb$field_type<>261
  where rdb$relation_name=:v$relation_name
  into :v$field_name
  do
  begin
   sql=sql||komma||v$field_name;
   komma=',';
  end
  sql=sql||') values (';
  komma='';
```

A comma-separated list of all field names is generated due to the previously empty variable and the comma variable defined in the loop, as required for an `INSERT` command. Then another sweep is made through the field list, in which the instance variable `NEW.` is prepared with the appropriate exclamation marks for the second part of the trigger source code. This part, due to lack of space here, can be found in the sample script.

This is now followed by the command to write the SQL command out of the trigger into the table `IBE$LOG`. With the subsequent request using the command, `EXECUTE STATEMENT :SQL`, the trigger source code is executed from the procedure, so creating the trigger.

```
sql=sql||')'';'||f_crlf()||
    '  insert into ibe$log(sql,idx) values (:sql,new.id);'||f_crlf()||
    'end;';
execute statement :sql;
```

Blob data

In the subsequent parts of the script, the update and delete triggers are constructed and generated in a similar way. Finally extra triggers are then created for each blob field, because only data should be logged which has actually been altered. For this purpose all field and table names with the type 261 are selected.

```
FOR
  select
  f_rtrim(rdb$relation_fields.rdb$relation_name),f_rtrim(rdb$relation_fields.rdb$field_name)
  from rdb$relation_fields
  join rdb$fields on rdb$relation_fields.rdb$field_source=rdb$fields.rdb$field_name
  where rdb$relation_fields.rdb$relation_name not containing '$'
  and rdb$fields.rdb$field_type=261
  INTO :V$RELATION_NAME, :V$FIELD_NAME
DO
BEGIN
  sql='RECREATE TRIGGER IBE$'||V$RELATION_NAME||V$FIELD_NAME||'_AI FOR '||V$RELATION_NAME||'
  '||f_crlf()||
  'ACTIVE AFTER INSERT POSITION 32000 '||f_crlf()||
  'AS '||f_crlf()||
  'begin '||f_crlf()||
  '  if (new.'||V$FIELD_NAME||' is not null) then  insert into ibe$log(sql,idx,dat) values
  ('''||V$RELATION_NAME||'.'||V$FIELD_NAME||''',new.id,new.'||V$FIELD_NAME||');'||f_crlf()||
  'end;';
  execute statement :sql;
  ....
```

The transaction log can now be activated in the database by executing the Firebird procedure `INITLOG` or in InterBase using the appropriate IBEBlock command. If data model alterations are to be made, it is wise to first deactivate this transaction log, as this way all references to the tables used will be deleted again. To this effect, the `DROPLOG` procedure is implemented in the sample script.

Replicating the transaction log

The actual replication, i.e. the data exchange from the transaction log in the correct order, now begins with an IBEBlock. An IBEBlock is a special extension within the IBExpert product family, which enables additional commands for the handling of scripts. An IBEBlock also offers commands for InterBase, which are not otherwise possible within a procedure, for example, the `EXECUTE STATEMENT` command. Furthermore it is possible to make a connection to multiple databases in an IBEBlock script. Replication can also optionally be carried out with all ODBC databases using the integrated ODBC port. Such IBEBlock commands may also be fully incorporated into your own applications using the DLL or EXE distribution licenses.

IBEBlock first makes the connections to the databases involved:

```
execute ibeblock
 as
 begin
  create connection src dbname 'localhost:c:\src.fdb'
  password 'repl' user 'REPL'
  clientlib 'fbclient.dll';

  create connection dest dbname 'localhost:c:\dest.fdb'
  password 'repl' user 'REPL'
  clientlib 'fbclient.dll';
```

After the connections have been made it is possible to switch backwards and forwards between any of the databases, using the `USE` command. The following loop now selects all entries in the `IBE$LOG` table in the source or reference database and inserts them into the `IBE$LOG` table in the target database. In order to avoid re-replicating data that has already been transferred, a table, in this example `IBE$TRANS`, is referenced, in which the ID from `IBE$LOG` is entered following successful data transmission. The user `REPL` was used for the replication, because this way it is possible to recognize which data have come via the replication and therefore do not need to be replicated back again.

```
 use src;
 for select id, usr, ts, sql, idx, dat
 from ibe$log where usr<>'REPL'
 and not exists (select ibe$trans.id from ibe$trans where ibe$trans.id=ibe$log.id)
 into :id, :usr, :ts, :sql, :idx, :dat
 do
 begin
   use dest;
   insert into ibe$log(id, ts, sql, idx, dat)
   values (:id, :ts, :sql, :idx, :dat);
   if (sql not starting with 'BLOB ') then  execute statement :sql;
   commit;
   use src;
   insert into ibe$trans(id) values (:id);
   commit;
 end
```

The approach to be taken when replicating blob data can be found in the sample script. This also demonstrates the procedure for bidirectional replication. Using this technology little effort is needed to supplement a system, which is capable of exchanging data for asynchronous replication using packed blob data and is sufficient for large data quantities, even when low band widths are used. It is also possible on a quick backbone to construct an extremely rapid and reliable database cluster using the InterBase/Firebird Event Alerter technology.

The customizable scripts can be implemented for partial replication, by using any number and combination of rules. This way it is possible to distribute data quantities to various servers according to logical criteria. For example, the customer base can be distributed to all servers, whilst the order data is only copied to country-specific databases or servers. Or the inverse direction can be used to combine and consolidate data from multiple databases.

Database corruption

The following articles provide important information regarding the causes leading to database corruption, as well as ways to recover a corrupt database. We would like to thank the authors for allowing us to publish their articles here.

How to corrupt a database

Although Firebird is extremely stable and secure, there are a few things that you should *NOT* do, as these could result in corrupting the database!

The following tips have been taken from the *Firebird Quick Start Guide, © IBPhoenix Publications 2002, 2003*. Many thanks to Paul Beach (http://www.ibphoenix.com)!

Modifying metadata tables

Firebird stores and maintains all of the metadata for its own and your user-defined objects in a Firebird database! More precisely, it stores them in relations (tables) right in the database itself. The identifiers for the system tables, their columns and several other types of system objects begin with the characters 'RDB$'.

Because these are ordinary database objects, they can be queried and manipulated just like your user-defined objects. However, just because you can does not say you should. The Firebird engine implements a high-level subset of SQL (DDL - please refer to Data Definition Language for further information) for the purpose of defining and operating on metadata objects, typically through CREATE, ALTER and DROP statements.

It cannot be recommended too strongly that you use DDL - not direct SQL operations on the system tables - whenever you need to alter or remove metadata. Defer the 'hot fix' stuff until your skills in SQL and your knowledge of the Firebird engine become very advanced. A wrecked database is neither pretty to behold nor cheap to repair.

Disabling forced writes

Firebird is installed with forced writes (synchronous writes) enabled by default. Changed and new data are written to disk immediately upon posting.

It is possible to configure a database to use asynchronous data writes - whereby modified or new data are held in the memory cache for periodic flushing to disk by the operating system's I/O subsystem. The common term for this configuration is forced writes off (or disabled). It is sometimes resorted to in order to improve performance during large batch operations.

The big warning here is - *do not disable forced writes on a Windows server*. It has been observed that the Windows server platforms do not flush the write cache until the Firebird service is shut down. Apart from power interruptions, there is just too much that can go wrong on a Windows server. If it should hang, the I/O system goes out of reach and your users' work will be lost in the process of rebooting.

- Windows 9x and ME do not support deferred data writes.

Disabling Forced Writes on a Linux server

Linux servers are safer for running an operation with forced writes disabled temporarily. Do not leave it disabled once your large batch task is completed, unless you have a very robust fall-back power system.

Restoring a backup to a running database

One of the restore options in the GBAK utility (`gbak -r[estore]`) allows you to restore a `gbak` file over the top of an existing database. It is possible for this style of restore to proceed without warning while users are logged in to the database. Database corruption is almost certain to be the result.

Be aware that you will need to design your Admin tools and procedures to prevent any possibility for any user (including SYSDBA) to restore to your active database if any users are logged in. If is practicable to do so, it is recommended to restore to spare disk space using the `gbak -c[reate]` option and *test the restored database* using `isql` [or IBExpert]. If the restored database is good, shut down the server. Make a file system copy of the old database and then copy the restored database file (or files) over their existing counterparts.

Allowing users to log in during a restore

If you do not block access to users while performing a restore using `gbak -r[estore]` then users may be able to log in and attempt to do operations on data. Corrupted structures will result.

Recovering corrupt databases

The following is an excerpt from the successful Russian book, "The InterBase World" first published in September 2002, with a second edition following in April 2003. The authors of the book are Alexey Kovyazin, developer of IBSurgeon (http://www.ibsurgeon.com) and well-known Russian InterBase specialist, and Serg Vostrikov, CEO of the Devrace company (http://www.devrace.com).

Here the authors would like to offer you a draft copy of one chapter of this book devoted to recovery of InterBase/Firebird databases.

They would like to pass on their thanks to all who helped create this guide: Craig Stuntz, Alexander Nevsky, Konstantin Sipachev, Tatjana Sipacheva and all the other kind and knowledgeable members of the InterBase and Firebird community.

Main causes of database corruption

Unfortunately there is always a probability that any information stored will be corrupted and some of this information will be lost. Databases are not an exception to this rule. In this chapter we will consider the principal causes that lead to InterBase/Firebird database corruption, some methods of repairing databases and extracting information from them. We will also make recommendations and offer precautions that will minimize the probability of information loss.

First of all, if we speak about database repair we should perhaps first define "database corruption". A database is usually described as damaged if, when trying to extract or modify some information, errors appear and/or the information to be extracted turns out to be lost, incomplete or incorrect. There are cases when database corruption is hidden and can only be found by testing with special facilities. However there are also real database corruptions, when it is impossible to connect to the database, when adjusted programs send strange errors to the clients (without any data manipulation having occurred), or when it is impossible to restore the database from a backup copy.

Principal causes of database corruption are:

1. Abnormal termination of the server computer, especially an electrical power interruption. For the IT-industry it can be a real blow and that is why we hope there is no need to remind you once again about the necessity of having a source of uninterrupted power supply on your server.
2. Defects and faults on the server computer, especially the HDD (hard disk drive), disk controllers, the computer's main memory and the cache memory of Raid controllers.
3. An incorrect connection string to a multi-client database with one or more users (in versions prior to 6.x). When connecting via TCP/IP, the path to the database must be pointed to a server name: `drive:/path/databasename/`
   For servers on UNIX platforms: `servername: /path/databasename/`
   Using a NetBEUI protocol: `servernamedrive:pathdatabasename`.
   Even when connecting to a database from the computer, on which the database is located and where the server is running, the same specification should be used, renaming the servername as localhost. It is not possible to use mapped drives in the connection specification. If you break one of these rules, the server thinks that it is working with different databases and database corruption is guaranteed.
4. File copy or other file access to the database when the server is running. The execution of the command `shutdown`, or disconnecting the users in the usual way is not a guarantee that the server is doing nothing with the database. If the sweep interval is not set to 0, garbage collection may be being executed. Generally the garbage collection is executed immediately after the last user disconnects from the database. Usually it takes several seconds, but if many `DELETE` or `UPDATE` operations were committed before it, the process may take longer.
5. Using unstable InterBase server versions 5.1-5.5. The Borland Company officially admitted that there were several errors in these servers and these were removed in the stable upgrade 5.6 only after the release of certified InterBase 6 was in free-running mode for all clients of servers 5.1-5.5 on its site.
6. Exceeding size restriction of a database file. At the time of writing this, for most existing UNIX platform servers the limit is 2 GB, for Windows NT/2000 - 4 GB, but it is recommended to assume 2 GB. When the database size is approaching its limit, an additional file must be created.
7. Exhaustion of free disk space when working with the database.
8. For Borland InterBase servers using versions under 6.0.1.6 - exceeding the restriction of the maximum number of generators, according to Borland InterBase R & D defined as follows (please refer to table 1 below).

Critical number of generators in early InterBase versions				
Version	**Page size=1024**	**Page size=2048**	**Page size=4096**	**Page size=8192**
Pre 6	248	504	1016	2040
6.0.x	124	257	508	1020

For all Borland InterBase servers - exceeding the permissible number of transactions without executing a backup/restore. The number of transactions that have been made in the database since the last backup and restore can be determined by invoking the utility `GSTAT` with the key `-h` parameter `NEXT TRANSACTION ID`.

According to Ann W. Harrison, the critical number of transactions depends on the page size, and has the following values (please refer to table 2 below):

Critical number of transactions in Borland InterBase servers	
Database page size	**Critical number of transactions**
1024 byte	131 596 287
2048 byte	265 814 016
4096 byte	534 249 472
8192 byte	1 071 120 384

The constraints of Borland InterBase servers enumerated above are not applicable to Firebird servers except for the earliest versions 0.x, the existence of which has already become history. If you use the final version Firebird 1.0 or above, or InterBase 6.5-7.x, you should not worry about points 5, 6, 8 and 9 and should instead concentrate your efforts on other causes. Now we will consider the most frequent of these in detail.

Power supply failure

When shutting off the power on the server, all data processing activities are interrupted in the most unexpected and (according to Murphy's law) dangerous places. As a result the information in the database may be distorted or lost. The simplest case is when all uncommitted the data from a client's applications are lost as a result of an emergency server shutdown. After a power-cut restart the server. This analyzes data, makes a note of incomplete transactions related to none of the clients, and cancels all modifications made within the bounds of these «dead» transactions. Actually such behavior is normal and assumed from the start by InterBase developers.

However power supply interruption is not always followed just by such insignificant losses. If the server was executing a database extension at the moment of power supply interruption, there is a large probability of orphan pages present in the database file (pages that are physically allocated and registered on the page inventory page (PIP), upon which it is however impossible to write data).

Only GFIX, the repair and modification tool (we will consider it further on), is able to combat orphan pages in the database file. Actually orphan pages lead to unnecessary use of disk space and, as such, are not the cause of data loss or corruption. Power loss leads to more serious damages. For example, after shutting off the power and restarting, a great amount of data, including committed data, may be lost (after adding or modification of which the command «commit transaction» was executed). This happens because confirmed data is not written immediately to the database file on disk. The file cache of the operating system (OS) is used for this purpose. The server process gives the data write command to the OS. Then the OS assures the server that all the data has been saved to disk although in reality the data is initially stored in the file cache. The OS doesn't hurry to save this data to disk, because it assumes that there is a lot of main memory left, and therefore delays the slow operation of writing to disk until the main memory is full. Please refer to the next subject - Forced Writes - cuts both ways - for further information.

Forced writes - cuts both ways

In order to influence this situation, tuning of the data write mode is provided in InterBase 6 and Firebird. This parameter is called FORCED WRITES (FW) and has 2 modes - ON (synchronous) and OFF (asynchronous). FW modes define how InterBase/Firebird communicates with the disk. If FW is turned on, the setting of synchronous writes to disk is switched on, and confirmed data is written to disk immediately following the COMMIT command, the server waits for writing completion and only then continues processing. If FW is switched off InterBase doesn't hurry to write data to disk after a transaction is committed, and delegates this task to a parallel thread, while the main thread continues data processing, not waiting until all writes are written to disk.

Synchronous writes mode is one of the most careful options and it minimizes any possible data loss. However it may cause some loss of performance. Asynchronous writes mode increases the probability of loss of a great quantity of data. In order to achieve maximum performance FW OFF mode is usually set. But as a result of power interruption a much higher quantity of data is lost using the asynchronous writes mode than when using the synchronous mode. When setting the write mode you should decide whether a few percentage points of performance are more significant than a few hours of work should power be interrupted unexpectedly.

Very often users are careless with InterBase. Small organizations save on any trifle, often on the computer server, where the DBMS server and different server programs (not only server) are installed and running as well. If they hang-up people don't think for long, and simply press *RESET* (it happens several times a day). Although InterBase is very stable with regard to such activities compared with other DBMS, and allows work with the database to start immediately after an emergency reboot, such a procedure is not recommended. The number of orphan pages increases and data lose connections among themselves as a result of faulty reboots. It may still function and continue for a long time, but sooner or later it will come to an end. When damaged pages appear among PIP or generator pages, or if the database header page is corrupted, the database may never open again and become a big chunk of separate data from which it is impossible to extract a single byte of useful information.

Corruption of the hard disk

Hard disk corruptions lead to the loss of important database system pages and/or the corruption of links among the remaining pages. Such corruptions are one of the most difficult cases, because they almost always require low-level interference to restore the database.

Database design mistakes

It is necessary to learn of some mistakes made by database developers that can lead to an impossible database recovery from a backup copy (*.gbk files created by the GBAK program). First of all a careless use of constraints at database level. A typical example is the constraint NOT NULL. Let's suppose that we have a table filled with a number of records. Now using the ALTER TABLE command we'll add one more column to this table and specify that it mustn't contain the non-defined value NULL. Something like this:

```
ALTER TABLE sometable Field/INTEGER NOT NULL
```

In this case there will be no server error as should be expected. This metadata modification will be committed and we won't receive any error or warning message, which creates an illusion of normality.

However, if we backup the database and try to restore it from the backup copy, we'll receive an error message at the phase of restoring (because NULLS are inserted into the column that has NOT NULL constraint, and the process of restoring will be interrupted. (An important note provided by Craig Stuntz: with version InterBase 7.1 constraints are ignored by default during a restore (this can be controlled by a command-line switch) and nearly any non-corrupt backup can be restored. It's always a good idea to do a test restore after performing a backup, but this problem should pretty much disappear in version 7.1.). This backup copy can't be restored. If the restore was directed to a file having the same name as the existing database (during restoration of the existing database the working file was being rewritten), we'll lose all information.

It has to do with the fact that NOT NULL constraints are implemented by system Triggers which check only incoming data. During restoration, data from the backup copy is inserted into the empty, newly created tables - here we can find inadmissible NULLS in the column with the constraint NOT NULL.

Some developers consider such InterBase behavior to be incorrect, but others will be unable to add a field with NOT NULL restriction to the database table.

The question about required value by default and filling with this value at the moment of creation was widely discussed by Firebird architects, but it wasn't accepted because of the fact that the programmer is obviously going to fill it according to an algorithm, which is rather complicated and maybe iterative. But there is no guarantee, whether he'll be able to distinguish the records ignored by previous iteration from unfilled records or not.

A similar problem can be caused by a garbage collection fault, caused by the specification of an incorrect path to the database (the cause of corruption 3) at the time of connection, and file access to database files when the server is working with it (the cause of corruption 4), and records wholly filled with NULLS can appear in some tables. It's very difficult to detect these records, because they don't correspond to integrity control restrictions, and operator SELECT just doesn't see them, although they get into the backup copy. If it is impossible to restore for this reason, the GFIX utility should be used (see below), to find and delete these records using non-indexed fields as search conditions. After this try to make a backup copy again and restore the database from it. In conclusion we can

say that there are a great number of causes of database corruption and you should always be prepared for the worst - that your database could become damaged for one reason or another. You should therefore be prepared at all times to restore and rescue valuable information.

Precautions and methods of repair

And now we shall consider precautions that guarantee Firebird/InterBase database security, as well as methods of repairing damaged databases.

Regular backups

In order to prevent database corruption, backup copies should be created regularly (if you want to know more about backup then please refer to Backup and Restore for further information). It's the most trusted method to prevent and combat database corruption. Only a backup gives 100% guarantee of database security. As described above, it is possible get a useless copy as the result of restoring a backup file (i.e. a copy that can't be restored); that's why restoring a base from the copy should not be performed by writing over the script, and a backup must be carried out according to definite rules. Firstly, a backup should be executed as often as possible, secondly it must be serial and thirdly, backup copies must be checked for their restoring capability.

Usually, a backup means that it's necessary to make a backup copy rather often, for example, once every twenty-four hours. The shorter the period is between database backups, the less data will be lost as a result of a fault. The sequence of backups means that the number of backups should increase and should be stored for at least a week. If possible, backups should be written to special devices such as a streamer, but if this is not possible - copy them to another computer. The history of backup copies will help to discover hidden corruptions and cope with an error that perhaps arose some time ago but has only just showed up unexpectedly. It is necessary to check whether it is possible to restore the saved backup without errors or not. This can be checked in only one way - through the test restore process. It should be mentioned that the restore process takes 3 times longer than the backup, and it's difficult to execute restore validation every day for large databases, because it may interrupt the users' work for a few hours (a night break may not be enough).

It would be better if big organizations didn't save at the wrong end and assigned one computer just for these purposes.

In this case, if the server must work with a serious load 24 hours 7 days a week, we can use the `SHADOW` mechanism for taking snapshots of the database, and performing further backup operations from the immediate copy. When creating a backup copy and then restoring the database from this backup, all data in the database is recreated. This process (backup/restore or b/r) contributes to the correction of most non-fatal errors in the database connected with hard disk corruptions, detecting problems with integrity in the database, cleaning the database of garbage (old versions and fragments of records, incomplete transactions) which decreases the database size considerably.

Regular backup/restore is a guarantee of Firebird/InterBase database security. If the database is working, then it is recommended to execute backup/restore on a weekly basis. To tell the truth, there are some examples of Firebird/InterBase databases that are intensively used for some years without a single backup/restore.

Nevertheless, to be on the safe side it's desirable to perform this procedure regularly, especially as it can be easily automated (please refer to Backup and Restore).

If it's impossible to perform a regular backup/restore for certain reasons, then the `GFIX` tool can be used for checking and restoring the database. `GFIX` allows you to check and remove many errors without performing a backup/restore.

Using GFIX

The command-line utility `GFIX` is used for checking and restoring databases. Furthermore `GFIX` can also execute various database control activities: changing the database dialect, setting and canceling the mode "read-only", setting cache size for a specific database and also some important functions.

`GFIX` is committed in command-line mode and has the following syntax:

```
Gfix [ options] db_name
```

Options is a set of options for executing `GFIX`, `db_name` is the name of the database for which the operations are to be performed, defined by a set of options. The following table displays the `GFIX` options related to database repair:

GFIX tool options for database restoration	
Option	**Description**
-f[ull]	This option is used in combination with -v and means it's time to check all fragments of records
-i[gnore]	Option makes GFIX ignore checksum errors at the time of validation or database cleaning
-m[end]	Marks damaged records as not available, as a result of which they will be deleted during the following backup/restore. This option is used when preparing a corrupted database for backup/restore.
-n[o_update]	Option is used in combination with -v for read-only database validation without correcting corruptions
-pas[sword]	Option allows the password to be set when connecting to the database. (Note that there is an error in the InterBase documentation: -pa[ssword], the shortcut "-pa" will not work - you need to use "-pas")
-user	Option allows the user's name to be set when connecting to the database
-v[alidate]	Option for presetting the database validation when errors are discovered
-m[ode]	Option for setting the write mode for the database - for read-only or read/write. This parameter can accept 2 values - read write or read only.
-w[rite] {**sync** | **async**}	Option that switches on and off the mode synchronous/ asynchronous forced writes to database. sync - to turn synchronous writes on (FW ON); async -to turn asynchronous writes on (FW OFF);

Here are some typical `GFIX` examples:

```
gfix -w sync -user SYSDBA -pass masterkey firstbase.gdb
```

In this example we set for our test database, `firstbase.gdb`, the synchronous writes mode (`FW ON`). (Of course, this is more useful before corruption occurs). And below is the first command that you should use to check the database after corruption has occurred:

```
gfix -v -full -user SYSDBA -pass masterkey firstbase.gdb
```

In this example we start checking our test database (option `-v`) and specify that fragments of records must be checked as well (option `-full`). Of course, it is more convenient to set various options for the checking and restoring process using IBExpert or another GUI interface, but we'll review the functions of database recovery using command-line tools. These tools are included in InterBase and Firebird and you can be sure that their behavior will be the same on all OS running InterBase. It is vital that they always be close to the server. Besides the existing tools, allowing you to execute database administration from a client's computer, you can use the Services API, which isn't supported by the InterBase server Classic architecture. That means you need to use a third party product (such as IBExpert or other administration tool) with the SuperServer architecture.

Repairing a corrupt database

Let's assume there are some errors in our database. Firstly, we have to check the existence of these errors; secondly, we have to try to correct these errors. We recommend the following procedure:

You should stop the InterBase server if it's still working and make a copy of the file or the database files. All the restore activities should only be performed with a database copy, because it may lead to an unsatisfactory result, and you'll have to restart the restore procedure (from a starting point). After creating a copy we'll perform the complete database validation (checking fragments of records).

We should execute the following command for this (or use the IBExpert Services menu item Database Validation):

```
gfix -v -full corruptbase.gdb -user SYSDBA -password
```

In this case `corruptbase.gdb` - is a copy of the damaged database. This command will check the database for any structural corruption and produce a list of unsolved problems. If such errors are detected, we'll have to delete the damaged data and get ready for a backup/restore using the following command (or using the IBExpert Services menu item Backup Database):

```
gfix -mend -user SYSDBA -password your_masterkey corruptbase.gdb
```

After committing this command you should check if there are any errors left in the database. Run `GFIX` using the options `-v -full`, and when the process is over, perform a database backup:

```
gbak -b -v -ig -user SYSDBA -password corruptbase.gdb corruptbase.gbk
```

This command performs a database backup (option `-b`) and we'll get detailed information about the backup process execution (option `-v`). Errors with regard to checksums will be ignored (option `-ig`).

Please refer to `GBAK` and Backup Database for further information.

If some errors are found during the backup, you should start it in another configuration:

```
gbak -b -v -ig -g -user SYSDBA -password
corruptbase.gdb corruptbase.gbk
```

Where option `-g` will switch off garbage collection during the backup. This often helps to solve backup problems.

Also it may be possible to make a backup of a database if it is set in the read-only mode beforehand. This mode prevents writing any modifications to the database and sometimes helps to complete the backup of a damaged database. For setting a database to read-only mode, you should use the following command (or the IBExpert Services menu item Database Properties):

```
gfix -m read_only -user SYSDBA -password masterkey
Disk:Pathfile.gdb
```

Following this, you should try to perform the database backup again using the parameters given above (or the IBExpert Services menu item Backup Database).

If the backup was completed successfully, you should restore the database from the backup copy, using the following command (or the IBExpert Services menu item Restore Database):

```
gbak -c -user SYSDBA -password masterkey Disk:Pathbackup.gbk
Disk:Pathnewbase.gdb
```

When you are restoring the database, you may come across some problems, especially when creating the indices.

In this case the `-inactive` and `-one_at_a_time` options should be added to the restore command. These options deactivate indices when creating from the database backup and commit data confirmation for each table. Alternatively use the IBExpert Services menu item Restore Database.

Extract data from a corrupt database

It is unfortunately possible that even the operations previously mentioned in this section do not lead to a successful database recovery.

It means that the database is seriously damaged or it cannot be restored as a single entity, or a huge effort must be made to recover it. For example, it is possible to execute a modification of system metadata, use non-documented functions and so on. It is very hard, time-consuming and ungrateful work with doubtful chances of success. If at all possible, try to evade it and use other methods. If a damaged database opens and allows you to perform reading and modification operations with some data, you should take advantage of this possibility and save the data by copying it to a new database, and say good-bye to the old one for good.

So, before transferring the data from the old database, it's necessary to create a new destination database. If the database hasn't been altered for a long time, you can use the old backup, from which metadata can be extracted for creating the new database. Based on these metadata it is necessary to create a data destination and start copying the data. The main task is to extract the data from the damaged database. Then we'll have to allocate the data in a new base, but that's not very difficult, even if we have to restore the database structure from memory.

When extracting data from tables, you should use the following algorithm of operations:

1. At first you should try to execute `SELECT *` from `table N`. If it ran normally you could save the data you've got in the external source. It's better to store data in a script (using the IBExpert Tools menu item Extract Metadata for example), as long as the table doesn't contain blob fields. If there are blob fields in the table, then this data should be saved to another database by a client program that will play the role of mediator.
2. If you failed to retrieve all data, you should delete all the indices and try again. In fact, indices can be deleted from all the tables from the beginning of the restore, because they won't be needed any more.
3. Of course, if you don't have a metadata structure which is the same as that of the corrupted database, it's necessary to input a protocol of all operations that you are doing with the damaged database source.

   If you cannot read all the data from the table after deleting the indices, try to execute a range query by primary key, i.e. select a definite range of data. For example:

   ```
   SELECT * FROM table N WHERE field_PK >=0 and field_PK <=10000
   ```

   `Field_PK` here is a primary key.

InterBase has page data organization and that's why a range query of values may be rather effective.

Nevertheless it works because we can expel data from the query from damaged pages and fortunately read the other ones. You may recall our thesis that there is no defined order of storing records in SQL.

Really, nobody can guarantee that an unordered query will, during restarts, return the records in the same order, but nevertheless the physical records are stored within the database in a defined internal order. It's obvious that the server will not mix the records purely to abide to SQL standards. Try to use this internal order when extracting data from a damaged database. Vitaliy Barmin, an experienced Russian InterBase developer reported that in this way he managed to restore up to 98% of information from an unrecoverable database (there were a great number of damaged pages). Thus, data from a damaged database must be moved to a new database or into external sources such SQL scripts. When you copy the data, pay attention to [Generator | generator]] values in the damaged database (they must be saved for restarting proper work in the new database. If you don't have a complete copy of the metadata, you should extract the texts of stored procedures, triggers, constraints and the definition of indices.

Restoring hopeless databases

In general, restoring a database can be very troublesome and difficult and that's why it's better to make a backup copy of the database and then restore the damaged data and whatever has happened, you shouldn't despair because a solution can be found even in the most difficult situations. And now we'll consider two cases.

The first case (a classic problem): A backup that can't be restored because of having `NULL` values in a column with `NOT NULL` constraints (the restore process was run over the working file). The working file was erased and the restore process was interrupted because of an error. And as a result of thoughtless actions

the result was a great amount of useless data (that can't be restored) instead of a backup copy. But a solution was found. The programmer managed to recollect which table and which column contained the constraint NOT NULL. The backup file was loaded to a hexadecimal editor. And a combination of bytes, corresponding to the definition of this column, was found by searching. After innumerous experiments it turned out that the constraint NOT NULL adds 1 somewhere near the column name. In the HEX-editor this 1 was corrected to 0 and the backup copy was restored. Following this, the programmer memorized once and for all how to execute the backup process and restore successfully!

The second case: The situation was catastrophic. The database corrupted on the extension phase because of lack of disk space. When increasing the database size, the server creates a series of critically important pages (for example, Transaction Inventory Page and Page Inventory Page, additional pages for RDB$Pages relations) and writes them down at the end of database.

As a result, the database could not be opened, neither by administration facilities nor using the utility GBAK. And when we tried to connect to the database, an error message (Unexpected end of file) appeared.

When we ran the utility GFIX strange things happened: The program was working in an endless cycle. When GFIX was working, the server was writing errors to log (file InterBase log) at high speed (around 100 Kb per second). As a result, the log file filled all the free disk space very quickly. We even had to write a program that erased this log by timer. This process lasted for a long time - GFIX was working for more than 16 hours without any results.

The log was full of the following errors: Page XXX doubly allocated. When starting InterBase sources (in file val.c) there is a short description of this error. It says that this error appears when the same data page is used twice. It's obvious that this error is a result of corruption of critically important pages.

As a result, after several days of unsuccessful experiments, all attempts to restore the data in the standard way were abandoned. Which is why we had to use a low-level analysis of the data stored in the damaged database.

Alexander Kozelskiy, head of Information Technologies at East View Publications Inc, had the idea of how to extract information from similar unrecoverable databases. The method of restoring, arrived at as a result of our research, was based on the fact that a database has page organization and data from every table is collected by data pages. Each data page contains an identifier of the table for which it stores data. It was especially important to restore data from several critical tables. There was data from similar tables, received from an old backup copy that worked perfectly and could be used as a model. This database sample was loaded into an editor of hexadecimal sources and then we searched for the patterns of the data that interested us. This data was copied into a buffer in hexadecimal format and then the remains of the damaged database were loaded into the editor. A sequence of bytes corresponding to the sample was found in the damaged database, and the page was analyzed (on which this sequence was found).

At first we needed to define the start page, which wasn't difficult because the size of the database file is divisible by the data page size. The number of current bytes divided by page size - 8192 bytes, approximates the result to integer (and we obtained the number of the current page). Then the number of current page was multiplied by page size and we got the number of bytes corresponding to the beginning of the current page. Having analyzed the header, we defined the type of page (for pages with data the type is 5 - please refer to the file ods.h from the set of InterBase sources as well as the identifier of the necessary table.

Then a program was written, that analyzed the whole database, collected all the pages for the necessary table into one single piece and moved it to file. Thus, once we had the data we initially needed, we began analyzing the contents of the selected pages.

InterBase uses data compression widely in order to save space. For example, a string such as VARCHAR containing an ABC string, stores a sequence of following values: string length (2 bytes), in our case it is 0003, and then the symbols themselves followed by a checksum. We had to write an analyzer of the string as well as other database types that converted data from hexadecimal format into an ordinary view. We managed to extract up to 80% of the information from several critical tables using a "manual" method of analyzing the database contents. Later, on the basis of this experience, Oleg Kulkov and Alexey Kovyazin, one of the authors of this book, developed the utility InterBase Surgeon which performs direct access to the database, bypassing the InterBase engine and enables you to read directly and interpret the data within the InterBase database in a proper way.

Using InterBase Surgeon, we have managed to detect the causes of corruption and restore up to 90% of absolutely unrecoverable databases, which can't be opened by InterBase and restored by standard methods. This program can be downloaded from the official site http://www.ib-aid.com.

See Also:
Database Validation
GFIX
GBAK and GSPLIT
Backup Database
Restore Database
System Objects
Forced Writes
Database Properties
Firebird for the database expert: Episode 3 - On Disk Consistency
Preventing data loss
Alternative database repair methods

Firebird for the database expert: Episode 1 - Indexes

By Ann Harrison

Firebird differs in significant ways from other relational database management systems. Understanding the differences will allow you to create better-performing Firebird applications.

Audience: Experienced database application developers.

Moving to Firebird can be disconcerting for developers who have worked with other relational database management systems. In theory, relational databases separate the logical design of an application from the physical storage of the data, allowing developers to focus on what data they want their applications to access, rather how the data should be retrieved. In practice, the mechanics of each database management system make some styles of access much faster than others.

Developers learn to use methods that work with the database management systems they know. Developers who are familiar with Oracle or Microsoft SQL Server find that Firebird indexes, concurrency model, and failure recovery behave differently from the databases they know. Understanding and working with those differences will make your move to Firebird less stressful and more successful. This paper focuses on the unusual characteristics of Firebird indexes.

Index types

Firebird supports only one index type: a b-tree variant. Indexes can be unique or allow duplicates; they can be single key or compound key, ascending or descending.

Record location

Many databases cluster records on the primary key index, either directly storing the data in the index or using the key to group records. In a well-balanced system clustering on primary keys makes primary key lookup very efficient. If the full record is stored in the index, the data level becomes very wide, making the whole index deep and more expensive to traverse than a shallower, denser index. Record clustering can result in sparse storage or overflows depending on the design specifications and data distribution.

Firebird stores records on data pages, using the most accessible page with sufficient space. Indexes are stored on index pages and contain a record locater in the leaf node. Access costs of primary and secondary indexes. When data is clustered on the primary key, access by primary key is very quick. Access through secondary indexes is slower, especially when the secondary key index uses the primary key as the record locater. Then a secondary index lookup turns into two index lookups. In Firebird, the cost of primary and secondary index lookups is identical.

Index access strategy

Most database systems read an index node, retrieve the data - this technique also leads to bouncing between index pages and data, which can be resolved by proper placement control, assuming that the DBA has the time and skill to do so. For non-clustered indexes this technique also results in rereading data pages.

Firebird harvests the record locaters for qualifying records from the index, builds a bitmap of record locaters, and then reads the records in physical storage order.

Index optimization

Because their access strategy binds index access and record access tightly, most database optimizers must choose one index per table as the path to data. Firebird can use several indexes on a table by `'AND'`ing and `'OR'`ing the bitmaps it creates from the index before accessing any data.

If you have a table where several different fields are used to restrict the data retrieved from a query, most databases require that you define a single index that includes all the fields. For example, if you are looking for a movie that was released in 1964, directed by Stanley Kubrick, and distributed by Columbia you would need an index on `Year`, `Director`, and `Distributor`. If you ever wanted to find all pictures distributed by Stanley Kubrick, you would also need an index on `Director` alone etc. With Firebird, you would define one index on `Director`, one on `Distributor`, and one on `ReleaseDate` and they would be used in various combinations.

Long duplicate chains

Some databases (Firebird 2 for one) are better than others (Firebird 1.x for one) at removing data from long (>10000) duplicate chains in indexes. If you need an index on a field with low selectivity for a Firebird 1.x database, create a compound key with the field you want to index first and a more selective field second. For example, if you have an index on `DatePaid` in the table `Bills`, and every record is stored with that value null when the bill is sent, then modified when the bill is paid, you should create a two-part index on `DatePaid`, `AccountNumber`, instead of a single key index on `DatePaid`.

Indexes in lieu of data

Non-versioning databases resolve some queries (counts for example) by reading the index without actually reading the record data. Indexes in Firebird (like Postgres and other natively versioning databases) contain entries that are not yet visible to other transactions and entries that are no longer relevant to some active transactions. The only way to know whether an index entry represents data visible to a particular transaction is to read the record itself.

The topic of record versions deserves a long discussion. Briefly, when Firebird stores a new record, it tags the record with the identifier of the transaction that created it. When it modifies a record, it creates a new version of the record, tagged with the identifier of the transaction that made the modification. That record points back to the previous version. Until the transaction that created the new version commits, all other transactions will continue to read the old version of the record.

In the previous example, when a transaction modifies the indexed field `DatePaid`, Firebird creates a new version of the record containing the new data and the identifier of the transaction that made the change. The index on that field then has two entries for that record, one for the original `NULL` value and one for the new `DatePaid`.

The index does not have enough information to determine which entry should be counted in responding to a query like "`select count (*) from Bills where DatePaid is not null`".

Index key length

In Firebird Version 1.x, the total length of an index key must be less than 252 bytes. Compound key indexes and indexes with non-binary collation sequences are more restrictive for reasons described in the section on key compression. Firebird 2 allows keys up to 1/4 of the page size, or a maximum of 4Kb.

Index key representation

Firebird converts all index keys into a format that can be compared byte-wise. With the exception of 64bit integer fields, all numeric and date fields are stored as double precision integer keys, and the double precision number is manipulated to compare byte by byte. When performing an indexed lookup, Firebird converts the input value to the same format as the stored key. What this means to the developer is that there is no inherent speed difference between indexes on strings, numbers, and dates. All keys are compared byte-wise, regardless of the rules for their original datatype.

Index key compression

Firebird index keys are always stored with prefix and suffix compression. Suffix compression removes trailing blanks from string fields and trailing zeros from numeric fields. Remember that most numeric values are stored as double precision and so trailing zeros are not significant. Suffix compression is done for each key field in a compound key without losing key field boundaries. After removing the trailing blanks or zeros, the index compression code pads field to a multiple of four bytes, and inserts marker bytes every four bytes to indicate the position of the field in the key.

Consider the case of a three field key with these sets of values:

```
"abc","def","ghi"
"abcdefghi","",""
```

Simply eliminating trailing blanks would make the two sets of values equal. Instead, Firebird turns the first set of key values into “abc 1def 2ghi 3” and the second into “abcd1efgh1i 1 2 3”.

Firebird version 1.x compresses the prefix of index keys as they are stored on pages in the index. It stores the first key on a page without prefix compression. Subsequent keys are stored after replacing the leading bytes that match the leading bytes of the previous key with a single byte that contains the number of bytes that were skipped. The two keys above would be stored like this:

```
"0abc 1def 2ghi 3" "3d1efgh1i 1 2 3"
```

An index entry that exactly matches the previous entry is stored as a single byte that contains its length. Firebird 2 also performs prefix compression, but uses more dense representation. The combination of compression techniques eliminates some of the rules about constructing keys. Suffix compression occurs on all segments of a key, so long varchar fields should be placed in their logical spot in a compound key, not forced to the end. On the other hand, if part of a compound key has a large number of duplicate values, it should be at the front of a compound key to take advantage of prefix compression.

See also:
Index/Indices
Indexed reads/non-indexed reads
Indices
Recompute selectivity of all indices
Recreating Indices 1
Recreating Indices 2

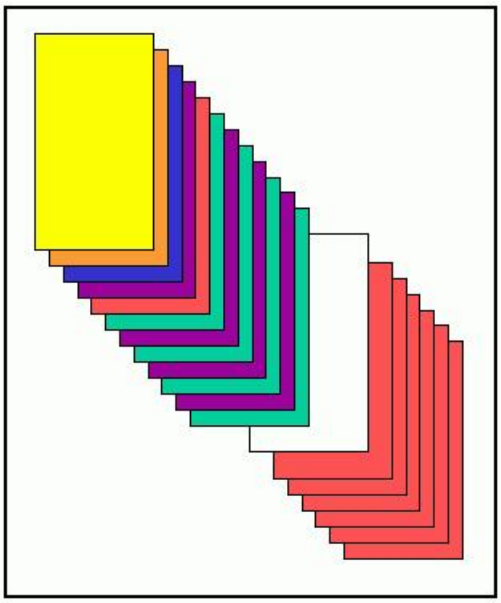
Firebird for the database expert: Episode 2 - Page Types

By Ann Harrison

Database file

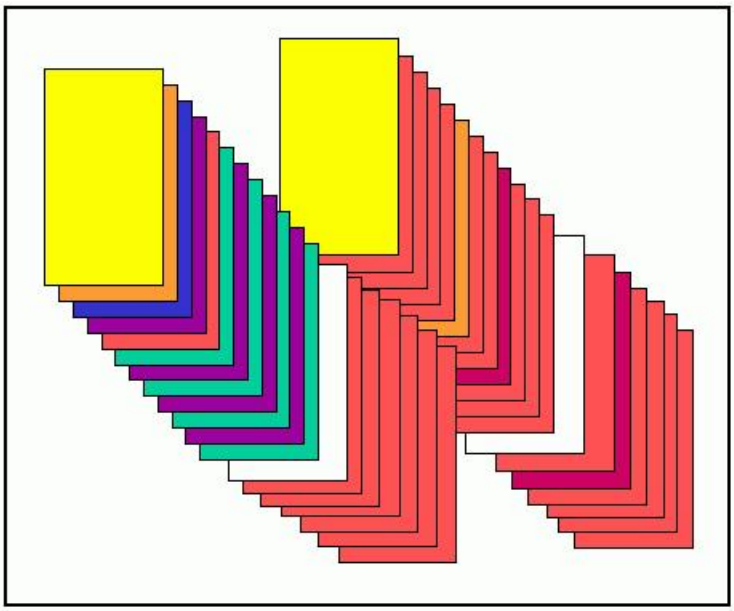
A Firebird database is a sequence of fixed length pages normally all contained in a single file.

Different pages have different functions - in this case the yellow page is the database header, followed by a PIP, the unused WAL, a pointer page, a data page, then alternating index root and pointer pages. The white page indicates that the diagram skips several hundred pages then continues with data pages.



Multi-file database

A multi-file database breaks the sequence into multiple files, each with a header page. Aside from the extra header pages, there is no difference between a multi-file database and a single file database.



Generic page header

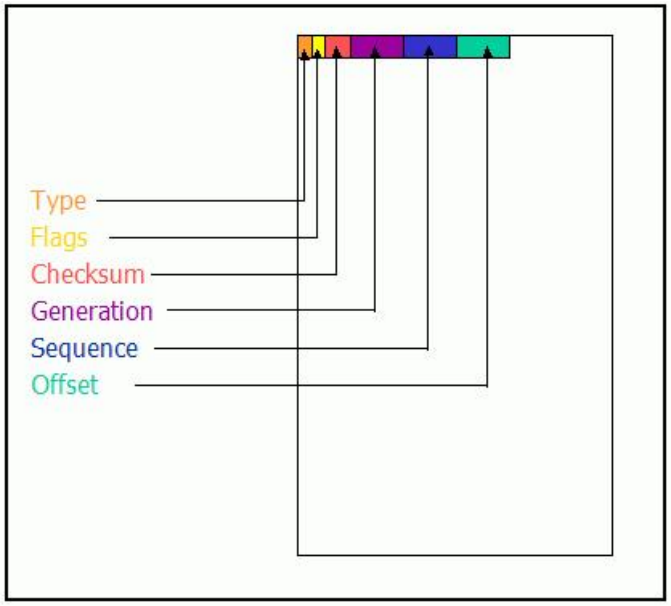
Each page has a header that indicates what type of page it is, and provides other information that applies to all pages. Most page types have addition header information that follows the standard header. In the standard header, the first byte is the page type.

The next byte contains flags that are specific to individual page types. Currently, only blob pages and b-tree (index) pages use the page flags. Other page types - the header for one - also have a separate area for flags.

The next two bytes were a checksum, but now always contain the value 12345.

The next four bytes are the page generation incremented each time the page is written.

The next eight bytes are reserved for the sequence and offset of the page's entry in a log. The logging project has been abandoned and those bytes are waiting for a good use.

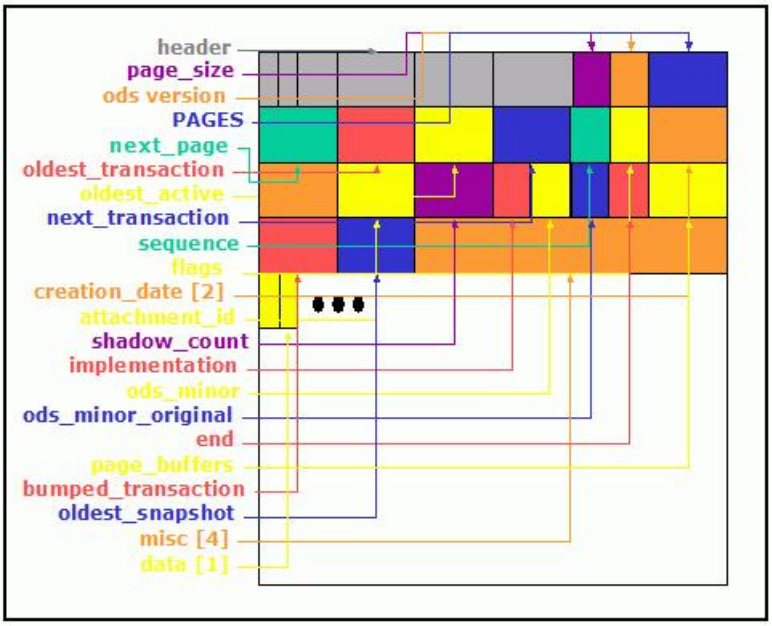


Header page (HDR)

Page Type 1 is a header page. Each database file has one header page, which is page 0 in the file.

The first header page in a database describes the database: the page size, next transaction id, various settings, etc.

The header pages of subsequent files in the database contain only the length of the current file and the name of the next file.



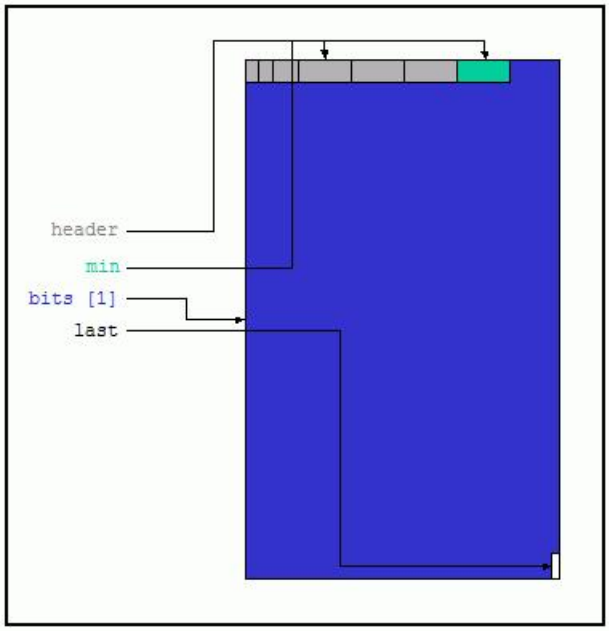
Please also refer to *Structure of a header page*.

Page Inventory Page (PIP)

Page Type 2 is a page inventory page (PIP). PIPs map allocated and free pages. The header of a PIP includes the offset on this page of the bit that indicates the first available page on the PIP.

The body of a PIP contains an array of single bits that reflect the state of pages in the database. If the bit is one, then the corresponding page is not in use. If the bit is zero, then the page is in use.

PIPs occur at regular intervals through the database, starting at page 1. The last page allocated on each PIP is the next PIP.

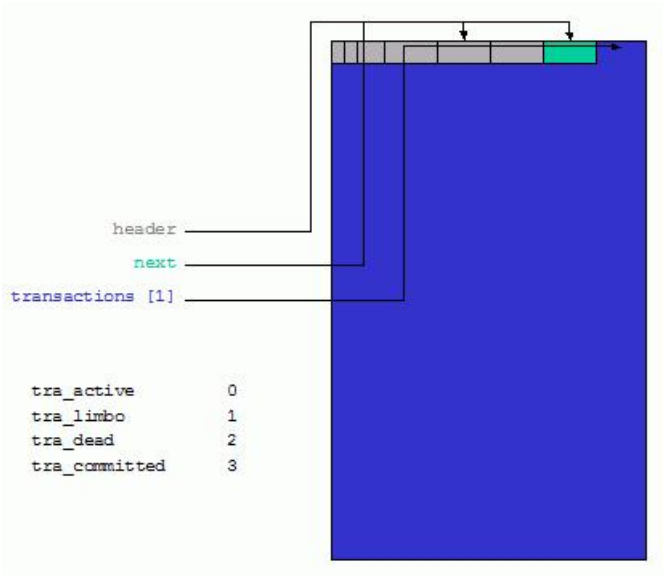


Transaction Inventory Page (TIP)

Page Type 3 is a transaction information page (TIP). The TIP header includes the address of the next TIP.

The body of a TIP is an array of pairs of bits that reflect the state of transactions. If both bits are 0, the transaction is active or has not started. If both bits are 1, the transaction is committed. If the first bit is 1 and the second bit is 0, the transaction . If the first bit is 0 and the second is 1, the transaction is in limbo.

Limbo is the state of a two phase transaction that has completed the first phase, but not the second.

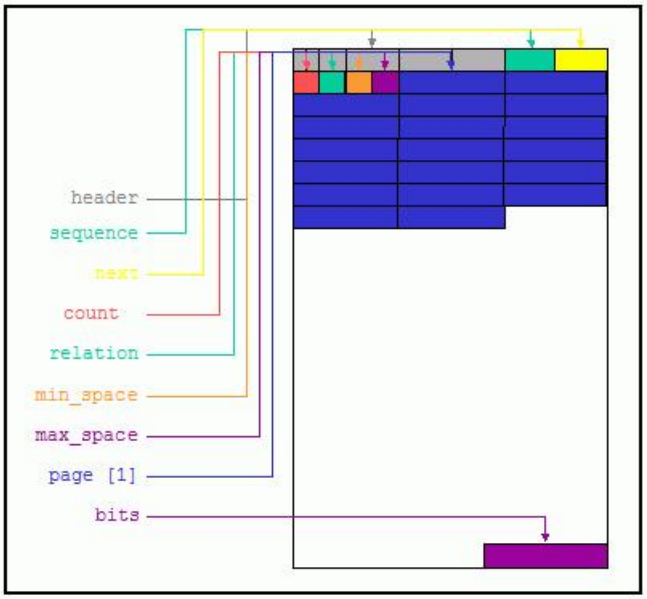


Pointer page (PTR)

Page Type 4 is a pointer page. Each pointer page belongs to a particular table and has a specific sequence within the table.

The additional header information on a pointer page includes its sequence in the pointer pages for this table, the page number of the next pointer page for the table, the next free slot on the page, the number of used slots on the page, the relation id of the table, the offset of the first slot on the page that indicates a page that is not full, and the offset of the last slot on the page that indicates a data page that is not full.

Pointer pages contain arrays of 32-bit integers that contain the page numbers of pages in a table. At the bottom of the pointer page, an array of bits indicates the fill level of each page.



Data page (DPG)

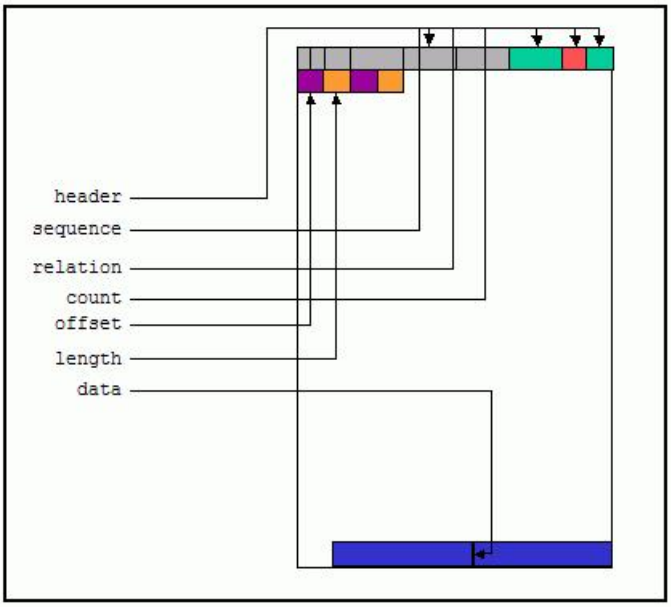
Page Type 5 is a data page. Each data page belongs to a specific table.

The additional header information in a data page is the position of this page in the list of data pages for the table, the relation id of the table, and the number of entries on this page.

The body of a data pages starts with an array of pairs of 16 bit words. The first part of the pair is the offset on the page of a piece of data - a record, blob, or record fragment. The second part of the pair is the length of the data. As more data is stored on the page, the index grows downward.

The data - records, blobs, and fragments - start at the end of the page and go upward.

Further information can be found in the chapters, *Structure of a data page* and *Where do data pages come from*.



Index Root page (IRT)

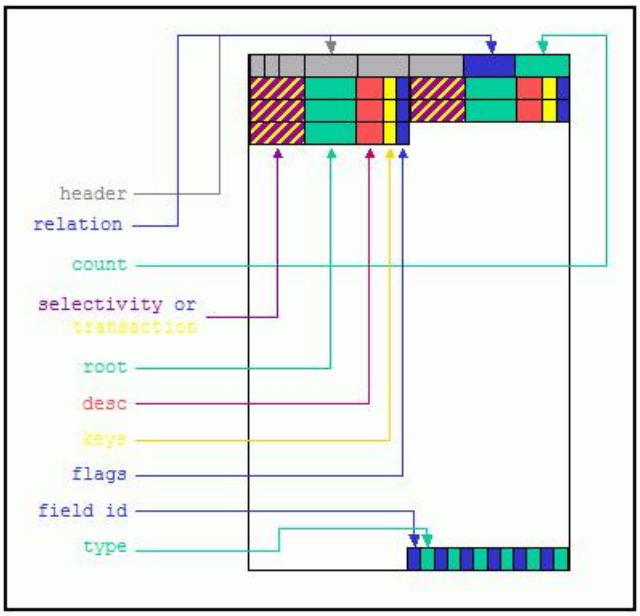
Page Type 6 is an index root page. Each table has a single index root page that describes the indexes for the table. This page describes the IRT in Firebird 1.5 and earlier.

The additional header information for an index root page is the identifier of the relation to which the page belongs, and a count of the number of indexes for that table.

The body of an index root page contains an array of index descriptors coming down from the top of the page and an array of index segment descriptors coming up from the bottom.

Each index descriptor starts with the selectivity if the index has already been created, or a transaction id if the index is being created. The next 32 bits are the page number of the top of the actual index. Next is the 32-bit offset of the field descriptors for the index at the bottom of the page. The next byte is the number of key fields, then a flag byte.

The array of segment descriptors contains two bytes per segment, one for the field id and one for the field type.



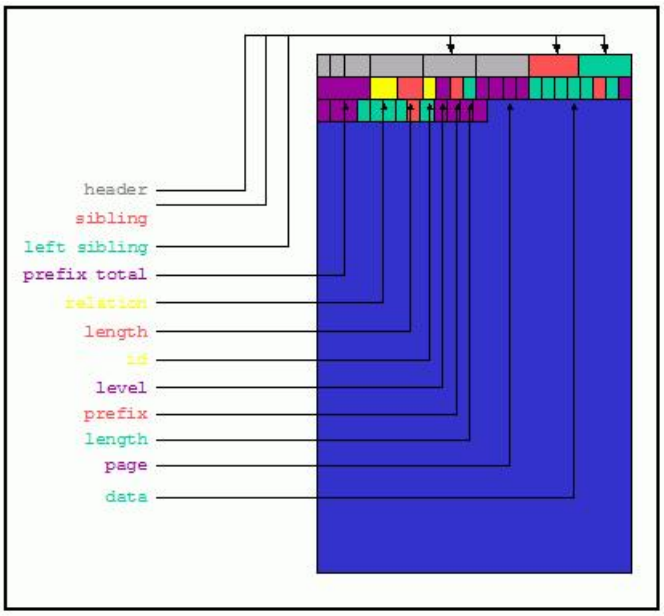
B-tree page (BTR)

Page Type 7 is an index or b-tree page.

All indexes in Firebird are a b-tree variant, starting with a single page at the top - confusingly called the root - confusing both because the root is at the top and because the root of an index is different from the table's index root page.

The additional header data in a b-tree page includes the number of the page with the next higher values for this level of the index, the address of the page with the next lower values for this level, the total amount of space which is saved on this page by the use of prefix compression, the relation id of the table this index describes, the amount of space used on this page, the identifier of the index in which this page participates, and the level of this page in the index.

The rest of the page is filled with index entries.

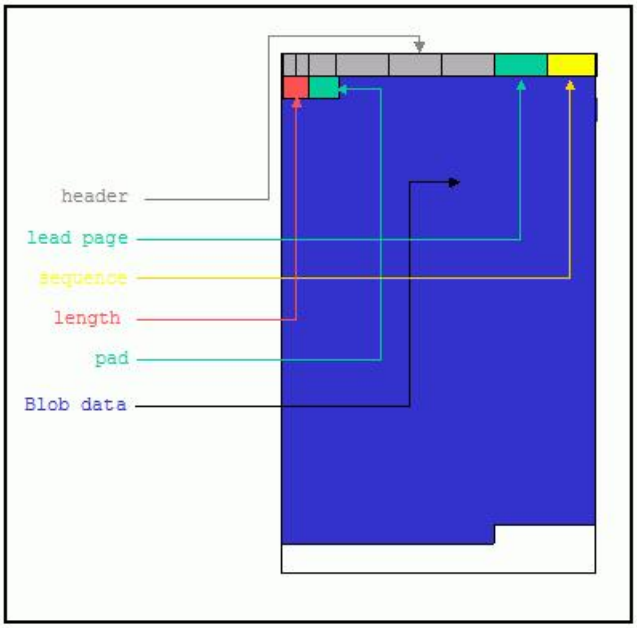


Blob page (BLP)

Page Type 8 is a blob page. Small blobs are stored on data pages. Blobs larger than a page are stored on a sequence of blob pages.

The type-specific header information for a blob page includes the page number of the first page of this blob, the position (sequence) of this page in the list of pages that contain the blob, the amount of data stored on the page, and a pad word to allow the blob data to start on a long word boundary.

The remainder of the page contains blob data for a single blob.

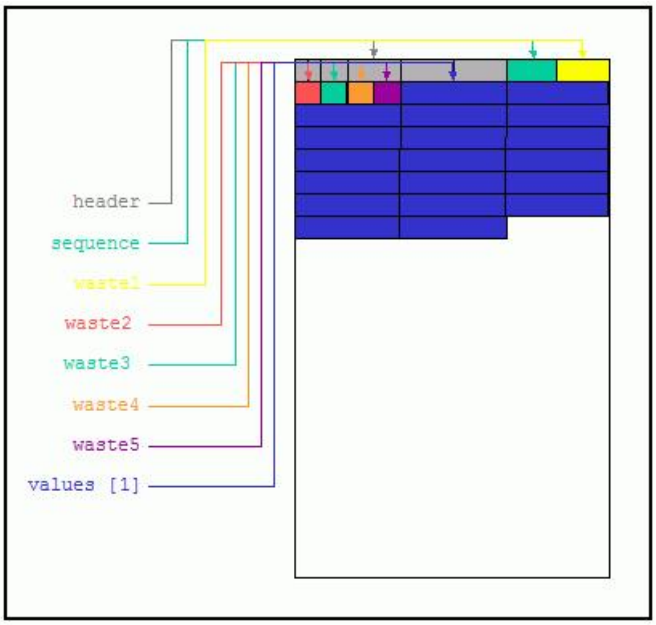


Generator page (GEN)

Page Type 9 is a generator page.

There is no extra information in the header of a generator page, but there are several wasted words. Originally generator pages were a subset of pointer pages and did not have their own type. When generators were extended from 32 to 64 bits, having a separate page type became important, but changing the header would have invalidated old databases. Sometime we ought to fix that and add a sequence number to the generator page header.

A generator page contains an array of 64-bit integers. Each element of the array contains the current value of a generator.



Firebird for the database expert: Episode 3 - On Disk Consistency

By Ann Harrison

Unlike most databases, Firebird has no external journal file or log for recovery from transaction or system crashes. The database is its own log. After a system crash, productive work begins as soon as the server restarts. Changes made by transactions that failed are removed automatically and transparently (see Record versions as an undo log). One necessary precondition for instant recovery is that the disk image of the database must always be consistent. Firebird achieves that consistency by tracking relationships between database pages and writing pages in an order that mainatains those dependencies. The ordering is called *careful write*.

On disk consistency

Reduced to its essence, the careful write means that the database on disk will always be internally consistent. More pragmatically, when the system writes a page that references another page, that other page must have been written previously in a state that supports the reference. Before writing a page that has a pointer from a record to its back version on another page, the system must have written that other page. Before writing out a new data page, the system must write out a version of a page inventory page (PIP) that shows the page is in use. The new data page has to be on disk, formatted and happy, before the table's pointer page that references the new page can be written.

Inter-page relationships are handled in the code through a dependency graph in the cache manager. Before a page is written, the cache manager checks the graph and writes out all pages that page depends on. If a change will create a loop in the graph, the cache manager immediately writes as many pages as necessary to avoid the loop.

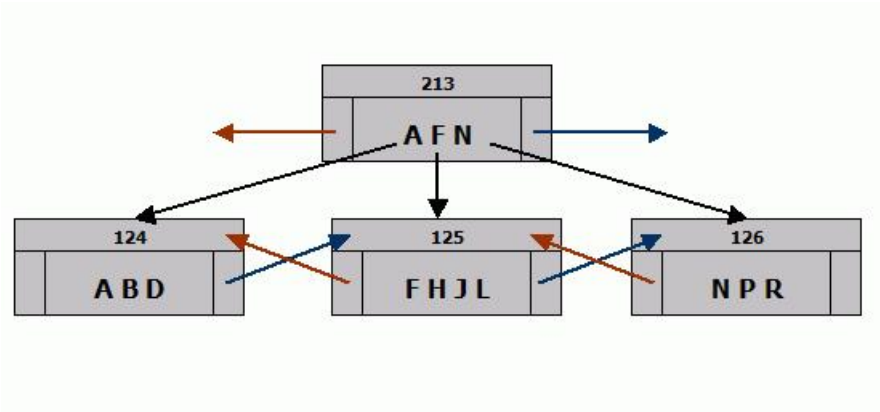
The tricky bits are identifying dependencies and avoiding the impossible situation - well, those and keeping the system fast. Identifying dependencies just requires meticulous coding. If you have to put a record back version on a different page from the main version, the page with the pointer has a dependency on the page with the back version. If you allocate a new data page, that data page has a dependency on the PIP that records whether the page is in use, and the pointer page that ties the data page into the table has a dependency on the data page. For more information on page allocation see Where do data pages come from?

The impossible situation is one where pages point to each other in a way that can't be separated. Two pages can point to each other - you can have a primary record version on page 214 with its back version on page 215 and a different record with its primary version on page 215 and a back version on 214. The chances that the cache manager will find a cycle in the dependency graph are high, and one page may need to be written twice in the process of flushing the pages out, but it works because the two relationships are separable.

If, on the other hand, you need a double-linked chain of pages - index pages come to mind, there is no separable relationship. Each page depends on the other and neither can be written first. In fact, Firebird index pages are double-linked, but the reverse link (high to low in a descending index) is handled as unreliable. It's used in recombining index pages from which values have been removed, but not for backward data scans. The architecture group is currently discussing ways to make the reverse link reliable enough for retrieving data, but we haven't agreed on a solution.

For those who haven't spent an embarrassing part of their adult lives worrying about on disk consistency and double-linked lists, let me try to explain.

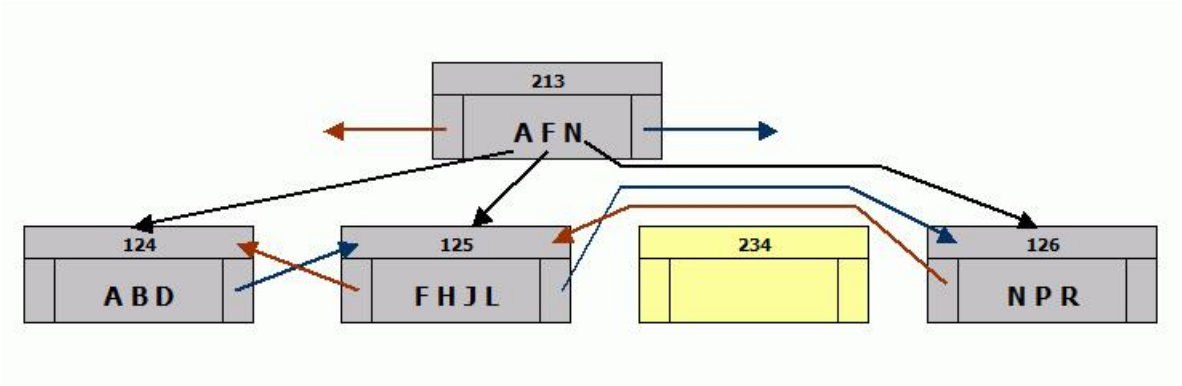
Assume that each index page can hold only four values - instead of the hundreds that it actually holds. Consider the leaf level of an an index that consists of pages 124, 125, and 126 in that order. The next level in the index is represented as page 213. Each index page has a pointer to its left and right neighbor. The neighbors of page 213 are omitted as boring. Page 124 holds A, B, D; page 125 holds F, H, J, L and page 126 holds N, P, R. Now you want to add a new entry to the index. It has to be put on page 125, but page 125 is full, so you need to add a new page between 125 and 126. The color key for diagrams can be found at the end of this article.



You want to store K.

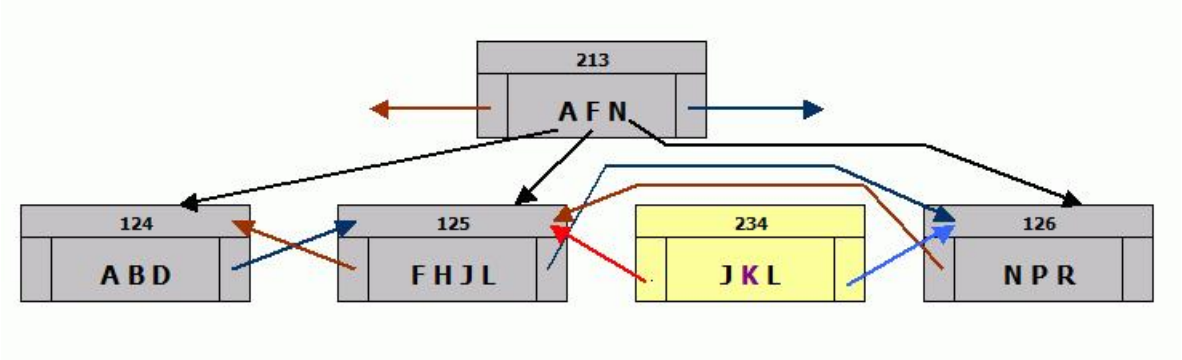
The way the index code handles this sort of problem is:

1. Read the current PIP (page information page) to find a free page - lets say it's 234.
2. Change the PIP to reflect that page 234 is not available.
3. Set up a dependency so that PIP will be written before the new index page.
4. Format a buffer to look like an index page with the page number 234.

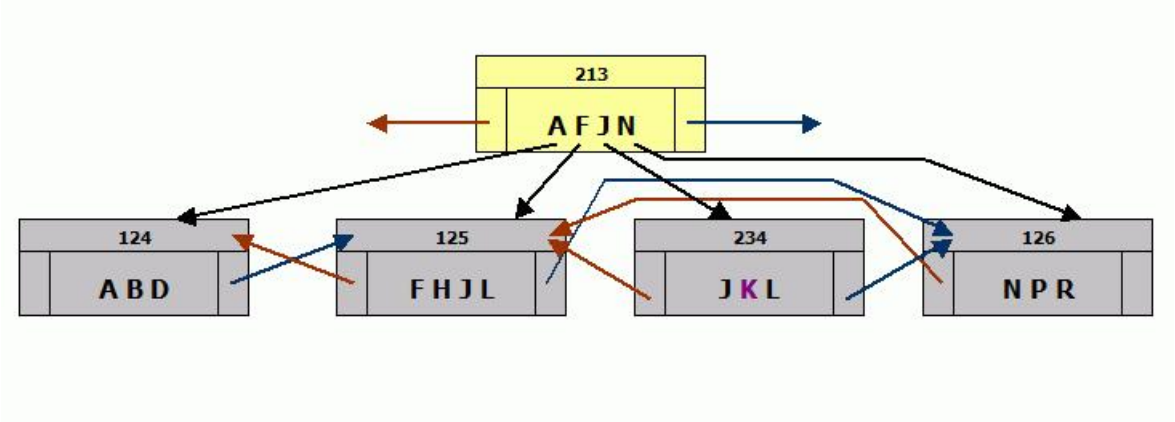


5. Copy half the index entries - entries J, K, and L - from the page that overflowed onto page 234.
6. Copy the pointer to the next index page from the page that overflowed (125) onto the new page (234).
7. Make the new page (234) point backward to the page that overflowed (125).
8. Mark page 234 to be written. Now page 234 can be written if it is needed by another transaction, as long as the PIP is written first.

At this point, page 125 still points forward to 126, which points backward to 125. There are two copies of the index entries for J & K, but that doesn't matter because there's no way to get to page 234 - it's not in the upper index level yet and will be skipped by a scan, regardless of direction.

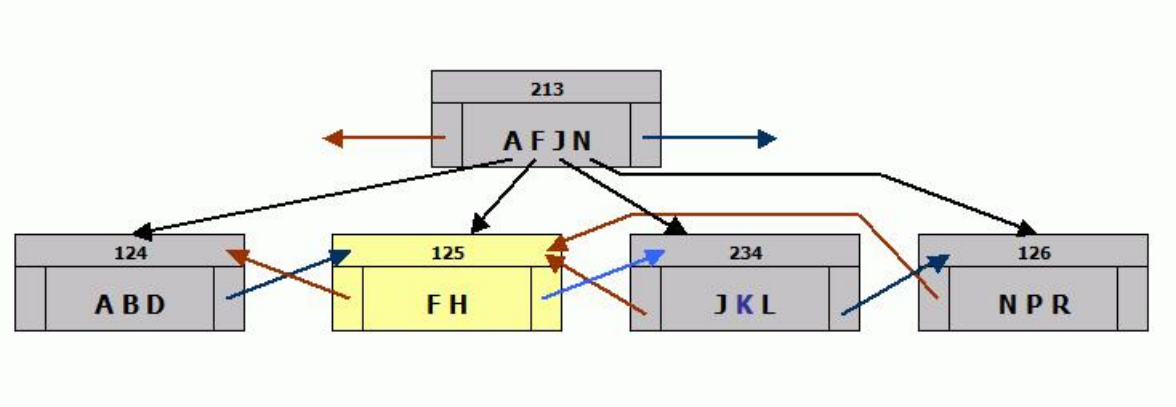


9. Fix the upper levels so they include the first value of the new page. That change may cause an upper level page to overflow, resulting in the same series of steps at that level, and so on up to the top. If the very top page splits, the index gets a new level.



Now, the upper level contains an entry for J points to 234 rather than 125. Scans still work because anything that starts lower than J will skip node 246 and anything higher than J will skip 125.

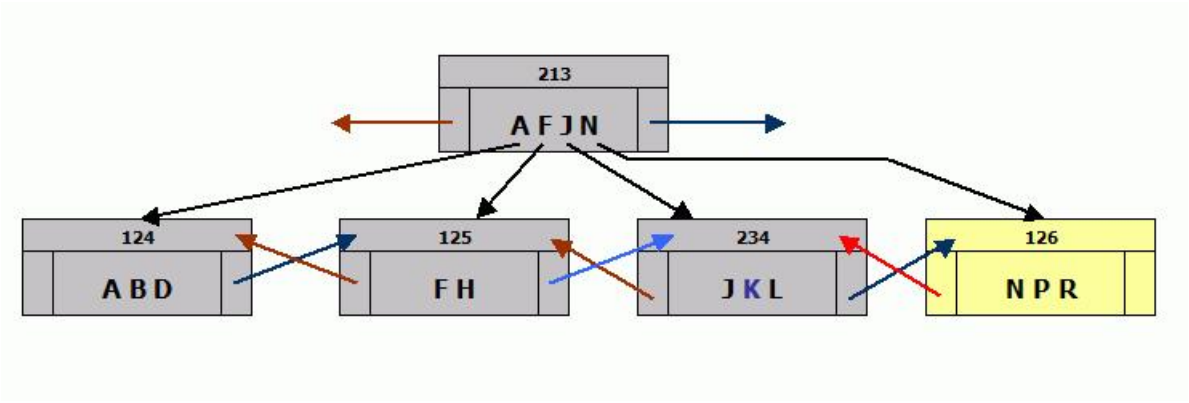
10. Remove the copied index entries from the page that overflowed and change its back pointer to point to the new page.
11. Write that page.



A forward scan still works, but a backward scan that starts with N or higher will never see the values J and L. The code that handles recombinations does a lot of sanity checking and quits when it sees a problem. That strategy doesn't work for record retrievals.

12. Fix the back pointer on the page after the new page to point to the new page.

Now the structure works again.



There are a couple of unavoidable awkward situations that occur during page allocation and release, and result in orphan pages and orphan back versions. Orphans are wasted space but do not affect the integrity of the database.

At the page level, GFIX will sometimes report orphan pages after a crash. If the system crashes after a page has been allocated and the PIP written, but before the pointer page that makes the data page part of the table has been written, that data page becomes an orphan. Note that the data on that page is uncommitted because the change that commits a transaction - writing the transaction inventory page with the transaction marked committed - does not happen until all page that were created by the transaction have been written.

If the system crashes after a pointer page has been written, removing an empty data page from a table, but before the PIP has been written to reflect that the page is free.

If the system crashes in the middle of dropping an index or table, GFIX may find lots of orphan pages - a single write releases all the pages that were part of the table or index, and that write must happen before any of the PIPs can be changed.

Back versions must be written before the record that points to them and can not be removed until after the pointer to them has been cleared. A crash between those steps makes the back version an orphan - it occupies space but is not connected to a record.

Key to diagram colors

See also:
Database Corruption
Preventing data loss
Alternative database repair methods

Firebird for the database expert: Episode 4 - OAT, OIT and Sweep

By Ann Harrison

This is an ancient message from an InterBase self-help list, responding to a question about slow inserts. It deals with questions of sweeping, oldest active transaction, oldest interesting transaction, etc. I've cleaned up the spelling and added a few side notes.

From: Ann Harrison

Subject: Re: Interbase - what is it doing?

Let me also take a crack at this, since I may be the only person with more experience trying to explain it than Jim (Starkey - my previous & current boss/mentor/ (he says "say husband") etc.). The problem may be a sweep.

First, for Novice InterBasians (and fresh-hatched Firebirdies) - when I say transaction, I mean a set of actions against the database, ending with a Commit, Rollback, Prepare/Commit (two-phase commit), or abrupt disconnection from the database. A single action, like inserting, updating, or deleting a record is a statement. Many tools provide automatic transaction support, so you may not be aware of the number of transactions created on your behalf. Any tool that performs a commit per statement is not your friend if you're loading a database.

Here's the hard-core stuff.

Explanations of sweeping tend to be unsatisfactory because the subject is complicated, and depends on understanding several other complicated ideas.

Disclaimer: This description applies to the state of the world in V3.x, with extrapolation to V4.x specifically noted. I have no current connection with InterBase or Borland. (See note 1 in the Summary).

Lets begin by defining transaction states, garbage, garbage collection, and Oldest Interesting Transaction (OIT), Oldest Active Transaction, and sweeping...

Transaction States

Transactions have four states: active, committed, limbo, and rolled back.

Taking these cases in order from the least complex to the most:

- **Limbo:** A transaction that started a two-phase commit by calling the `PREPARE` routine. The transaction may be alive or not. At any point, the transaction may re-appear and ask to `COMMIT` or `ROLLBACK`. Changes it made can neither be trusted nor ignored, and certainly cannot be removed from the database.
- **Committed:** A transaction is which completed its activity successfully. Either A) it called `COMMIT` and the commit completed successfully, or B) it called `ROLLBACK` but made no changes to the database, or C) it called `ROLLBACK` and its changes were subsequently undone and its state changed to committed. This transaction is finished and will never be heard from again, and its remaining changes are now officially part of the database.
- **Rolled back:** A transaction which either: A) called `ROLLBACK` and requested that its changes be removed from the database, or B) never called `COMMIT` so was marked as `ACTIVE`, but discovered to be dead by another transaction which marked it as rolled back. In either case, changes made by this transaction must be ignored and should be removed from the database.
- **Active:** A transaction which: A) hasn't started. B) has started and hasn't finished. C) started and ended without calling any termination routine. (e.g. crashed, lost communication, etc.)

How do transactions know about each others state?

The state of every transaction is kept on a Transaction Inventory Page (TIP). The single change made to the database when a transaction commits is to change the state of the transaction from `ACTIVE` to `COMMITTED`. When a transaction calls the rollback routine, it checks its *Update* flag - if the flag is not set, meaning that no updates have been made, it calls `COMMIT` instead. So, rolling back read-only transactions doesn't mess up the database.

How can a transaction go back from *Active* to *Rolled Back* if it exists abnormally?

This can happen in one of two ways:

1. When a transaction starts, it takes out a lock on its own transaction id. If a transaction (B) attempts to update or delete a record and finds that the most recent version of the record was created by a transaction (A) whose TIP state is `ACTIVE`, transaction B tries to get a conflicting lock on A's transaction id. A live transaction maintains an exclusive lock on its own id, and the lock manager can probe a lock to see if the owner is still alive. If the lock is granted, then B knows that A died and changes A's TIP state from `ACTIVE` to `ROLLED BACK`.
2. When a transaction starts, it checks to see if it can get an exclusive lock on the database - if it can no other transactions are active. Every active transaction has a shared lock on the database. If it gets an exclusive lock, it converts all Active TIP entries to `ROLLED BACK`.

To reiterate, a transaction is ACTIVE (meaning that it appears to be alive), LIMBO (meaning that its outcome can not be determined), COMMITTED (meaning that it completed successfully) or ROLLED BACK (meaning it acknowledged its faults and left the field in disgrace).

Garbage

InterBase is a multi-generational database. When a record is updated, a copy of the new values is placed in the database, but the old values remain (usually as a bytewise difference from the new value). The old value is called a "Back Version". The back version is the rollback log - if the transaction that updated the record rolls back, the old version is right there, ready to resume its old place. The back version is also the shadow that provides repeatable reads for long running transactions. The version numbers define which record versions particular tranasctions can see.

When the transaction that updated the record commits and all concurrent transactions finish, the back version is unnecessary. In a database in which records are updated significantly and regularly, unnecessary back versions could eventually take up enough disk space that they would reduce the performance of the database. Thus they are GARBAGE, and should be cleaned out.

Garbage Collection

Garbage collection prevents an update-intensive database from filling up with unnecessary back versions of records. It also removes record versions created by transactions that rolled back. Every transaction participates in garbage collection - every transaction, including read-only transactions.

When a client applicaition reads a record from a Firebird database, it gets a record that looks like any record from any database. Two levels lower, somewhere in the server, InterBase/Firebird pulls a string of record versions off the disk. Each version is tagged with the transaction id of the transaction that created it. The first one is the most recently stored. At this point, the server has two goals: 1) produce an appropriate version of the record for the current transaction 2) remove any versions that are garbage - either because they were created by a transaction that rolled back or because they are so old that nobody will ever want to see them again.

Extra Credit Aside: There is a third kind of garbage collection which happens at the same time. InterBase also uses a "multi-generational" delete. When transaction deletes a record, does the record go away right then? No, of course not. The deletion could be rolled back. So instead of removing the record, InterBase sticks in a new record version containing only a DELETE marker, and keeps the old version. Sooner or later the deletion commits and matures. Then the whole thing, deletion marker and all record versions are garbage and get ... (right you are!) garbage collected.

Garbage Collection – resumes:

Garbage collection is co-operative, meaning that all transactions participate in it, rather than a dedicated garbage team. Old versions, deleted records, and rolled back updates are removed when a transaction attempts to read the record. In a database where all records are continually active, or where exhaustive retrievals (i.e. non-indexed access) are done regularly on all tables, co-operative garbage collection works well, as long as the transaction mask stays current.

For databases in which all access is indexed, old records are seldom - or never - revisited and so they seldom - or never - get garbage collected. Running a periodic backup with gbak has the secondary effect of forcing garbage collection since gbak performs exhaustive retrievals on all tables.

See also:
Backup Database / Garbage Collection
Garbage Collectors

Oldest Interesting Transaction (OIT)

To recognize which record versions can garbage collected, and which updates are rolled back and can be ignored, every transaction includes a transaction mask which records the states of all interesting transactions. A transaction is interesting to another transaction if it is concurrent - meaning that its updates are not committed, or if it rolled back - meaning that its updates should be discarded, or if it's in limbo.

The transaction mask is a snapshot of the states of all transactions from the oldest interesting, to the current. The snapshot is made when the transaction starts and is never updated. The snapshot depends on the number of transactions that have started since the oldest interesting transaction.

Oldest Active Transaction (OAT)

This one sounds easy - but it's not. The oldest active transaction is not the oldest transaction currently running. Nor is it the oldest transaction marked ACTIVE in the TIP. (Alas). It is the oldest transaction that was active when the oldest transaction currently active started. The bookkeeping on this is hairy and I frankly don't remember how it was done - now I do -, but that's the rule, and it does work.

Any record version behind a committed version created by a transaction older than the oldest transaction active when the oldest transaction currently active started is garbage and will never be needed ever again.

That's pretty dense. Lets ignore the commit/rollback question briefly.

Simple case: I'm transaction 20 and I'm the only transaction running. I find a record created and committed by transaction 15. I modify it and commit. You are transaction 25, and when you start, you are also the only transaction active. You read the same record, recognize that all active transactions can use the version of the record created by me, so you garbage collect the original version. In this case, your threshold for garbage collection (aka Oldest Active) is yourself.

Harder case: You continue puttering around, modifying this and that. Another transaction, say 27 starts. You are its oldest active. It too can modify this and that, as long as it doesn't modify anything you modified. It commits. I start a transaction 30. You are also my oldest active transaction, and I can't garbage collect any record version unless the newer version is older than you. I run into a record originally created by transaction 15, modified by transaction 20, then modified again by 27. All three of those transactions are committed, but I can garbage collect only the original version, created by transaction 15. Although the version created by transaction 27 is old enough for me, it is not old enough for you, and being cooperative, I have to consider your needs too.

Hardest case: I'm transaction 87, and when I started, all transactions before 75 had committed, and everybody from 75 on was active. Transaction 77 modifies a record, created originally by transaction 56. I continue to read the 56 version. All is well. Transaction 77 commits. You are transaction 95. When you start, I, number 87, am the oldest active. You read the record created by 56 and modified by 77. You can't garbage collect anything in that record because I can't read records created by any transaction newer than 74.

Maybe you know now why descriptions of the oldest active tend to be a little peculiar.

Sweeping

Sweeping is *NOT* just organized garbage collection. What sweeping seeks to do is to move the Oldest Interesting Transaction up, and reduce the size of transaction masks. It does so by changing rolled back transactions to committed transactions.

"What!!!", you say. "The woman is nuts."

But that's what a sweep does. It removes all the changes made by a rolled back transaction then changes it state to committed. (Remember we agreed earlier that a read-only transaction that rolled back could be considered committed for all the harm it did. Remove the damage, and its safe to consider the transaction committed.)

At the same time, sweep garbage collects like any other transaction.

Prior to version 4.2, the unlucky transaction that triggered the sweep gets to do the work. Other concurrent transactions continue, largely unaffected. In version 4.2 and later, a new thread is started and sweeps the database while everybody else goes about life as normal. Well, more or less normal, where the less is the amount of CPU and I/O bandwidth used by the sweep.

See also:
Database sweep / sweep interval
Database repair and sweeping using `GFIX`

Aside on limbo transactions

A transaction in limbo cannot be resolved by a sweep, will continue to trigger sweeps, and will block attempts to update or delete record versions it created. However, InterBase gives good diagnostics when it encounters a record in that state, and no tool is likely to generate incomplete two-phase commits on a random basis.

Some examples

The unfortunate case that started this message was an attempt to insert 1,000,000 records, one transaction, and one commit per record. The process slowed to a crawl, which was blamed on sweeps. Sweeping may be the problem, but I doubt it.

Case 1

Single stream of non-concurrent transactions. Transaction 1 inserts record 1, and commits. Transaction 2 starts and is both oldest active and oldest interesting. It inserts record 2 and commits. Transaction 3 starts, is oldest active and oldest interesting, inserts its record and commits. Eventually, transaction 1,000,000 starts and it too is both oldest interesting and oldest active. No sweeps.

Case 2

Lurker in the background. Transaction 1 starts, looks around, and goes off for a smoke. Transaction 2 starts, notices that 1 is oldest interesting and oldest active, inserts record 1 and commits. Transaction 3 starts, notices that 1 is still OI and OA, inserts record 2 and commits. Eventually transaction 1,000,001 starts, notices that 1 is still OI and OA so the difference between the two is still 0, stores, and commits. No sweeps again.

Case 3

Suicidal lurker. Transaction 1 starts, does something, goes out for a smoke. Transaction 2 starts, notices that 1 is oldest interesting and oldest active, inserts record 1 and commits. Transaction 3 starts, notices that 1 is still OI and OA, inserts record 2 and commits. Eventually transaction 1 succumbs to smoke inhalation and dies quietly in his corner. Transaction 15,034 (by luck) starts, gets an exclusive lock on the database, and sets Transaction 1's state to Rolled Back. Now the oldest interesting is still 1, but the oldest active is 15,034. The difference is 15,033, so no sweep yet. 4,967 transactions later the sweep occurs. Depending on the version of InterBase, transaction 20,001 may actually be charged with the time spent sweeping. Versions since 4.1 start a new thread. Once the sweep is done, the OI and OA march up together, hand in hand, and there is no more sweeping unless another transaction goes into an interesting and non-active state.

Case 4

Suicidal Twin. If for every record stored, the tool started one transaction which stored the record then rolled back, followed by a second transaction which stored the record and committed, then the difference between the OA and the OI would go up one for each record successfully stored. (Transaction 1 becomes OI when it rolls back. Transaction 2 is OA when it starts and the difference is 1. Transaction 3 rolls back, but is not OI because Transaction 1 is still older. Transaction 4 is OA and sees a difference of 3 between it and Transaction 1, and so on until transaction 20,001 which sweeps, and brings the OA and OI together at 20,001. Unfortunately its only storing record 10,001 since half the attempts to store are failing. In this *EXTREMELY UNLIKELY* case, storing 1,000,000 records would cause 100 sweeps. However, it would require an *UNUSUALLY* bad programmer to create anything that *AMAZINGLY* inefficient. Grounds for a career change.

Summary

Beats me why the load was so slow, although the commit per insert does a lot more writing than just inserting. That and forced write might explain a lot. Maybe a really fragmented disk?

Note 1: This message was written sometime last century, before I got involved with InterBase and then Firebird. I now know a lot more about InterBase 4.x, 5.x, 6.x and Firebird 1.0x, 1.5x, 2.0x, and Vulcan. That knowledge will show up passim.

See also:
Multi-generational architecture (MGA) and record versioning

Firebird for the database expert: Episode 5 - Locking and Record Versions

By Ann Harrison

Concurrency control is the mechanism that allows simultaneous users to read and write data as if each user had complete control of the database. This state of bliss is called "serializability". The state of the database after a group of concurrent transactions complete is the same as if each transaction ran alone in some unspecified order. Few, if any, database systems offer serializable transactions as their default mode.

Until recently, the most common concurrency control mechanism was locking. Of course, since transactions are imaginary electronic things, they don't actually put brass padlocks on the bits on a disk. Instead, the database system imposes a discipline on access to records, so each transaction's record use is recorded in memory and no transaction can conflict with another's noted level of use. Transactions acquire locks as they access records but never release any lock until they commit or rollback. The strategy of incrementally locking records and releasing all locks simultaneously at the end of the transaction is called two-phase locking.

Locking

Write locks prevent dirty writes

In a system that relies on locks for concurrency control, when a transaction modifies, inserts, updates, or deletes a record it gets a write lock on that record. Write locks are exclusive only one transaction can hold a write lock at any one time. That lock alone is sufficient to keep two transactions from changing the same record at the same time and satisfies the lowest generally recognized level of concurrency - no "dirty" writes. A dirty write could happen like this:

Transaction A: reads an employee record and increases the salary.
Transaction B: reads the same employee record and gives the employee a promotion.

If the two updates run at the same time, the result without write locks could easily be that the employee gets either the salary raise or the promotion, but not both.

Read locks

When a transaction in a locking database reads a record, it gets a read lock on that record. Read locks are compatible with other read locks, but not compatible with write locks. Read locks prevent dirty reads.

A dirty read allows a transaction to see the results of an uncommitted concurrent transaction.

Transaction A: reads an employee record and increases the salary.
Transaction B: reads the same record and adds the salary to the departments budget report.
Transaction A: rolls back. Transaction B has the wrong total for the department budget.

Consistent read

Transactions running alone in a database always see the same state of data, plus any changes they make themselves. That state is called "consistent read" if a transaction reads the same record twice, it sees the same data unless it changed the data itself. If a transaction running alone in a database reads all the records in a table once, it will see exactly the same number of records with the same contents the next time it reads the table, give or take changes it makes itself. Write and read locks alone do not produce consistent reads. Consider this case:

Transaction A: counts the number of employees in department Z, locking every employee record for read.

Either

Transaction B: stores a record for a new employee in department Z, with a write lock on the record. Or
Transaction B: updates an existing employee record changing the department to Z.
Transaction B: commits and releases all its locks.
Transaction A: counts the number of employees and gets a different total.

To insure that its reads are repeatable, Transaction A has to lock something more than the existing records, something more abstract. Those abstract locks are called predicate or existence locks, locks that keep something new from being added to a result set, either by inserting a new record or modifying an existing record so it meets the criteria for the result set.

Predicate locks can be implemented as locks on the access paths to records.

If the department is an indexed field, Transaction A would acquire a read lock on that part of the index that points to records for department Z. Then when Transaction B tried to create a new index entry for its record, it would find a conflicting lock and wait for A to complete and release its lock.

If the department field is not indexed, Transaction A acquires a read lock on the entire employee table including the ability to add new records to the table. No employee records can be inserted, updated, or deleted until Transaction A completes.

Serializability

Holding two-phase write, read, and predicate locks produces serializable transactions. However, it also produces large lock tables, contention, and deadlocks.

Lock table size

Even though locks are small temporary things, reading a few million records builds up a lot of locks. For that reason, most systems that use read locks employ strategies called lock demotion and promotion.

Contention and deadlocks

A major reporting transaction that hold two-phase read locks on records and access paths can easily block all writers from the database. In turn, those writers can hold locks that block reports, causing deadlocks. The end result is that performance is often worse when transactions run concurrently using two-phase serializable locking than would be if the transactions were actually run one at a time.

Multi-version concurrency control

Firebird uses record versions in place of write locks, read locks, predicate locks, and transaction logs. Using record versions for transaction recovery is described under Record versions as an undo log.

Write locks - dirty writes

Every record version is tagged with the version of the transaction that created it. Every transaction knows what transactions are currently active. No transaction can update or delete a record whose most recent version is not committed. Dirty writes are impossible.

Read locks - dirty reads

Because records are tagged with their version and every transaction knows what transactions are currently active, no transaction can read a record version created by an active transaction. Dirty reads are impossible.

Repeatable read

Here the issue of transaction modes raises its ugly head. Firebird supports three orthogonal modes

- consistency/concurrency/read committed,
- wait/no wait,
- snapshot/no snapshot.

This paper describes the one true Firebird transaction: concurrency, wait, snapshot. Consistency transactions lock tables and are too boring to talk about. Read committed mode does not provide repeatable read because newly committed data becomes available to a running transaction. No wait transactions err as soon as then encounter any type of conflict. No snapshot transactions read only the most recently committed record, and are useful only with read committed mode.

A concurrency, wait, snapshot transaction always provides repeatable read. When the transaction starts, it creates a list of all transactions that were committed when it started, and when it encounters a record, it walks backward through the version until it finds a version whose transaction marker is on the committed list. Changes made by concurrent transactions are ignored.

Serializability

Unlike locking systems, a multi-generational concurrency control system can provide repeatable reads without being completely serializable. Here are two anomalies that affect Firebird.

Exchanges

An exchange occurs when two transactions use data from different records and apply change in inverse order. An example might help.

The problem is to be sure that all employees in the same job class have the same salary, regardless of gender. One solution is to read the records for men in each class and update the records for women with the salary from the men's records. Another, cost saving solution is to read the records for women and update the men's records to the salary from the women's records.

Transaction A: reads men, updates women
Transaction B: reads women, updates men

The result is that the salary gap is inverted, but still exists. That result could not occur if the two transactions ran separately. The transactions do not conflict because each record is modified only once. Changes made by the other transaction are not visible because when either transaction attempts to read a record that has been modified, it automatically reads the previous committed version. The solution is to be aware of the possibility of this error and choose a specific order when copying data from one record to another.

Insert anomalies

Insert anomalies are another problem than can occur during concurrent data modifications.

Consider this case.

```
Create table foo (f1 integer);
Commit;
Transaction A: insert into foo (f1) select count (*) from foo;
Transaction B: insert into foo (f1) select count (*) from foo;
Transaction A: insert into foo (f1) select count (*) from foo;
Transaction B: insert into foo (f1) select count (*) from foo;
Transaction A: insert into foo (f1) select count (*) from foo;
Transaction B: insert into foo (f1) select count (*) from foo;
Transaction A: commit;
Transaction B: commit;
Transaction A1: select f1 from foo order by f1;
0
0
1
1
2
2
```

Each transaction saw only its own changes, so each count ignored records stored by the other transaction. If the transactions were run serially, the results would have been:

```
0
1
2
3
4
5
```

The solution is to put a unique index on any data that might be stored containing the count (or max) of values in the table. Unique indexes are correctly enforced even when the transactions involved can not see each other's records.

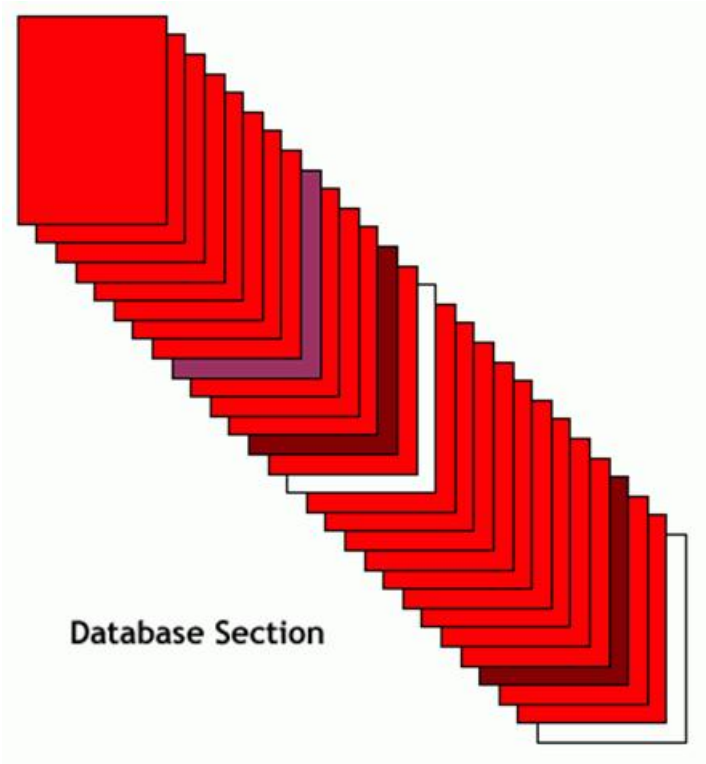
Firebird for the Database Expert - Episode 6: Why can't I shrink my databases

By Ann Harrison

New Firebird users often ask "Why doesn't the database get smaller when I delete records?" or "Where's the PACK function".

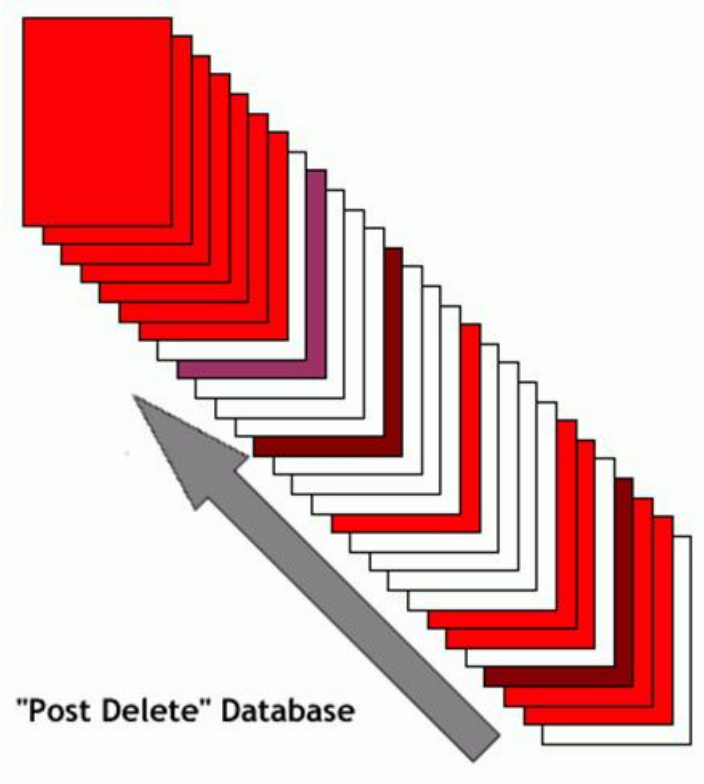
The usual answer is that releasing and reallocating space is more expensive than reusing it internally. That's true, but it's not the whole answer. The real issue is the relationships between pages, and to understand that, it helps to have some understanding of the structure of a Firebird database. There's a more complete description in Episode 2, but, briefly, a Firebird database is a single file. The file contains data for all tables, indexes, and structural information that allows Firebird to allocate and release pages, locate tables and indexes, maintain generators, etc.

The database file is made up of pages. Pages are fixed length blocks within the file. Each page has a specific function. The most common are data pages, each holds records for a single table. When you store a record in a table, Firebird first tries to store it on a data page for that file that are already in the page cache. Then, it looks for other pages belonging to the table that have space. If there is no data page for that table in the file with space for the new record, then Firebird looks for free pages - pages that have been allocated to the file that are not currently used. Finally, if all those searches fail to find a place for the record, Firebird extends the file and allocates a page in the new space.

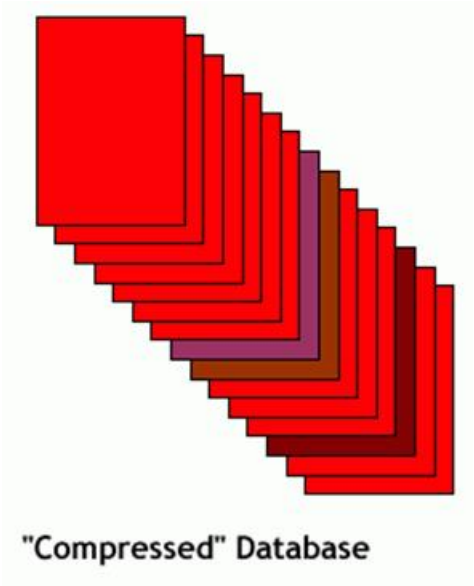


This diagram represents a section at the end of a database file. The red pages are data pages. Brown pages are index pages. The purple page is a page inventory page. The two white pages represent former data pages that are now empty. The first page cannot be released because file systems do not allow space to be removed from the middle of a file. In theory, the last page in the database could be released, by truncating the file slightly. However, the effect would be minimal unless a large number of the deleted records were on the last pages allocated. That situation is rare.

One common case of mass deletes is an application step in which records must be stored in a temporary table for processing before being inserted into their final location. In that case, pages allocated for the temporary table would precede the pages allocated for the permanent table, making truncation impossible. Another case is a rolling archive: an active table holds records for a period of time, after which they are archived to a different table or database. In that case, the deleted records would be stored before the most recent records, again preventing significant truncation. In fact, it is difficult to think of an application that stores a large number of records, and then deletes them without storing or modifying other data, aside from test databases.



One might imagine that the database with empty pages could be compacted by sliding the pages together. That thought gravely underestimates the internal linkages in a Firebird database. Pointer pages, index root pages, transaction inventory pages, and generator pages are located through a table called RDB$PAGES which would have to be updated with their new location. Pointer pages are arrays of page numbers, all of which would need to be updated to reflect the new locations of pages containing data for the tables. And those are the easy cases.



Page inventory pages - the purple page in the diagrams - occur at fixed intervals and cannot be moved. A page inventory page is an array of bits that indicate whether the corresponding page is in use. Since the correspondence is by page number, page inventories would have to be updated to reflect the new location of the pages. Within a data page, records identify their back versions and fragments by page number. Because here is no pointer back from the fragment or back version, if a page containing a fragment is moved, the system would need to search the whole table to find the record that owns the fragment and fix its pointer.

Indexes pages point to their left and right neighbors by page number, and upper levels reference lower levels by pages number. At the bottom level, the index indicates the location of records by page number. Moving data or index pages would invalidate the whole index.

To summarize, there is no simple way to release all free space in a database to the operating system because free pages do not typically congregate at the end of the database file. The internal structure of the database file is so complex that any effort to compact the file would require taking the database off line for longer than a backup and restore, with less satisfactory results.

Structure of a header page

By Ann Harrison

A Firebird database has one header page per file, but the first one is by far the most important. When Firebird opens a database, it reads the first 1024 bytes of the file to determine whether the file is actually a database, whether its format (i.e. On Disk Structure or ODS) is one that the current engine understands, the size of a database page, whether the database is read/only, whether forced writes are required, and many other important bits of information. Subsequent header pages contain only the page number of the header of the next file, the sequence number of this file in the database, and the name of the next file.

The individual fields on the primary header page are:

Field type	Size in bytes	Function
hdr_header	16	This structure is defined on every page and includes the information below.
hdr_page_size	2	Length of a database page in bytes.
hdr_ods_version	2	Major and minor On Disk Structure version number.
hdr_PAGES	4	The page number of the first pointer page for the RDB$PAGES table. The format format of the RDB$PAGES table is fixed for any ODS. The first pointer page allows the system to read the RDB$PAGES table and find all other parts of the metadata.
hdr_next_page	4	Page number of the header page of the next file in the database.
hdr_oldest_transaction	4	Oldest uncommitted transaction, whether rolled back, limbo, or active.
hdr_oldest_active	4	Oldest transaction active when any active transaction started.
hdr_next_transaction	4	Transaction id to be assigned to the next transaction when it starts.
hdr_sequence	2	Sequence number of this file in the database.
hdr_flags	4	Flag settings, see below.
hdr_creation_date	8	Timestamp of database creation.
hdr_attachment_id	4	Identifier to assign to the next connection.
hdr_shadow_count	4	Event count for shadow synchronization.
hdr_implementation	2	Implementation number of the database engine which created the database.
hdr_ods_minor	2	Current minor on disk structure version number.
hdr_ods_minor_original	2	Minor on disk structure version at the time of database creation.
hdr_end	2	Offset of the last entry in the variable length portion of the header.
hdr_page_buffers	4	Maximum number of pages in the database cache.
hdr_bumped_transaction	4	Unused, part of the abandoned write-ahead log.
hdr_oldest_snapshot	4	Confusing and redundant variant of oldest active.
hdr_backup_pages	4	Number of pages in files locked for backup (NBAK?).
hdr_misc	12	Stuff to be named later, present for alignment, I think.
hdr_data[1]	1	Clumplet data.

Header Page Clumpets

Clumplets are optional extensions of the header information and start at the end of the fixed portion of the header. Clumplet data items have the format:

```
<type_byte>  <length_byte>  <data...>
```

New clumplet types can be added without invalidating the on disk structure because the engine skips unrecognized clumplets.

Clumplet name	Value	Meaning
HDR_end	0	Last clumplet in the header.
HDR_root_file_name	1	Original name of root file.
HDR_journal_server	2	Name of journal server.
HDR_file	3	Secondary file.
HDR_last_page	4	Last logical page number of file.
HDR_unlicensed	5	Count of unlicensed activity.
HDR_sweep_interval	6	Transactions between sweeps.
HDR_log_name	7	Replay log name.

HDR_journal_file	8	Intermediate journal file.
HDR_password_file_key	9	Key to compare to password db.
HDR_backup_info	10	WAL backup information.
HDR_cache_file	11	Shared cache file – unused.
HDR_max	11	Maximum HDR_clump value.

Standard Page Header

Every page in the database starts with the standard page header, containing the following fields. The values present in the standard header for the first header page of a database are listed.

Field type	Size in bytes	Function
page type	1	Value 1 meaning header page.
page flags	1	Not used for header pages.
page checksum	2	The value 12345.
page generation	4	A value incremented each time the page is written.
page sequence number	4	Reserved for future use.
page offset	4	Reserved for future use.

Header Page Flags

Possible settings for the flag field in the database header:

Flag name	Hex value	Decimal value	Meaning
hdr_active_shadow	0x1	1	File is an active shadow file.
hdr_force_write	0x2	2	Forced writes are enabled if this flag is set.
hdr_short_journal	0x4	4	Short-term journaling. Part of an abandoned journaling subsystem.
hdr_long_journal	0x8	8	Long-term journaling. Part of an abandoned journaling subsystem.
hdr_no_checksums	0x10	16	Don't calculate checksums. Checksums are no longer calculated.
hdr_no_reserve	0x20	32	Don't reserve space on each page for record versions created by updates and deletes.
hdr_disable_cache	0x40	64	Disable shared cache file. Another abandoned project.
hdr_shutdown	0x80	128	Database is shutdown.
hdr_SQL_dialect_3	0x100	256	Database SQL dialect 3.
hdr_read_only	0x200	512	Database in *ReadOnly*. If not set, DB is RW.

See also:
Structure of a data page
Firebird for the database expert: Episode 2 - Page Types

Structure of a data page

By Paul Beach
(With thanks to Dave Schnepper and Deej Bredenberg)

A database is considered to be a collection of pages, each page has a pre-defined size, this size is determined when the database is created by a database parameter that is passed in the `isc_database_create` call (`gds_dpb_page_size`). Pages are identifed by a page number (4 byte unsigned integer), stating at 0 and increasing sequentially from the beginning of the first database file to the end of the last database file.

Page 0 of a database is always the database header page, which contains the information that is needed when you attach to a database. Page 1 is the first PIP page (Page Inventory Page) and the first WAL page is always page 2. By convention, page 3 is the first pointer page for the `RDB$PAGES` relation, but that location is described on the header page so it could (in theory) change.

Except for the header page there is no specific relationship beetween a page number and the type of data that could be stored on it.

The types of pages are defined in `ods.h` and are as follows:

```
#define pag_header 1           /* Database header page */
#define pag_pages 2            /* Page inventory page */
#define pag_transactions 3     /* Transaction inventory page */
#define pag_pointer 4          /* Pointer page */
#define pag_data 5             /* Data page */
#define pag_root 6             /* Index root page */
#define pag_index 7            /* Index (B-tree) page */
#define pag_blob 8             /* Blob data page */
#define pag_ids  9             /* Gen-ids */
#define pag_log  10            /* Write ahead log information */
```

Pages are located in the database by seeking within the database file to position `page_number*bytes_per_page`. The structure of a data page, as defined in `ods.h` is as follows:

All pages have a page header, the page header consists of,

```
typedef struct pag {
   SCHAR pag_type;
   SCHAR pag_flags;
   USHORT pag_checksum;
   ULONG pag_generation;
   ULONG pag_seqno;     /* WAL seqno of last update */
   ULONG pag_offset;    /* WAL offset of last update */
} *PAG
```

1	2	Length, bytes	Description
`pag_type`	Page Type	1	=`pag_data`
`pag_flags`	Page Flags	1	e.g. Data page is orphaned (it doesn't appear on any pointer page), page is full, or a blob or an array exist on the page.
`pag_checksum`	Page Checksum	2	Always `12345` for known versions.
`pag_generation`	Page Generation	4	how many times has the page been updated.
`pag_seqno`	Page Sequence Number	4	WAL sequence number of last update, unused.
`pag_offset`	Page Offset	4	WAL offset of last update, unused.

The remainder of the page (less the 16 bytes above) is used to store page-specific data.

A data page holds the actual data for a table, and a data page can only be used by a single table, i.e. it is not possible for data from two different tables to appear on the same data page. Each data page holds what is basically an array of records (complete or fragmented). Below the header is 8 bytes of:

- **Page Sequence** (`dpg_sequence` 4 bytes) sequence number of the data page in a table, used for integrity checking.
- **Page's Table/Relation id** (`dpg_relation` 2 bytes) this id is also used for integrity checking.
- **Number of Records** or record fragments that exist on the data page (`dpg_count` 2 bytes).

This is then followed by an array of descriptors each of the format: offset of record or fragment, length of record or fragment. This descriptor describes the size and location of records or fragments stored on a page. For each record or fragment that is stored on the page there is an equivalent record descriptor at the top of the page. As records get stored the array grows down the page, whilst the records or fragments are inserted backwards from the end of the page. The page is full when they meet in the middle.

```
typedef struct dpg {
      struct pag dpg_header;
      SLONG dpg_sequence;    /* Sequence number in relation */
      USHORT dpg_relation;   /* Relation id */
      USHORT dpg_count;      /* Number of record segments on page */
      struct dpg_repeat
      {
      USHORT dpg_offset;     /* Offset of record fragment */
      USHORT dpg_length;     /* Length of record fragment */
      }  dpg_rpt [1];
  } *DPG;
```

Obviously data records can vary in size, so the number of records that may fit on a page can vary. Equally records may get deleted, leaving gaps on a page.

The page free space calculation works by looking at the size of all of the records that exist on a page. If space can be created on the page for a new record, then the records will get compressed i.e. shifted downwards to fill the gaps that would get created during normal insert, update and deletion of data. When the free space is less than the size of the smallest possible fragment - then the page is full.

A record may be uniquely identified by its record number (`rdb$db_key`).

The record header structure is,

1	Length, bytes	Description
`rhd_transaction`	4	Record header transaction. The transaction id that wrote the record.
`rhd_b_page`	4	Record header back pointer. Page number of the back version of the record.
`rhd_b_line`	2	Record header back line. Line number of the back version of the record.
`rhd_flags`	2	Record header flags. Possible flags are:
		• `rhd_deleted` - the record has been logically deleted, but hasn't yet been garbage collected.
		• `rhd_chain` - this record is an old version, a later version points backwards to this one.
		• `rhd_fragment` - the record is a fragment of a record.
		• `rhd_incomplete` - the initial part of the record is stored here, but the rest of it may be stored in one or multiple fragments.
		• `rhd_blob` - the record stores data from a blob.
		• `rhd_stream_blob` - the record stores data from a stream blob.
		• `rhd_delta` - the prior version of this record must be obtained by applying the differences to the data stored in this array.
		• `rhd_large` - this is a large record object such as a blob or an array.
		• `rhd_damaged` - the record is known to be corrupt.
		• `rhd_gc_active` - the record is being garbage collected as an unrequired record version.
`rhd_format`	1	Record header format. The metadata version of the stored record. When a record is stored or updated, it is marked with the current format number for that table. A format is a description of the number and physical order of fields in a table and the datatype of each field.

When a field is added or dropped, or the datatype of a field is changed, a new format is generated for that table. A history of all of the formats for a table is stored in `RDB$FORMATS`. This allows the database to reconstruct records that were stored at any time based on the format that existed for the table at that time. Metadata changes, such as the above do not directly affect the records when the metadata change itself takes place, only when the records are actually next visited.

Record header data (`hd_data` size `n` as needed) is the actual record data and is compressed by RLE (Run Length Encoding). When a run takes place the compression algorithm will use 1 extra byte per 128 bytes, to represent the run length followed by one or more bytes of data. A positive run length indicates that the next sequence of bytes should be read literally, whilst a negative run length indicates that the following byte is to be repeated `ABS(n)` times.

```
typedef struct rhd {
    SLONG rhd_transaction;    /* transaction id */
    SLONG rhd_b_page;         /* back pointer */
    USHORT rhd_b_line;        /* back line */
    USHORT rhd_flags;         /* flags, etc */
    UCHAR rhd_format;         /* format version */
    UCHAR rhd_data [1];
} *RHD;
```

This paper was written by Paul Beach in September 2001, and is copyright Paul Beach and IBPhoenix Inc.

See also:
Structure of a header page
Firebird for the database expert: Episode 2 - Page Types

Garbage Collectors

By Ann Harrison

It is no longer true that "every" transaction participates in garbage collection. In the olden days, before InterBase 5, all garbage collection was cooperative. Each transaction looked at each record it read, and if it found unnecessary back versions, stopped whatever it was doing and removed them.

That behavior had the "unfair" effect of charging a transaction that did not change the database with lots of I/O spent cleaning up after transactions that did make changes. In V6, InterBase introduce a "garbage collect thread" for SuperServer only.

When the garbage collect thread is enabled, transactions identify unneeded back versions and put them on a list to be removed. When the system is idle, a special thread starts, reads the list, and starts cleaning up. The theory was that garbage collection would happen during slow times and not affect performance. Like many theories, this one has a flaw. Garbage collection is cheap if the back version to be removed is on the same page with the version of the record that is staying. There's only one page to change, and there are no tricky interactions with careful write. Normally, back versions are stored on the same page with the most recent record version. If that page fills up, then back versions need to go elsewhere, and the cost of storing and removing them increases enormously.

So, in a busy system, the garbage collect thread doesn't run often enough, back versions accumulate, and performance degrades markedly.

Vulcan disabled the garbage collect thread and performance is more even. Firebird 2 implements a hybrid mode for SuperServer in which threads remove back versions themselves if the back version is on the same page with the primary record version. If not, the record goes on a list for the garbage collector. At some point, we'll test the various methods and pick the one that works best under load.

See also:
Garbage collection
Garbage collection in IBExpert

Record versions as an undo log

By Ann Harrison

Firebird has no undo log or before-image journal. Instead, it uses old record versions to back out changes of transactions that fail.

When a record is changed or deleted, the system creates a back version of the record that contains enough information to transform the newer version into the previous version. The newest record version contains a link to the next older version, which may contain a link to the next older version, and so on. However, there is, at most, one uncommitted version of each record.

When a transaction rolls back, the next older version of each record it changed is the undo log for that record. A transaction that rolls back under program control undoes its own actions. If the transaction cannot undo its own actions, its changes are undone through cooperative garbage collection. When a transaction encounters a record version created by a transaction that failed, the active transaction removes that record version and replaces it with the previously committed version of the record.

See also:
OAT (Oldest Active Transaction) OIT (Oldest Interesting Transaction)

Where do data pages come from?

By Ann Harrison

A Firebird database is an array of fixed-length pages in no particular order. How does the engine determine where a record should be stored?

Records are stored on data pages. When the engine prepares to store a record, it first compresses the record, then looks for a data page with available space.

1. Often, when a table is active, there is a suitable page in cache, already allocated to the right table for the record, with space for the new record, and nothing special must be done.
2. If not, the system first checks the current pointer page for the table, checks the array at the bottom to find the first page that isn't full, reads that page, and puts the record there.
3. If the current pointer page doesn't have a page with free space, the system checks subsequent pointer pages for data pages that can hold the new record.
4. If a new page must be allocated,
    a. The engine finds the current page inventory page (PIP), looks on its header to find the first free page,
        i. If there are no free pages on the PIP, check the next PIP, until one has space, or the last one is found.
        ii. If there is only one free page on the last PIP, use it to allocate another PIP.
    b. The system changes the state of the next bit on the PIP that represents a free page page.
    c. marks the PIP has having been changed.
    d. and formats a buffer to look like a data page.

Once a page with sufficient space has been found, the engine locates a block of space for the record and an empty page index, if one is available, or creates a new page index. It then puts the lenght of the compressed record and its offset on the data page into the page index.

See also:
Structure of a data page

Optimize database cache utilization to improve database performance

By Holger Klemt

Did you ever think about possibilities to improve your database performance? Sure, a database system such as InterBase or Firebird is able to speed up typical operations internally but, in a lot of cases, there are very easy but powerful methods to improve performance.

Here is a first example:

When the first user connects to a database, the database cache is empty and all database and index pages must be read from the hard disk. The Superserver architecture will use the cache for all connected users for this database, but when the users are disconnected again, the cache is cleared and everything starts over again.

This is not only important for typical Delphi/C++/.net/Java client applications, but also for web server applications using PHP or ASP.

How to improve the database open performance?

1. Use available memory as cache. The cache setting for a specific database can be changed in the IBExpert menu item Tools / Database Properties / Buffers / Pages. Maximum values depend on the used InterBase/Firebird server version, but Firebird 2.0 supports up to 128k (131072) pages here.
2. Use a large page size. Firebird 2.0 can be used with a 16k page size, so 131072 pages cache means about 2 GB ram is used as cache. When using an 8k page size, the maximum ram is 1 GB etc. To change the page size, just perform a backup and then restore with the changed page size.
3. *Important:* Do not set this combination higher than the free available physical memory on your database server. It should also not be much higher than the database file size.
4. How to fill the cache? When daily work starts, for example at 8:00am, it might be helpful to have the cache already filled before the employees start their work. For this reason, we create a simple stored procedure:

   CREATE PROCEDURE FILLCACHE AS declare variable SQL VARCHAR(200); declare variable cnt integer; BEGIN

```
/* Fillcache Procedure (c) IBExpert Team*/
FOR
  select rdb$relation_name sql from rdb$relations
  INTO :SQL
DO
BEGIN
  sql='select count(*) from '||sql;
  execute statement sql into cnt;
END
```

   END

This procedure is compatible with firebird >=1.5, but it can be also altered to be implemented with InterBase or older Firebird versions. Since it counts all data in all tables, all data pages are copied from the hard disk to the cache. When there is enough free memory, all cache pages remain in the memory until the last connection disconnects.

This script should be executed, for example, the first time every morning at 7:30 am. Write a batch file and create a job in the Windows Task Manager or Linux cron:

```
connect 'localhost:C:\db1.fdb' user 'sysdba' password 'masterkey';
execute procedure fillcache;
commit;
shell sleep 3600000
execute procedure fillcache;
commit;
shell sleep 3600000
execute procedure fillcache;
commit;
shell sleep 3600000
......

exit;
```

This script connects to the database, executes the `fillcache` procedure, commits the transaction and sleeps for one hour before it runs again. The operation is repeated as often as desired and the connections remain active until the command `exit` is executed. For example when executed hourly 12 times, it fills the cache for twelve hours and stops after that time. On the next day, the script starts again automatically.

5. Additional advantages: this script also starts the garbage collector when it finds outdated records in the database, but this will only happen as long as there is no older active transaction (OAT) blocking the garbage collector.

6. Resume

Feel free to implement these operations in your database server to improve the performance. We have a number of customers who have used this and reported very satisfactory improvements.

Selecting the right datatype to improve database performance

By Holger Klemt

Here is a further example of just one more method to improve your database performance: use the right datatype!

We were set the challenge to find out how much influence the changes between GUID and Int32 or Int64 primary keys have in the database design regarding performance. So we created 3 different databases on a Windows machine, each with two simple tables (`m` for master, `d` for detail).

Here is the database structure for Int32 IDs:

```
CREATE TABLE M (
    ID   INTEGER NOT NULL PRIMARY KEY,
    TXT  VARCHAR(30));

CREATE TABLE D (
    ID    INTEGER  NOT NULL PRIMARY KEY,
    M_ID  INTEGER REFERENCES M(ID),
    TXT  VARCHAR(30));
```

Here is the database structure for Int64 IDs:

```
CREATE TABLE M (
    ID   BIGINT NOT NULL PRIMARY KEY,
    TXT  VARCHAR(30));

CREATE TABLE D (
    ID    BIGINT  NOT NULL PRIMARY KEY,
    M_ID  BIGINT REFERENCES M(ID),
    TXT  VARCHAR(30));
```

Here is the database structure for GUIDs:

```
CREATE TABLE M (
    ID   CHAR(32) NOT NULL PRIMARY KEY,
    TXT  VARCHAR(30));

CREATE TABLE D (
    ID    CHAR(32)  NOT NULL PRIMARY KEY,
    M_ID  CHAR(32) REFERENCES M(ID),
    TXT  VARCHAR(30));
```

To create the database for the GUID, we used a UDF from http://www.ibexpert.com/download/udf/uuidlibv12.zip.

```
DECLARE EXTERNAL FUNCTION GUID_CREATE
    CSTRING(36) CHARACTER SET NONE
    RETURNS PARAMETER 1
    ENTRY_POINT 'fn_guid_create' MODULE_NAME 'uuidlib';
```

Next we created a stored procedure to generate the data in the GUID database.

```
CREATE PROCEDURE INITDATA (ANZ INTEGER)
AS
declare variable m varchar(40);
declare variable d varchar(40);
declare variable dx integer;
begin
  while (anz>0) do
  begin
    m=guid_create();
    m=strreplace(m,'-','');
    insert into m(id,txt) values (:m,current_timestamp);
    dx=10;
    while (dx>0) do
    begin
      select guid_create() from rdatabase$database into :d;
      d=strreplace(d,'-','');
      insert into d(id,txt,m_id) values (:d,current_timestamp,:m);
      dx=dx-1;
    end
    anz=anz-1;
  end
end
```

The procedure to create the `Integer ID` data is much easier using a generator.

After we created all 3 databases with the parameter `500000` (i.e. 500,000 master and 5,000,000 detail records were created), we disconnected and reconnected? again to the database to ensure that any cache influence did not alter the results.

To perform a typical SQL operation, we started a `SELECT` that joins all records from all tables:

```
select count(*) from m join d on d.m_id=m.id
```

Here are the results:

```
Operation/Info   Int32   Int64   GUID
Database Size    505 MB   550 MB   1030 MB
INITDATA(500000) 271s   275s   420s
Backup 49s   54s   90s
Restore    124s   127s   144s
Select     22s   22s   49s
```

Resume

The changes between Int64 and Int32 are negligible, but the changes to a GUID is a problematic design. The integer datatypes will give you better performance.

To discover more hints and tips about where you can improve the performance of your database, just open the IBExpert menu item Tools / Stored Procedure/ Trigger/View Analyzer and press [F9]. This analyzes all objects and displays all parts that do not use an index in a red color. To modify these objects, simply double click the line. A well-designed database should have no red line at all!

This feature is not available in the IBExpert Personal Edition, but is part of the IBExpert Trial Edition, which allows you to test all IBExpert on your database for 45 daysfunctionalities - free of charge, and which you can download from http://ibexpert.net/ibe/pmwiki.php?n=Main.DownloadTrial (scroll down to download the `setup_trial.exe` file).

The IBExpert Full Version gives you unlimited access to these performance-tuning tools and is available for just EUR 179.00 at http://ibexpert.net/ibe/pmwiki.php?n=Main.OnlineShop.

SQL Language Reference

Here is some basic information regarding DDL, DML and stored procedure and trigger language. Refer to the *InterBase SQL Language Reference* handbook for detailed information concerning InterBase syntax, and we recommend Helen Borrie's book, the *Firebird Book - a Reference for Database Developers*, for detailed information concerning Firebird 1.5. A complete SQL Reference is currently being prepared for Firebird 2.0 - the current preview can be found here at this documentation site: *Firebird 2 SQL Reference Guide*.

Please also refer to the IBExpert Tools menu: Script Executive / Script Language Extensions for IBExpert's own invaluable extensions, and the IBEBlock documentation. IBEBlock is a set of DDL, DML and other statements which include some specific constructions applicable only in IBExpert or IBEScript.

Structured Query Language

SQL is the abbreviation for Structured Query Language. It is used to communicate with a relational database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. It serves to define, manipulate, find and fetch data in a database.

InterBase and Firebird conform closely to the international industrial standards SQL '92. There were a number of features introduced in Firebird 1.5 which comply to the more recent SQL-99 standard.

Furthermore InterBase and Firebird offer a series of additional SQL enhancements, such as generators, triggers and stored procedures, allowing a more extensive modeling and manipulation of data. These enhancements are either based on the ANSI SQL2 Standard or already comply with the outline of the ANSI/ISO SQL3 standards.

DSQL - Dynamic SQL

DSQL is the subset in most common use today. It allows a program to create statements at run time. It can be used from conventional languages through the InterBase API. More often, it is used from modern development environments such as Delphi, which hide the mechanics of the API. A completed DSQL statement is very much like the "embedded" language, without the EXEC SQL and without the terminating semicolon.

ESQL - Embedded SQL

The embedded form of SQL is used in programs written in traditional languages such as C and Pascal, started by the EXEC SQL statement. A preprocessor turns SQL statements into host language data structures and calls to the InterBase server. The embedded language is written into the program; its statements cannot be generated dynamically. Statements in embedded SQL are terminated with a semicolon.

ESQL is invalid in stored procedures and triggers (just as procedure language (PSQL) ist not valid in ESQL); it can however execute stored procedures.

For further information, please refer to the Borland *InterBase 6.x Embedded SQL Guide*.

isql - Interactive SQL

ISQL is a command-line utility program which can be used to run SQL queries on the database. ISQL supports data definitions and data manipulation commands as well as SQL scripts with multiple SQL commands within one script. It can be used to create and modify the database's metadata, insertion, alteration and deletion of data, data queries and the display of results (all this can be done in the IBExpert SQL Editor), adding and removal of user database rights (see IBExpert User Manager and Grant Manager) and execution of other database administrative functions. It is very similar to DSQL, with some omissions, such as cursors, and a few additions, for example, SET and SHOW.

ISQL commands end with ;. Each command must be explicitly committed using the commit statement.

PSQL - Stored Procedure and Trigger Language

Please refer to the stored procedure and trigger language section for further information.

SQL Dialect

Structured Query Language is a language for [IBExpert Database menu | relational databases]], which serves to define, manipulate, find and fetch data in a database.

There are currently two SQL dialects used with InterBase and Firebird:

`Dialect 1` = database performance is fully compatible to InterBase 5.6 and earlier (e.g. numeric up to 15 digits). `Dialect 3` = all new functions in InterBase 6 and upwards with SQL 92 features are available (e.g. numeric up to 18 digits).

For those that work with the BDE, this can only work with dialect 1 up to and including Delphi 6 (i.e. dialect 3 from Delphi 7 onwards).

Differences between dialects 1 and 3 include:

- The numeric (15 or 18) size.
- Large exact numerics: `DECIMAL` and `NUMERIC` data types with precision greater than 9 are stored as `INT64` instead of `DOUBLE PRECISION`.
- The double quote (") has changed from a synonym for the single quote (') to the delimiter for an object name.
- `date` and `TIME` data types have altered:
  - `Dialect 1` = `Date` includes the date and time
  - `Dialect 3` = `Date` = date, `time` = time, `timestamp` = date and time.

For new projects it is recommended that dialect 3 be specified.

Occasionally the question arises "What about SQL Dialect 2?". Dialect 2 is similar to dialect 1, generates however warnings for all objects that are incompatible to Dialect 3 (i.e. only suitable for the client end); therefore, in principle, not really of importance.

The SQL dialect to be used in a database is specified when creating the database (IBExpert menu: Database / Create Database). It can subsequently be altered using the IBExpert menu Services / Database Properties (although watch out for possible dialect incongruencies, for example, the different date and time types).

See also:
Structured Query Language
`SET SQL DIALECT`

Query

A query is a qualified search for information held in the data sets stored in the database. The qualification can determine which tables should be searched, which range of values for specified columns should be included, etc.

For an overview of the conditions that are available in SQL, please refer to Comparison Operators.

`SUM` (total), `MIN` (minimum), `MAX` (maximum), `AVG` (average), and `COUNT` are aggregates that can also be used, for example, when the sales department needs to know how many orders are still open or the minimum/maximum or average order value in the past year.

A query on one or more tables produces a set of rows that is itself a table, subject to all the rules for tables in a relational database. This is known as *Closure*. InterBase/Firebird fully supports closure.

Regularly performed queries, such as a list of all unpaid invoices, or a list of all delivery notes that have gone out in the last week, can be stored as procedures.

Queries are optimized by InterBase/Firebird. The optimizer chooses which indices should be used, in order to perform the query as quickly and simply as possible.

Symbols and brackets used in code syntax

For those users new to SQL: in the notation used in this section (and generally in all Firebird and InterBase literature), the following symbols, punctuation and brackets have the following meaning:

`()`	round brackets	Elements of the syntax.
`,`	comma	Elements of the syntax.
`{}`	curly braces/brackets	Not part of the syntax; indicate mandatory phrases.
`[]`	square brackets	Not part of the syntax; indicate optional phrases.
`|`	pipe symbol	Not part of the syntax; indicates mutually exclusive options.

Comparison Operators

Comparison operators for use in conditional clauses:

Conditional Test	Description
`value = value`	Equal to
`value < value`	Less than
`value > value`	Greater than
`value <= value`	Less than or equal to
`value >= value`	Greater than or equal to
`value !< value`	Not less than
`value !> value`	Not greater than
`value <> value`	Not equal to
`value != value`	Not equal to
`value LIKE value`	Wildcard search, use '%' for 0 or more characters and '_' for one character only
`value BETWEEN value AND value`	Within an inclusive range
`value IN (value, ... value)`	One of the elements in a list
`value IS NULL`	One of the elements in a list
`value IS NOT NULL`	One of the elements in a list
`value CONTAINING value`	Includes
`value STARTING WITH value`	Begins with

See also:
Conditional Test Operator

Firebird SQL

Every database management systemhas its own idiosyncrasies in the ways it implements SQL. Firebird adheres to the SQL standard more rigorously than any other RDBMS except possibly its 'cousin', InterBase®. Developers migrating from products that are less standards-compliant often wrongly suppose that Firebird is quirky, which is really not true at all.

The following excerpts have been taken from the *Firebird 2 Quick Start Guide*, ©IBPhoenix Publications 2008.

Division of an integer by an integer

Firebird accords with the SQL standard by truncating the result (quotient) of an integer/integer calculation to the next lower integer. This can have bizarre results unless you are aware of it. For example, this calculation is correct in SQL:

```
1 / 3 = 0
```

If you are upgrading from an RDBMS which resolves integer/integer division to a float quotient, you will need to alter any affected expressions to use a float or scaled numeric type for either dividend, divisor, or both. For example, the calculation above could be modified thus in order to produce a non-zero result:

```
1.000 / 3 = 0.333
```

Things to know about strings

String delimiter symbol

Strings in Firebird are delimited by a pair of single quote (apostrophe) symbols: `'I am a string'` (ASCII code 39, not 96). If you used earlier versions of Firebird's relative, InterBase®, you might recall that double and single quotes were interchangeable as string delimiters. Double quotes cannot be used as string delimiters in Firebird SQL statements.

Apostrophes in strings

If you need to use an apostrophe inside a Firebird string, you can "escape" the apostrophe character by preceding it with another apostrophe. For example, this string will give an error:

```
'Joe´s Emporium'
```

because the parser encounters the apostrophe and interprets the string as 'Joe' followed by some unknown keywords. To make it a legal string, double the apostrophe character:

```
'Joes´´ Emporium'
```

Notice that this is TWO single quotes, not one double-quote.

Concatenation of strings

The concatenation symbol in SQL is two "pipe" symbols (ASCII 124, in a pair with no space between). In SQL, the "+" symbol is an arithmetic operator and it will cause an error if you attempt to use it for concatenating strings. The following expression prefixes a character column value with the string "`Reported by: `":

```
'Reported by: ' || LastName
```

Firebird will raise an error if the result of a string concatenation exceeds the maximum (var)char size of 32 Kb.

If only the potential result – based on variable or field size – is too long you'll get a warning, but the operation will be completed successfully. (In pre-2.0 Firebird, this too would cause an error and halt execution.)

See also the section below, Expressions involving `NULL`, about concatenating in expressions involving `NULL`.

Double-quoted identifiers

Before the SQL-92 standard, it was not legal to have object names (identifiers) in a database that duplicated keywords in the language, were case-sensitive or contained spaces. SQL-92 introduced a single new standard to make any of them legal, provided that the identifiers were defined within pairs of double-quote symbols (ASCII 34) and were always referred to using double-quote delimiters.

The purpose of this "gift" was to make it easier to migrate metadata from non-standard RDBMSes to standards-compliant ones. The down-side is that, if you choose to define an identifier in double quotes, its case-sensitivity and the enforced double-quoting will remain mandatory.

Firebird does permit a slight relaxation under a very limited set of conditions. If the identifier which was defined in double-quotes:

1. was defined as all upper-case,
2. is not a keyword, and
3. does not contain any spaces,

...then it can be used in SQL unquoted and case-insensitively. (But as soon as you put double-quotes around it, you must match the case again!)

Warning: Don't get too smart with this! For instance, if you have tables `"TESTTABLE"` and `"TestTable"`, both defined within double-quotes, and you issue the command:

```
SQL>select * from TestTable;
```

...you will get the records from `"TESTTABLE"`, not `"TestTable"`!

Unless you have a compelling reason to define quoted identifiers, it is usually recommended that you avoid them. Firebird happily accepts a mix of quoted and unquoted identifiers – so there is no problem including that keyword which you inherited from a legacy database, if you need to.

Warning: Some database admin tools enforce double-quoting of all identifiers by default. Try to choose a tool which makes double-quoting optional.

Expressions involving `NULL`

In SQL, `NULL` is not a value. It is a condition, or *state*, of a data item, in which its value is unknown. Because it is unknown, `NULL` cannot behave like a value. When you try to perform arithmetic on `NULL`, or involve it with values in other expressions, the result of the operation will almost always be `NULL`. It is not zero or blank or an "empty string" and it does not behave like any of these values.

Below are some examples of the types of surprises you will get if you try to perform calculations and comparisons with `NULL`.

The following expressions all return `NULL`:

- `1 + 2 + 3 + NULL`
- `not (NULL)`
- `'Home ' || 'sweet ' || NULL`

You might have expected 6 from the first expression and "Home sweet " from the third, but as we just said, `NULL` is not like the number 0 or an empty string – it's far more destructive!

The following expression:

- `FirstName || ' ' || LastName`

will return `NULL` if either `FirstName` or `LastName` is `NULL`. Otherwise it will nicely concatenate the two names with a space in between – even if any one of the variables is an empty string.

Tip: Think of `NULL` as *UNKNOWN* and these strange results suddenly start to make sense! If the value of `Number` is unknown, the outcome of `'1 + 2 + 3 + Number'` is also unknown (and therefore `NULL`). If the content of `MyString` is unknown, then so is `'MyString || YourString'` (even if `YourString` is non-`NULL`). Etcetera.

Now let's examine some PSQL (Procedural SQL) examples with if-constructs:

- `if (a = b) then`

```
MyVariable = 'Equal';
else
MyVariable = 'Not equal';
```

After executing this code, `MyVariable` will be `'Not equal'` if both `a` and `b` are `NULL`. The reason is that `'a = b'` yields `NULL` if at least one of them is `NULL`. If the test expression of an "`if`" statement is `NULL`, it behaves like false: the `'then'` block is skipped, and the `'else'` block executed.

Warning: Although the expression may behave like false in this case, it's still `NULL`. If you try to invert it using `not()`, what you get is another `NULL` – not "`true`".

- `if (a <> b) then`

```
MyVariable = 'Not equal';
else
MyVariable = 'Equal';
```

Here, `MyVariable` will be `'Equal'` if `a` is `NULL` and `b` isn't, or vice versa. The explanation is analogous to that of the previous example.

The `DISTINCT` keyword comes to the rescue!

Firebird 2 implements a new use of the `DISTINCT` keyword allowing you to perform (in)equality tests that take `NULL` into account. The semantics are as follows:

- Two expressions are `DISTINCT` if they have different values or if one is `NULL` and the other isn't;
- They are `NOT DISTINCT` if they have the same value or if both are `NULL`.

Notice that if neither operand is `NULL`, `DISTINCT` works exactly like the "<>" operator, and `NOT DISTINCT` like the "=" operator.

`DISTINCT` and `NOT DISTINCT` always return `true` or `false`, never `NULL`.

Using `DISTINCT`, you can rewrite the first PSQL example as follows:

```
if (a is not distinct from b) then
MyVariable = 'Equal';
else
MyVariable = 'Not equal';
```

And the second as:

```
if (a is distinct from b) then
MyVariable = 'Not equal';
else
MyVariable = 'Equal';
```

These versions will give you the results that a normal human being (untouched by SQL standards) would expect, whether there are `NULL`s involved or not.

See also:
Firebird 2 SQL Reference Guide

Data Retrieval

The most frequently used operation in transactional databases is the data retrieval operation.

SELECT is used to retrieve zero or more rows from one or more tables in a database. In most applications, SELECT is the most commonly used DML command. In specifying a SELECT query, the user specifies a description of the desired result set, but they do not specify what physical operations must be executed to produce that result set. Translating the query into an optimal query plan is left to the database system, more specifically to the query optimizer.

SELECT

The SELECT statement has the following syntax:

Syntax InterBase 7.1

```
SELECT [TRANSACTION transaction]
       [DISTINCT | ALL]
       {* | val [, val ...]}
       [INTO :var [, :var ...]]
       FROM tableref [, tableref ...]
       [WHERE search_condition]
       [GROUP BY col [COLLATE collation] [, col [COLLATE collation] ...]
       [HAVING search_condition]
       [UNION [ALL] select_expr]
       [PLAN plan_expr]
       [ORDER BY order_list]
       [ROWS value [TO upper_value] [BY step_value][PERCENT][WITH TIES]]
       [FOR UPDATE [OF col [, col ...]]];
```

Description

SELECT retrieves data from tables, views, or stored procedures. Variations of the SELECT statement make it possible to:

- Retrieve a single row, or part of a row, from a table. This operation is referred to as a singleton select. In embedded applications, all SELECT statements that occur outside the context of a cursor must be singleton selects.
- Retrieve multiple rows, or parts of rows, from a table. In embedded applications, multiple row retrieval is accomplished by embedding a SELECT within a DECLARE CURSOR statement. In isql, SELECT can be used directly to retrieve multiple rows.
- Retrieve related rows, or parts of rows, from a join of two or more tables.
- Retrieve all rows, or parts of rows, from union of two or more tables.
- Return portions or sequential portions of a larger result set; useful for Web developers, among others.
- All SELECT statements consist of two required clauses (SELECT, FROM), and possibly others INTO, WHERE, GROUP BY, HAVING, UNION, PLAN, ORDER BY, ROWS).

Notes on SELECT syntax

- When declaring arrays, you must include the outermost brackets, shown below in bold. For example, the following statement creates a 5 by 5 two-dimensional array of strings, each of which is 6 characters long:

  my_array = varchar(6)**[**5,5**]**

  Use the colon (:) to specify an array with a starting point other than 1. The following example creates an array of integers that begins at 10 and ends at 20: `my_array = integer[20:30]`
- In SQL and isql, you cannot use val as a parameter placeholder (like "?").
- In DSQL and isql, val cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.

Important: In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in isql, the semicolon is a terminating symbol for the statement, so it must be included.

Source: InterBase 7.1 Language Reference Guide

The Firebird syntax deviates slightly from InterBase:

Syntax Firebird up to 1.5

```
SELECT
   [FIRST (m)] [SKIP (n)] [[ALL] | DISTINCT]
   <list of columns> [, [column-name] | expression | constant ] AS alias-name]
FROM <table-or-procedure-or-view]
   [{[[INNER] | [{LEFT | RIGHT | FULL} [OUTER]] JOIN}] <table-or-procedure-or-view>
   ON <join-conditions [{JOIN ...]]
   [WHERE <search-conditions>]
   [GROUP BY <grouped-column-list>]]
   [HAVING <search-condition>]
   [UNION <select-expression>[ALL]]
   [PLAN <plan_expression>]
   [ORDER BY <column-list>]
   [FOR UPDATE [OF col1 [, col2 ...]][WITH LOCK]];
```

Source: The Firebird Book by Helen Borrie

Syntax Firebird 2.0

```
<select statement> ::=
   <select expression> [FOR UPDATE] [WITH LOCK]

<select expression> ::=
   <query specification> [UNION [{ALL | DISTINCT}] <query specification>]

<query specification> ::=
   SELECT [FIRST <value>] [SKIP <value>] <select list>
   FROM <table expression list>
   WHERE <search condition>
   GROUP BY <group value list>
   HAVING <group condition>
   PLAN <plan item list>
   ORDER BY <sort value list>
   ROWS <value> [TO <value>]

<table expression> ::=
   <table name> | <joined table> | <derived table>

<joined table> ::=
   {<cross join> | <qualified join>}

<cross join> ::=
   <table expression> CROSS JOIN <table expression>

<qualified join> ::=
   <table expression> [{INNER | {LEFT | RIGHT | FULL} [OUTER]}] JOIN <table expression>
   ON <join condition>

<derived table> ::=
   '(' <select expression> ')'
```

Conclusions

- FOR UPDATE mode and row locking can only be performed for a final dataset, they cannot be applied to a subquery.
- Unions are allowed inside any subquery.
- Clauses FIRST, SKIP, PLAN, ORDER BY, ROWS are allowed for any subquery.

Notes:

- Either FIRST/SKIP or ROWS is allowed, but a syntax error is thrown if you try to mix the syntaxes.
- An INSERT statement accepts a select expression to define a set to be inserted into a table. Its SELECT part supports all the features defined for select statments/expressions.
- UPDATE and DELETE statements are always based on an implicit cursor iterating through its target table and limited with the WHERE clause. You may also specify the final parts of the select expression syntax to limit the number of affected rows or optimize the statement.

Also new to Firebird 2.0: EXECUTE BLOCK statement - The SQL language extension EXECUTE BLOCK makes "dynamic PSQL" available to SELECT specifications. It has the effect of allowing a self-contained block of PSQL code to be executed in dynamic SQL as if it were a stored procedure. For further information, please refer to EXECUTE BLOCK statement.

Clauses allowed at the end of UPDATE/DELETE statements are PLAN, ORDER BY and ROWS.

Source: Firebird 2.0.4 Release Notes

FIRST (m) SKIP (n)

<FIRST (m) and SKIP (n) are optional keywords, which can be used together or individually. They allow selection and/or the omission of the first m/n rows from the resulting data sets of an ordered set. m and n are integers or simple integer arguments (both without the brackets) or expressions (within brackets) resolving to integers. Logically it should only be used with an ordered set (specified by ORDER BY). If used, these should precede all other specifications.

DISTINCT

This suppresses all duplicate rows in the output or resulting sets, thus preventing duplicate values from being returned.

ALL

This retrieves every value which meets the specified conditions. It is also the default for the return sets, and so therefore does not need to be explicitly specified.

FROM

The FROM clause specifies a list of tables, views, and stored procedures (with output arguments) from which to retrieve data. if the query involves joining one that one structure, FROM specifies the leftmost structure. The list then needs to be completed using joins (joins can even be nested). Please refer to [@JOIN@] statement for further information.

New to Firebird 2.0: support for derived tables in DSQL (subqueries in FROM clause) as defined by SQL200X. A derived table is a set, derived from a dynamic SELECT statement. Derived tables can be nested, if required, to build complex queries and they can be involved in joins as though they were normal tables or views.

Syntax

```
SELECT
   <select list>
FROM
   <table reference list>

   <table reference list> ::= <table reference> [{<comma> <table reference>}...]

   <table reference> ::=
      <table primary>
     | <joined table>

   <table primary> ::=
      <table> [[AS] <correlation name>]
        | <derived table>

   <derived table> ::=
      <query expression> [[AS] <correlation name>]
        [<left paren> <derived column list> <right paren>]

   <derived column list> ::= <column name> [{<comma> <column name>}...]
```

Examples can be found here.

Points to Note

- Every column in the derived table must have a name. Unnamed expressions like constants should be added with an alias or the column list should be used.
- The number of columns in the column list should be the same as the number of columns from the query expression.
- The optimizer can handle a derived table very efficiently. However, if the derived table is involved in an inner join and contains a subquery, then no join order can be made.

WHERE

The WHERE clause is a filter specification, used to define or limit the rows for the return sets or which rows should be forwarded for further processing such as ORDER BY or GROUP BY.

A WHERE clause can also contain its own SELECT statement, referred to as a subquery.

```
<search_conditions> include the following:

<search_condition> = val operator {val | (select_one)}
    | val [NOT] BETWEEN val AND val
    | val [NOT] LIKE val [ESCAPE val]
    | val [NOT] IN (val [, val ...] | select_list)
    | val IS [NOT] NULL
    | val {>= | <=} val
    | val [NOT] {= | < | >} val
    | {ALL | SOME | ANY} (select_list)
    | EXISTS (select_expr)
    | SINGULAR (select_expr)
    | val [NOT] CONTAINING val
    | val [NOT] STARTING [WITH] val
    | (search_condition)
    | NOT search_condition
    | search_condition OR search_condition
    | search_condition AND search_condition
```

Please refer to Comparison Operators for a full list of valid operators.

GROUP BY

GROUP BY is an optional clause, allowing the resulting sets to be grouped and summarized by common column values into one or more groups, thus aggregating or summarizing the returned data sets. these groupings often include aggregate functions. It is used in conjunction with HAVING.

The group is formed by aggregating (collecting together) all rows where a column named in both the column list and the GROUP BY clause share a common value. The column and/or field specified must of course be groupable, otherwise the query will be rejected. Any NULL values contained in rows in the targeted column are ignored for the aggregation. So if, for example, you wish to calculate averages, you must first consider whether NULL fields should be left out of the calculation, or treated as zero (which entails a little work on the developer side with a BEFORE INSERT trigger).

Firebird 2.0 introduced some useful improvements to SQL sorting operations - please refer to Improvements in sorting in the Firebird 2.0.4. Release Notes for details.

COLLATE

Specifies the collation order for the data retrieved by the query.

Collation order in a GROUP BY clause: when CHAR or VARCHAR columns are grouped in a SELECT statement, it can be necessary to specify a collation order for the grouping, especially if columns used for grouping use different collation orders.

To specify the collation order to use for grouping columns in the GROUP BY clause, include a COLLATE clause after the column name.

Please note that it is not possible to specify a COLLATE order for Blob columns.

HAVING

The HAVING condition is optional and may be used together with GROUP BY to specify a condition that limits the grouped rows returned - similar to the WHERE clause. In fact, the HAVING clause can often replace the WHERE clause in a grouping query. Perhaps the simplest way to discern the correct use of these two clauses is to use a WHERE clause to limit rows and a HAVING clause to limit groups. The HAVING clause is applied to the groups after the set has been partitioned. A WHERE filter may still be necessary for the incoming set. To maximize perfomance it is important to use WHERE conditions to pre-filter groups and then use HAVING for filtering on the basis of the results returned (after the grouping has been done) by aggregating functions.

The HAVING clause can use the same arguments as the WHERE clause:

```
<search_conditions> include the following:

<search_condition> = val operator {val | (select_one)}
   | val [NOT] BETWEEN val AND val
   | val [NOT] LIKE val [ESCAPE val]
   | val [NOT] IN (val [, val ...] | select_list)
   | val IS [NOT] NULL
   | val {>= | <=} val
   | val [NOT] {= | < | >} val
   | {ALL | SOME | ANY} (select_list)
   | EXISTS (select_expr)
   | SINGULAR (select_expr)
   | val [NOT] CONTAINING val
   | val [NOT] STARTING [WITH] val
   | (search_condition)
   | NOT search_condition
   | search_condition OR search_condition
   | search_condition AND search_condition
```

Please refer to Comparison Operators for a full list of valid operators.

UNION

Combines the results of two or more SELECT statements, which may involve rows from multiple tables or multiple sets from the same table, to produce a single result set (read-only), i.e. one dynamic table without duplicate rows. The unified columns in each separate output specification must match by degree (number and order of columns), type (data type) and size - what is known as union compatability. Which means they must each output the same number of columns in the same left-to-right order. Each column must also be consistent throughout in data type and size. By default UNION suppresses all duplicates in the final resulting sets. The ALL option keeps identical rows separate.

New to Firebird 2.0: Please refer to Enhancements to UNION handling for improvements of the rules for UNION queries.

PLAN

Specifies the query plan, optionally included in the query statement, which should be used by the query optimizer instead of one it would normally choose.

```
<query_specification>
PLAN <plan_expr>

<plan_expr> =
   [JOIN | [SORT] [MERGE]] ({plan_item | plan_expr}
   [, {plan_item | plan_expr} ...])

<plan_item> = {table | alias}
   {NATURAL | INDEX (index [, index ...])| ORDER index}
```

where plan_item specifies a table and index method for a plan.

It tells the optimizer which , join order and access methods should be used for the query. Although the optimizer creates its own plan, and as a rule, usually selects the best method, there are situations where performance can be increased by specifying the plan yourself.

The IBExpert SQL Editor's Plan Analyzer and Performance Analysis allow the user to analyze and compare the optimizer's plan with their own.

Firebird 2.0's improvements to the PLAN clause can be referred to in the Firebird 2.0.4 Release Notes, *Improvements in handling user-specified query plans*.

ORDER BY

The ORDER BY clause is used to sort a query's return sets, and can be used for any SELECT statement which is capable of retrieving multiple rows for output. It is placed after all other clauses (except a FOR UPDATE clause, if used, or a stored procedure's INTO clause).

The InterBase 7.1 syntax is as follows:

```
order by <order_list>

where

<order_list> =
{col | int} [COLLATE collation]
[ASC[ENDING] | DESC[ENDING]]
[, order_list ...]
```

It specifies columns to order, either by column name or ordinal number in the query. Sorting items are usually columns. Ideal are indexed columns, as they are sorted much faster. A compound index may speed up performance considerable when sorting more than one column. N.B. Both columns and compound index need to be in an unbroken left-to-right sequence.

The comma-separated order_list specifies the order of the rows, complemented by ASCENDING (which is the default value, therefore it need not be explicitly specified) or DESCENDING or DESC.

If there is more than one sorting item, please note that the sorting precedence is from left to right.

The Firebird 1.5 syntax is slightly different:

```
ORDER BY <order_list>
   <list_item> = <column> | <expression> | <degree number>
                 ASC | DESC
                [NULL LAST | NULLS FIRST]
```

Since Firebird 1.5 valid expressions are also allowed as sort items, even if the expression is not ouput as a runtime column. Sets can be sorted on internal or external function expressions or correlated subqueried scalars.

Firebird 1.5 supports the placement of NULLS, if and when present. The default value is NULLS LAST (sorts all nulls to the end of the return sets. NULLS FIRST needs to be explicitly specified, if null values are to be placed first.

New to Firebird 2.0: ORDER BY <ordinal-number> now causes SELECT * expansion - When columns are referred to by the ordinal number (degree) in an ORDER BY clause, when the output list uses SELECT * FROM ... syntax, the column list will be expanded and taken into account when determining which column the number refers to. This means that, now, SELECT T1.*, T2.COL FROM T1, T2 ORDER BY 2 sorts on the second column of table T1, while the previous versions sorted on T2.COL.

Tip: This change makes it possible to specify queries like SELECT * FROM TAB ORDER BY 5.

Firebird 2.0 also introduced some useful improvements to SQL sorting operations - please refer to Improvements in sorting in the Firebird 2.0.4. Release Notes for details.

ROWS

```
ROWS value
   [TO upper_value]
   [BY step_value]
   [PERCENT][WITH TIES]
```

- value is the total number of rows to return if used by itself.
- value is the starting row number to return if used with TO.
- value is the percent if used with PERCENT.
- upper_value is the last row or highest percent to return.
- If step_value = n, returns every nth row, or n percent rows.
- PERCENT causes all previous ROWS values to be interpreted as percents.
- WITH TIES returns additional duplicate rows when the last value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with ORDER BY.

Please also refer to ROWS syntax for Firebird 2.0 syntax, description and examples.

FOR UPDATE

```
[FOR UPDATE [OF col [, col ...]]
```

Only relevant when specifying columns listed after the SELECT clause of a DECLARE CURSOR statement that can be updated using a WHERE CURRENT OF clause.

Since Firebird 1.5 an optional WITH LOCK extension can be used with or without the FOR UPDATE syntax. Recommended however only for advanced developers as this supports a restricted level of explicit, row-level pessimistic locking.

RETURNING

The RETURNING clause syntax was implemented in Firebird 2.0 for the INSERT statement, enabling the return of a result set from the INSERT statement. The set contains the column values actually stored. Most common usage would be for retrieving the value of the primary key generated inside a BEFORE-trigger.

Available in DSQL and PSQL.

Syntax Pattern

```
INSERT INTO ... VALUES (...) [RETURNING <column_list> [INTO <variable_list>]]
```

Example(s)

1.

```
INSERT INTO T1 (F1, F2)
   VALUES (:F1, :F2)
RETURNING F1, F2 INTO :V1, :V2;
```

2.

```
INSERT INTO T2 (F1, F2)
   VALUES (1, 2)
RETURNING ID INTO :PK;
```

Note:

1. The INTO part (i.e. the variable list) is allowed in PSQL only (to assign local variables) and rejected in DSQL.
2. In DSQL, values are being returned within the same protocol roundtrip as the INSERT itself is executed.
3. If the RETURNING clause is present, then the statement is described as isc_info_sql_stmt_exec_procedure by the API (instead of isc_info_sql_stmt_insert), so the existing connectivity drivers should support this feature automatically.
4. Any explicit record change (update or delete) performed by AFTER-triggers is ignored by the RETURNING clause.
5. Cursor based inserts (INSERT INTO ... SELECT ... RETURNING ...) are not supported.
6. This clause can return table column values or arbitrary expressions.

See also:
Firebird 2.0.4 Release Notes: RETURNING clause for insert statements
INSERT INTO ... DEFAULT VALUES
SELECT
RETURNING
UPDATE OR INSERT
DCL - DataControlLanguage
DDL - Data Definition Language
DML -Data Manipulation Language
SQL basics

DML - Data Manipulation Language

DML is the abbreviation for Data Manipulation Language. DML is a collection of SQL commands that can be used to manipulate a database's data.

DML is part of the SQL language commands, which execute queries with database objects and changes to their contents. The various DML commands can be used to create, edit, evaluate and delete data in a database. DML commands are a subarea of SQL; the range of the SQL language is composed of DML and DDL together.

SIUD

SIUD is the abbreviation for SELECT, INSERT, UPATE, DELETE, which are the four DML commands used for data manipulation.

See also:
Create SIUD Procedures
INSERTEX

SELECT

Please refer to SQL Language Reference / Data Retrieval / SELECT for details.

INSERT

Adds one or more new rows to a specified table. Available in gpre, DSQL, and isql.

Syntax

```
INSERT [TRANSACTION transaction] INTO object [(col [, col …])]
   {VALUES (val [, val …]) | select_expr};

<object> = tablename | viewname

<val> = {:variable | constant | expr
   | function | udf ([val [, val …]])
   | NULL | USER | RDB$DB_KEY | ?} [COLLATE collation]

<constant> = num | 'string' | charsetname 'string'

<function> = CAST (val AS datatype)
   | UPPER (val)
   | GEN_ID (generator, val)
```

Argument	Description
expr	A valid SQL expression that results in a single column value.
select_expr	A SELECT that returns zero or more rows and where the number of columns in each row is the same as the number of items to be inserted.

Notes on the INSERT statement

- In SQL and isql, you cannot use val as a parameter placeholder (like "?").
- In DSQL and isql, val cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.

Important: In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in isql, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	Description
TRANSACTION transaction	Name of the transaction that controls the execution of the INSERT.
INTO object	Name of an existing table or view into which to insert data.
col	Name of an existing column in a table or view into which to insert values.
VALUES (val [, val …])	Lists values to insert into the table or view; values must be listed in the same order as the target columns.
select_expr	Query that returns row values to insert into target columns.

Description

INSERT stores one or more new rows of data in an existing table or view. INSERT is one of the database privileges controlled by the GRANT and REVOKE statements. Values are inserted into a row in column order unless an optional list of target columns is provided. If the target list of columns is a subset of

available columns, default or NULL values are automatically stored in all unlisted columns. If the optional list of target columns is omitted, the VALUES clause must provide values to insert into all columns in the table.

To insert a single row of data, the VALUES clause should include a specific list of values to insert.

To insert multiple rows of data, specify a select_expr that retrieves existing data from another table to insert into this one. The selected columns must correspond to the columns listed for insert.

Important: It is legal to select from the same table into which insertions are made, but this practice is not advised because it may result in infinite row insertions.

The TRANSACTION clause can be used in multiple transaction SQL applications to specify which transaction controls the INSERT operation. The TRANSACTION clause is not available in DSQL or isql.

Examples

The following statement, from an embedded SQL application, adds a row to a table, assigning values from host-language variables to two columns:

```
EXEC SQL
   INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID)
      VALUES (:emp_no, :proj_id);
```

The next isql statement specifies values to insert into a table with a SELECT statement:

```
INSERT INTO PROJECTS
   SELECT * FROM NEW_PROJECTS
   WHERE NEW_PROJECTS.START_DATE > '6-JUN-1994';
```

See also:

UPDATE

Changes the data in all or part of an existing row in a table, view, or active set of a cursor. Available in gpre, DSQL, and isql.

Syntax SQL form

```
UPDATE [TRANSACTION transaction] {table | view}
    SET col = val [, col = val ...]
    [WHERE search_condition | WHERE CURRENT OF cursor]
    [ORDER BY order_list]
    [ROWS value [TO upper_value] [BY step_value][PERCENT][WITH TIES]];

 DSQL and isql form:

 UPDATE {table | view}
    SET col = val [, col = val ...]
    [WHERE search_condition
    [ORDER BY order_list]
    [ROWS value [TO upper_value] [BY step_value][PERCENT][WITH TIES]]

 <val> = {
    col [array_dim]
    | :variable
    | constant
    | expr
    | function
    | udf ([val [, val ...]])
    | NULL
    | USER
    | ?}
    [COLLATE collation]

 <array_dim> = [[x:]y [, [x:]y ...]]

 <constant> = num | 'string' | charsetname 'string'

 <function> = CAST (val AS datatype)
    | UPPER (val)
    | GEN_ID (generator, val)
```

```
<expr> = A valid SQL expression that results in a single value.
<search_condition> = See CREATE TABLE for a full description.
```

Notes on the UPDATE statement

- In SQL and isql, you cannot use val as a parameter placeholder (like "?").
- In DSQL and isql, val cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.

Argument	Description
TRANSACTION transaction	Name of the transaction under control of which the statement is executed.

table | view	Name of an existing table or view to update.
SET col = val	Specifies the columns to change and the values to assign to those columns.
WHERE search_condition	Searched update only; specifies the conditions a row must meet to be modified.
WHERE CURRENT OF cursor	Positioned update only; specifies that the current row of a cursor's active set is to be modified. Not available in DSQL and isql.
ORDER BY order_list	Specifies columns to order, either by column name or ordinal number in the query, and the sort order (ASC or DESC) for the returned rows.

```
ROWS[1] value
[TO upper_value]
[BY step_value]
[PERCENT][WITH TIES]
```

- Value is the total number of rows to return if used by itself.
- Value is the starting row number to return if used with TO.
- Value is the percent if used with PERCENT.
- Upper_value is the last row or highest percent to return.
- If step_value = n, returns every nth row, or n percent rows.
- PERCENT causes all previous ROWS values to be interpreted as percents.
- WITH TIES returns additional duplicate rows when the last value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with ORDER BY.

[1] Please also refer to ROWS syntax for Firebird 2.0 syntax, description and examples.

New in Firebird 2.0: New extensions to UPDATE and DELETE syntaxes - ROWS specifications and PLAN and ORDER BY clauses can now be used in UPDATE and DELETE statements.

Users can now specify explicit plans for UPDATE/DELETE statements in order to optimize them manually. It is also possible to limit the number of affected rows with a ROWS clause, optionally used in combination with an ORDER BY clause to have a sorted record set.

Syntax

```
UPDATE ... SET ... WHERE ...
[PLAN <plan items>]
[ORDER BY <value list>]
[ROWS <value> [TO <value>]]
```

Description

UPDATE modifies one or more existing rows in a table or view. UPDATE is one of the database privileges controlled by GRANT and REVOKE.

For searched updates, the optional WHERE clause can be used to restrict updates to a subset of rows in the table. Searched updates cannot update array slices.

Important

Without a WHERE clause, a searched update modifies all rows in a table.

When performing a positioned update with a cursor, the WHERE CURRENT OF clause must be specified to update one row at a time in the active set.

Note: When updating a blob column, UPDATE replaces the entire blob with a new value.

Examples

The following isql statement modifies a column for all rows in a table:

```
UPDATE CITIES
   SET POPULATION = POPULATION * 1.03;
```

The next embedded SQL statement uses a WHERE clause to restrict column modification to a subset of rows:

```
EXEC SQL
   UPDATE PROJECT
   SET PROJ_DESC = :blob_id
   WHERE PROJ_ID = :proj_id;
```

DELETE

Removes rows in a table or in the active set of a cursor. Available in gpre, DSQL, and isql.

Syntax SQL and DSQL form

Important: Omit the terminating semicolon for DSQL.

```
DELETE [TRANSACTION transaction] FROM table
   {[WHERE search_condition] | WHERE CURRENT OF cursor}
```

```
    [ORDER BY order_list]
    [ROWS value [TO upper_value] [BY step_value][PERCENT][WITH TIES]];
```

`<search_condition>` = Search condition as specified in SELECT.

isql form

```
DELETE FROM TABLE [WHERE search_condition];
```

Argument	Description
`TRANSACTION transaction`	Name of the transaction under control of which the statement is executed; SQL only.
`table`	Name of the table from which to delete rows.
`WHERE search_condition`	Search condition that specifies the rows to delete; without this clause, `DELETE` affects all rows in the specified table or view.
`WHERE CURRENT OF cursor`	Specifies that the current row in the active set of cursor is to be deleted.
`ORDER BY order_list`	Specifies columns to order, either by column name or ordinal number in the query, and the sort order (`ASC` or `DESC`) for the returned rows.

```
ROWS[1] value
[TO upper_value]
[BY step_value]
[PERCENT][WITH TIES]
```

- `Value` is the total number of rows to return if used by itself.
- `Value` is the starting row number to return if used with `TO`.
- `Value` is the percent if used with `PERCENT`.
- `Upper_value` is the last row or highest percent to return.
- If `step_value = n`, returns every nth row, or `n` percent rows.
- `PERCENT` causes all previous `ROWS` values to be interpreted as percents.
- `WITH TIES` returns additional duplicate rows when the last value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with `ORDER BY`.

[1] Please also refer to `ROWS` syntax for Firebird 2.0 syntax, description and examples.

New in Firebird 2.0: New extensions to `UPDATE` and `DELETE` syntaxes- `ROWS` specifications and `PLAN` and `ORDER BY` clauses can now be used in `UPDATE` and `DELETE`]] statements.

Users can now specify explicit plans for `UPDATE`/`DELETE` statements in order to optimize them manually. It is also possible to limit the number of affected rows with a `ROWS` clause, optionally used in combination with an `ORDER BY` clause to have a sorted recordset.

Syntax

```
DELETE ... FROM ...
[PLAN <plan items>]
[ORDER BY <value list>]
[ROWS <value> [TO <value>]]
```

Description

`DELETE` specifies one or more rows to delete from a table or . `DELETE` is one of the database privileges controlled by the `GRANT` and `REVOKE` statements.

The `TRANSACTION` clause can be used in multiple transaction SQL applications to specify which transaction controls the `DELETE` operation. The `TRANSACTION` clause is not available in DSQL or isql.

For searched deletions, the optional `WHERE` clause can be used to restrict deletions to a subset of rows in the table.

Important

Without a `WHERE` clause, a searched delete removes all rows from a table.

When performing a positioned delete with a cursor, the `WHERE CURRENT OF` clause must be specified to delete one row at a time from the active set.

Examples

The following isql statement deletes all rows in a table:

```
DELETE FROM EMPLOYEE_PROJECT;
```

The next embedded SQL statement is a searched delete in an embedded application. It deletes all rows where a host-language variable equals a column value.

```
EXEC SQL
    DELETE FROM SALARY_HISTORY
    WHERE EMP_NO = :emp_num;
```

The following embedded SQL statements use a cursor and the `WHERE CURRENT OF` option to delete rows from `CITIES` with a population less than the host variable, `min_pop`. They declare and open a cursor that finds qualifying cities, fetch rows into the cursor, and delete the current row pointed to by the cursor.

```
EXEC SQL
   DECLARE SMALL_CITIES CURSOR FOR
   SELECT CITY, STATE
   FROM CITIES
   WHERE POPULATION < :min_pop;

EXEC SQL
   OPEN SMALL_CITIES;

EXEC SQL
   FETCH SMALL_CITIES INTO :cityname, :statecode;
   WHILE (!SQLCODE)
     {EXEC SQL
        DELETE FROM CITIES
        WHERE CURRENT OF SMALL_CITIES;
     EXEC SQL
        FETCH SMALL_CITIES INTO :cityname, :statecode;}
EXEC SQL
  CLOSE SMALL_CITIES;
```

MERGE

`MERGE` is used to combine the data of multiple tables. It is something of a combination of the `INSERT` and `UPDATE` elements.

See also:
DCL- Data Control Language
DDL - Data Definition Language
Data Retrieval
Data Transaction
SQL basics

DDL - Data Definition Language

DDL is the abbreviation for Data Definition Language.

The task of DDL is database definition, i.e. the predefinition and manipulation of the metadata. Using different DDL commands, the database metadata can be created, altered and deleted. For example table structure, use of indices, the activation of exceptions and construction of procedures can all be defined by DDL commands. DDL commands are a subarea of SQL; the range of the SQL language is composed of DDL and DML together.

Important: In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in isql, the semicolon is a terminating symbol for the statement, so it must be included.

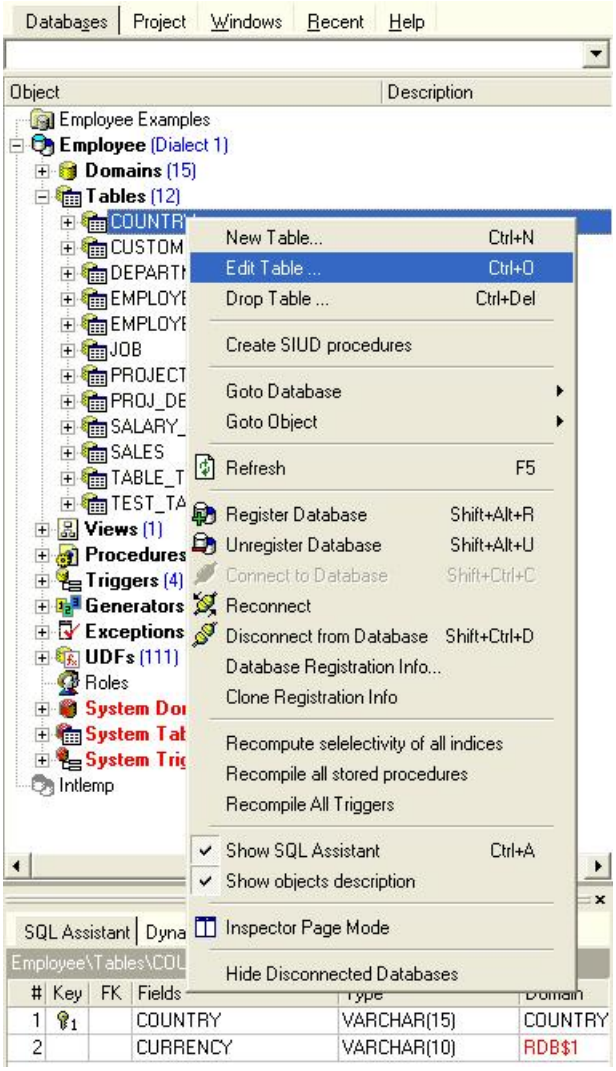
The source of all definitions included in this section is the Borland *InterBase Language Reference*.

ALTER

ALTER is the SQL command used to modify database objects, i.e. databases, domains, tables, views, triggers, procedures, generators/sequences, UDFs etc. can all be changed using the ALTER command.

The different versions of the ALTER command serve to extend or change an already defined structure, the type of alteration defined as an additional attribute of the command. This allows, for example, the metadata in already defined tables, stored procedures or triggers to be manipulated.

A database object can be altered in IBExpert using the DB Explorer right mouse button menu (Edit ...) or simply by double-clicking on the object to be altered.



Alterations can of course also be made directly in the SQL Editor.

CONNECT

A connection can be made to one or more existing databases using the CONNECT command.

The connection parameters can be specified in IBExpert using the menu item Database / Register Database. Here a specified connection may also be tested. the IBExpert menu item Services / Communication Diagnostics may be used to analyze connection problems. It delivers a detailed protocol of the test connect to a registered InterBase/Firebird server and the results. IBExpert also offers toolbar icons for connecting, reconnecting and disconnecting to a registered database.

The CONNECT statement initializes the database data structures and determines if the database is on the originating node (local database) or on another node (remote database). An error message occurs if InterBase/Firebird cannot locate the database. The CONNECT statement attaches to the database and verifies the header page. The database file must contain a valid database, and the on-disk structure (ODS) version number of the database must be recognized by the installed InterBase version on the server.

It is possible to specify a cache buffer for the process attaching to a database. In SQL programs, a database must first be declared with the SET DATABASE command, before it can be opened with the CONNECT statement. When attaching to a database, CONNECT uses the default character set (NONE), or one specified in a previous SET NAMES statement.

A subset of CONNECT features is available in ISQL (see syntax below). ISQL can only be connected to one database at a time. Each time the CONNECT statement is used to connect to a database, previous attachments are disconnected. ISQL does not use SET DATABASE.

Syntax ISQL form

```
CONNECT 'filespec' [USER 'username'][PASSWORD 'password']
   [CACHE int] [ROLE 'rolename']

SQL form:

CONNECT [TO] {ALL | DEFAULT} config_opts
   | db_specs config_opts [, db_specs config_opts...];
<db_specs> = dbhandle
   | {'filespec' | :variable} AS dbhandle
<config_opts> = [USER {'username' | :variable}]
   [PASSWORD {'password' | :variable}]
```

```
[ROLE {'rolename' | :variable}]
[CACHE int [BUFFERS]]
```

Argument	Description
{ALL | DEFAULT}	Connects to all databases specified with SET DATABASE; options specified with CONNECT TO ALL affect all databases.
'filespec'	Database file name - can include path specification and node. The filespec must be in quotes if it includes spaces. |
dbhandle	Database handle declared in a previous SET DATABASE statement; available in embedded SQL but not in isql.
:variable	Host-language variable specifying a database, user name, or password; available in embedded SQL but not in isql.
AS dbhandle	Attaches to a database and assigns a previously declared handle to it; available in embedded SQL but not in isql.
USER {'username' | :variable}	String or host-language variable that optionally specifies a user name for use when attaching to the database. The server checks the user name against the . User names are case insensitive on the server. PC clients must always send a valid user name and password.
PASSWORD {'password' | :variable}	String or host-language variable, up to 8 characters in size, that specifies password for a user listed in the security database, if used, for use when attaching to the database. The server checks the user name and password against the security database. Case sensitivity is retained for the comparison. PC clients must always send a valid user name and password.
ROLE {'rolename' | :variable}	String or host-language variable, up to 67 characters in size, which optionally specifies the role that the user adopts on connection to the database. The user must have previously been granted membership in the role to gain the privileges of that role. Regardless of role memberships granted, the user has the privileges of a role at connect time only if a ROLE clause is specified in the connection. The user can adopt at most one role per connection, and cannot switch roles except by reconnecting.
CACHE int [BUFFERS]	Sets the number of cache buffers for a database (default is 75), which determines the number of database pages a program can use at the same time. Values for int: a) Default: 256, b) Maximum value: system-dependent. This can be used to set a new default size for all databases listed in the CONNECT statement that do not already have a specific cache size, or specify a cache for a program that uses a single database. The size of the cache persists as long as the attachment is active. A decrease in cache size does not affect databases that are already attached through a server. Do not use the filespec form of database name with cache assignments.

Example

```
CONNECT C:\DB01\DB01.GDB USER SYSDBA PASSWORD masterkey
```

In the above example a connection is made to the InterBase database DB01.GDB in the C:\DB01 directory on a Windows NT Server.

When making a connection to a UNIX server the path definitions need to be adapted accordingly:

```
CONNECT /usr/db01/db01.gdb USER SYSDBA PASSWORD masterkey
```

If the user details are not specified when performing the CONNECT command, the relevant system variables for establishing the connection to the specified database are used. This can have the consequence, that if these variables have undefined values, a database connection is not made, and instead an appropriate error message appears.

CREATE

CREATE is the SQL command used to create database objects, i.e. databases, domain, tables, views, triggers, procedures, generators, UDFs etc. can all be defined using the CREATE command.

A database object can be created in IBExpert using the DB Explorer right mouse button menu (New ...), the Database menu, or the respective *NewDatabase Object* icon.

It can of course also be created, by those who are competent in SQL, directly in the SQL Editor. CREATE command syntax can be found under the respective subjects (e.g. Create Database, Create Domain, Create Table, etc.).

DECLARE EXTERNAL FUNCTION (incorporating a new UDF library)

In order to use an already defined or programmed UDF (User-Defined Function) within an InterBase/Firebird database, this has to be explicitly declared using the DECLARE EXTERNAL FUNCTION command.

The DECLARE EXTERNAL FUNCTION command syntax is as follows:

```
DECLARE EXTERNAL FUNCTION name [datatype | CSTRING (int)
[, datatype | CSTRING (int) …]]
   RETURNS {datatype [BY VALUE] | CSTRING (int) | PARAMETER n}  [FREE_IT]
   ENTRY_POINT <External_Function_Name>
   MODULE NAME <Library_Name>;
```

By declaring the UDF, the database is informed of the following for an existing UDF (<External_Function_Name>):

Argument	Description
name	Name of the UDF to use in SQL statements. It can be different to the name of the function specified after the ENTRY_POINT keyword.

datatype	Datatype of an input or return parameter. All input parameters are passed to a UDF by reference. Return parameters can be passed by value. It cannot be an array element.
CSTRING (int)	Specifies a UDF that returns a null-terminated string int bytes in length.
RETURNS	Specifies the return value of a function.
BY VALUE	Specifies that a return value should be passed by value rather than by reference.
PARAMETER n	Specifies that the nth input parameter is to be returned. Used when the return datatype is a blob.
FREE_IT	Frees memory of the return value after the UDF finishes running.
<External_Function_Name>	Quoted string that contains the function name as it is stored in the library that is referenced by the UDF. The entryname is the actual name of the function as stored in the UDF library. It does not have to match the name of the UDF as stored in the database.
<Library_Name>	Quoted specification identifying the library that contains the UDF. The library must reside on the same machine as the InterBase/Firebird server. On any platform, the module can be referenced with no path name if it is in. <InterBase/Firebird_home>/UDF or <InterBase/Firebird_home>/intl. If the library is in a directory other than <InterBase/Firebird_home>/UDF or InterBase/Firebird_home>/intl, you must specify its location in InterBase/Firebird's configuration file (ibconfig) using the EXTERNAL_FUNCTION_DIRECTORY parameter. It is not necessary to supply the extension to the module name.

The UDF name in the database does not have to correspond to the original function name. The input parameters are basically transferred BY REFERENCE. In the case of the return parameters it is also possible to specify the form BY VALUE, using the optional BY VALUE parameter.

Note: Whenever a UDF returns a value by reference to dynamically allocated memory, you must declare it using the FREE_IT keyword in order to free the allocated memory.

To specify a location for UDF libraries in a configuration file, enter the following for Windows platforms:

```
EXTERNAL_FUNCTION_DIRECTORY D:\Mylibraries\InterBase
```

For UNIX, the statement does not include a drive letter:

```
EXTERNAL_FUNCTION_DIRECTORY \Mylibraries\InterBase
```

The InterBase/Firebird configuration file is called ibconfig or firebird.conf on all platforms.

Examples

The following isql statement declares the TOPS() UDF to a database:

```
DECLARE EXTERNAL FUNCTION TOPS
   CHAR(256), INTEGER, BLOB
   RETURNS INTEGER BY VALUE
   ENTRY_POINT 'te1' MODULE_NAME 'tm1';
```

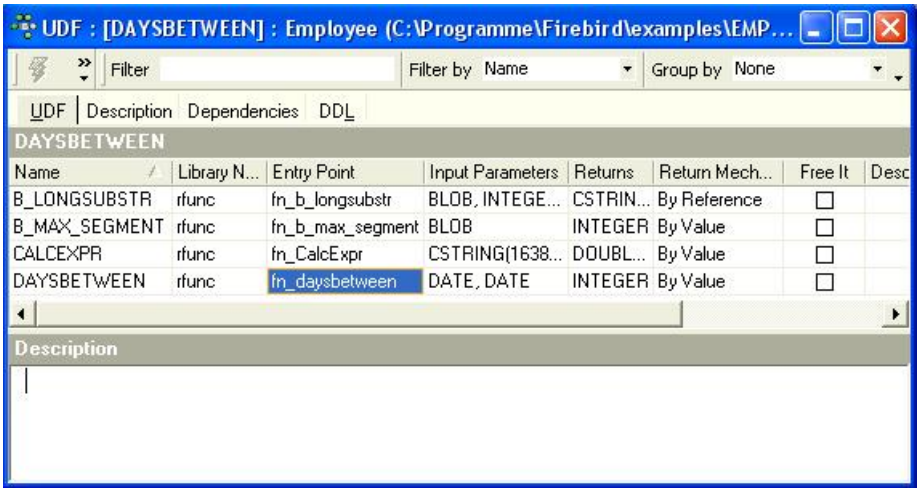
This example does not need the FREE_IT keyword because only cstrings, CHAR and VARCHAR return types require memory allocation.

The next example declares the LOWERS() UDF and frees the memory allocated for the return value:

```
DECLARE EXTERNAL FUNCTION LOWERS VARCHAR(256)
   RETURNS CSTRING(256) FREE_IT
   ENTRY POINT 'fn_lower' MODULE_NAME 'udflib';
```

In the example below (taken from the RFunc library) a function SUBSTR is declared, which calculates the substring of strings, from character i1 and length maximum i2:

```
DECLARE EXTERNAL FUNCTION SUBSTR
    CSTRING(256),
    INTEGER,
    INTEGER
RETURNS CSTRING(256)
ENTRY_POINT 'fn_substr' MODULE_NAME 'rfunc';
```



ENTRY_POINT

`ENTRY_POINT` is a term used in the declaration of an external function.

Syntax

```
ENTRY_POINT <External_Function_Name>
```

The entry point is a text which specifies when the function should jump into a starting address from a DLL.

MODULE NAME

The DLL name of a UDF is entered as the last parameter when declaring an external function.

Syntax

```
MODULE NAME <Library_Name>
```

It specifies in which UDF library the UDF can be found (`<Library_Name>`). Whether the file suffix needs to be entered or not, and how, is dependent upon the operating system. For example, Linux requires the suffix `.so` (Shared Object Library); in Windows `.DLL` (Dynamic Link Library).

RETURNS

`RETURNS` is a term used in the declaration of an external function. Here the output parameters are specified (i.e. datatype and in which form).

Syntax

```
RETURNS <Return_Type>
```

`RETURN` parameters can also be specified in the form `BY VALUE`, using the optional `BY VALUE` parameter.

DISCONNECT

The `DISCONNECT` command detaches an application from one or more databases, defined by its/their database handle, and frees the relevant sources. Available in gpre.

In IBExpert there is a toolbar icon to execute this command (or alternatively use the IBExpert menu item Database / Disconnect from Database).

Syntax

```
DISCONNECT {{ALL | DEFAULT} | dbhandle [, dbhandle] …]};
```

- **`ALL|DEFAULT`:** Either keyword detaches all open databases.
- **`dbhandle`:** Previously declared database handle specifying a database to detach.

`DISCONNECT` closes a specific database identified by a database handle or all databases, releases resources used by the attached database, zeroes database handles, commits the default transaction if the gpre `-manual` option is not in effect, and returns an error if any non-default transaction is not committed.

Before using `DISCONNECT`, commit or roll back the transactions affecting the database to be detached.

Examples

The following embedded SQL statements close all databases:

```
EXEC SQL
DISCONNECT DEFAULT;
```

```
EXEC SQL
DISCONNECT ALL;
```

The following embedded SQL statements close the databases identified by their handles:

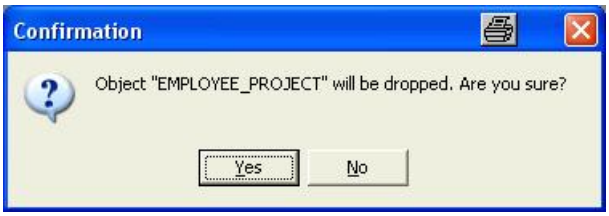
```
EXEC SQL
DISCONNECT DB1;

EXEC SQL
DISCONNECT DB1, DB2;
```

DROP

DROP is the SQL command used to delete database objects, i.e. databases, domains, tables, views, triggers, procedures, generators, UDFs etc. can all be deleted using the DROP command.

A database object can be dropped in IBExpert using the DB Explorer right mouse button menu (*Drop ...*).

IBExpert requires confirmation of this command, as it is irreversible.



The DROP command can of course also be used directly in the SQL Editor. More information can be found under the respective subjects (e.g. Drop Database, Drop Domain, Drop Table, etc.).

Syntax

```
DROP <database_object_type> <object_name>;
```

Example

```
DROP TABLE Customer;
```

END DECLARE SECTION

Identifies the end of a host-language variable declaration section. Available in gpre.

Syntax

```
END DECLARE SECTION;
```

The END DECLARE SECTION command is used in embedded SQL applications to identify the end of host-language variable declarations for variables used in subsequent SQL statements.

Example:

The following embedded SQL statements declare a section and single host-language variable:

```
EXEC SQL
   BEGIN DECLARE SECTION;
      BASED_ON EMPLOYEE.SALARY salary;

EXEC SQL
   END DECLARE SECTION;
```

EVENT

EVENT INIT

EVENT INIT is the first step in the InterBase two-part synchronous event mechanism:

1. EVENT INIT registers an application's interest in an event.
2. EVENT WAIT causes the application to wait until notified of the event's occurrence.

EVENT INIT registers an application's interest in a list of events in parentheses. The list should correspond to events posted by stored procedures or triggers in the database. If an application registers interest in multiple events with a single EVENT INIT, then when one of those events occurs, the application must determine which event occurred. The command EVENT INIT is only required by embedded SQL programmers, and not required when programming the BDE.

Events are posted by a POST_EVENT call within a stored procedure or trigger. The event manager keeps track of events of interest. At commit time, when an event occurs, the event manager notifies interested applications.

The EVENT INIT command is constructed as follows:

Syntax

```
EVENT INIT request_name [dbhandle]
   [('string' | :variable [, 'string' | :variable …]);
```

Argument	Description
request_name	Application event handle.
dbhandle	Specifies the database to examine for occurrences of the events; if omitted, dbhandle defaults to the database named in the most recent SET DATABASE statement.
'string'	Unique name identifying an event associated with event_name.
:variable	Host language character array containing a list of event names to associate with.

Example:

The following embedded SQL statement registers interest in an event:

```
EXEC SQL
  EVENT INIT ORDER_WAIT EMPDB ('new_order');
```

See also:
Create Procedure
Create Trigger
SET DATABASE

EVENT WAIT

Causes an application to wait until notified of an event's occurrence. Available in gpre.

Syntax

```
EVENT WAIT request_name;
```

Argument	Description
request_name	Application event handle declared in a previous EVENT INIT statement.

EVENT WAIT is the second step in the InterBase/Firebird two-part synchronous event mechanism. After a program registers interest in an event, EVENT WAIT causes the process running the application to sleep until the event of interest occurs.

Examples

The following embedded SQL statements register an application event name and indicate the program is ready to receive notification when the event occurs:

```
EXEC SQL
   EVENT INIT ORDER_WAIT EMPDB ('new_order');

EXEC SQL
   EVENT WAIT ORDER_WAIT;
```

EXECUTE

The EXECUTE command performs a specified SQL statement. The statement can be any SQL data definition, manipulation, or transaction management statement. Once it is prepared, a statement can be executed any number of times.

SQL commands can be executed using the [F9] key or following icon:

enabling the SQL code to be executed and tested before finally committing.

Should a part of the text have been highlighted, only the marked portion is executed, which often causes an error message. If the execution has been successful, the SQL can be committed using the respective icon or [Ctrl + Alt + C].

Syntax

```
EXECUTE [TRANSACTION transaction] statement
   [USING SQL DESCRIPTOR xsqlda] [INTO SQL DESCRIPTOR xsqlda];
```

Argument	Description
request_name	Application event handle declared in a previous EVENT INIT statement.
TRANSACTION transaction	Specifies the transaction under which execution occurs: This clause can be used in SQL applications running multiple, simultaneous transactions to specify which transaction controls the EXECUTE operation.
USING SQL DESCRIPTOR	Specifies those values corresponding to the prepared statement's parameters should be taken from the specified XSQLDA. It need only be used for statements that have dynamic parameters.
INTO SQL DESCRIPTOR	Specifies that return values from the executed statement should be stored in the specified XSQLDA. It need only be used for DSQL statements that return values.
xsqlda	XSQLDA host-language variable.

Note: If an EXECUTE statement provides both a USING DESCRIPTOR clause and an INTO DESCRIPTOR clause, then two XSQLDA structures must be provided.

EXECUTE carries out a previously prepared DSQL statement. It is one of a group of statements that process DSQL statements.

- **PREPARE:** Readies a DSQL statement for execution.
- **DESCRIBE:** Fills in the XSQLDA with information about the statement.
- **EXECUTE:** Executes a previously prepared statement.
- **EXECUTE IMMEDIATE:** Prepares a DSQL statement, executes it once, and discards it (please refer to the EXECUTE IMMEDIATE statement for further information).

Before a statement can be executed, it must be prepared using the PREPARE statement. The statement can be any SQL data definition, manipulation, or transaction management statement. Once it is prepared, a statement can be executed any number of times.

Example

The following embedded SQL statement executes a previously prepared DSQL statement:

```
EXEC SQL
   EXECUTE DOUBLE_SMALL_BUDGET;
```

The next embedded SQL statement executes a previously prepared statement with parameters stored in an XSQLDA:

```
EXEC SQL
   EXECUTE Q USING DESCRIPTOR xsqlda;
```

The following embedded SQL statement executes a previously prepared statement with parameters in one XSQLDA, and produces results stored in a second XSQLDA:

```
EXEC SQL
   EXECUTE Q USING DESCRIPTOR xsqlda_1 INTO DESCRIPTOR xsqlda_2;
```

EXECUTE PROCEDURE

Calls a specified stored procedure. Available in gpre, DSQL, and isql.

In IBExpert a procedure can be executed in the Stored Procedure Editor or SQL Editor using the [F9] key or following icon:

Syntax SQL form

```
EXECUTE PROCEDURE [TRANSACTION transaction]
   name [:param [[INDICATOR]:indicator]]
      [, :param [[INDICATOR]:indicator] …]
   [RETURNING_VALUES :param [[INDICATOR]:indicator]
      [, :param [[INDICATOR]:indicator] …]];
```

DSQL form

```
EXECUTE PROCEDURE name [param [, param …]]
   [RETURNING_VALUES param [, param …]]
```

isql form

```
EXECUTE PROCEDURE name [param [, param …]]
```

Argument	Description
TRANSACTION transaction	Specifies the TRANSACTION under which execution occurs.
name	Name of an existing stored procedure in the database.
param	Input or output parameter; can be a host variable or a constant.
RETURNING_VALUES: param	Host variable which takes the values of an output parameter.
[INDICATOR] :indicator	Host variable for indicating NULL or unknown values.

EXECUTE PROCEDURE calls the specified stored procedure. If the procedure requires input parameters, they are passed as host-language variables or as constants. If a procedure returns output parameters to a SQL program, host variables must be supplied in the RETURNING_VALUES clause to hold the values returned.

In isql, do not use the RETURN clause or specify output parameters. isql will automatically display return values.

Note: in DSQL, an EXECUTE PROCEDURE statement requires an input descriptor area if it has input parameters and an output descriptor area if it has output parameters.

In embedded SQL, input parameters and return values may have associated indicator variables for tracking NULL values. Indicator variables are integer values that indicate unknown or NULL values of return values.

An indicator variable that is less than zero indicates that the parameter is unknown or NULL. An indicator variable that is zero or greater indicates that the associated parameter is known and not NULL.

Examples

The following embedded SQL statement demonstrates how the executable procedure, DEPT_BUDGET, is called from embedded SQL with literal parameters:

```
EXEC SQL
   EXECUTE PROCEDURE DEPT_BUDGET 100
     RETURNING_VALUES :sumb;
```

The next embedded SQL statement calls the same procedure using a host variable instead of a literal as the input parameter:

```
EXEC SQL
   EXECUTE PROCEDURE DEPT_BUDGET :rdno
     RETURNING_VALUES :sumb;
```

SET

SET DATABASE

The SET DATABASE command creates a so-called database handle when creating embedded SQL applications for a specified database. It is available in gpre.

As it is possible to access several databases with embedded SQL applications, the desired database can be explicitly specified with the aid of the handle. The SET DATABASE command is only required by embedded SQL programmers and is not necessary for programming the BDE.

Syntax

```
SET DATABASE DB_Handle =
 [GLOBAL | STATIC | EXTERN]
 [COMPILETIME] [FILENAME] "<db_Name>"
 [USER "UserName" PASSWORD "PassString"]
 [RUNTIME] [FILENAME] {"<DB_Name>"|:VarDB}
 [USER {"Name"| :VarName}
  PASSWORD {"Password"| :VarPassWord=};
```

DB_Handle: This is the name of the database handle, defined by the application. It is an alias (usually an abbreviation) for a specified database. It must be unique within the program, follow the file syntax conventions for the server where the database resides, and be used in subsequent SQL statements that support database handles. For example, they can be used in subsequent CONNECT, COMMIT and ROLLBACK statements, or can also be used within transactions to differentiate table names when two or more attached databases contain tables with the same names. The optional parameters GLOBAL, STATIC and EXTERN can be used to specify the validity range of the database declaration. Following rules apply for the validity range:

Global	The database declaration is visible for all modules (default).
Static	Limits the database declaration to the current module (i.e. limit the database handle availability to the code module where the handle is declared).
Extern	References a global database handle in another module, rather than actually declaring a new handle.
Compiletime	Identifies the database used to look up column references during preprocessing. If only one database is specified in SET DATABASE, it is used both at runtime and compiletime.
Runtime	Specifies a database to use at runtime if different thatn the one specified for use during preprocessing. And if necessary, different standard users can be specified for both situations. InterBase/Firebird sets the same database for runtime and development time as standard, if the optional parameters COMPILETIME and RUNTIME are not used.
<DB_Name>	Represents a file specification for the database to associate with db_handle. It is platform-specific.
:VarDB	This is the host-language variable containing a database specification, user name, or password.

USER and PASSWORD	Valid user name and password on the server where the database resided. Required for PC client attachments, optional for all others.

Example

```
EXEC SQL
  SET DATABASE EMPDB = 'employee.gdb'
  COMPILETIME "Test.gdb"
  RUNTIME :db_runtime;
```

SET GENERATOR

The SET GENERATOR command sets a new start value for an existing generator.

The SET GENERATOR command syntax is composed as follows:

```
SET GENERATOR Gen_Name TO int_value;
```

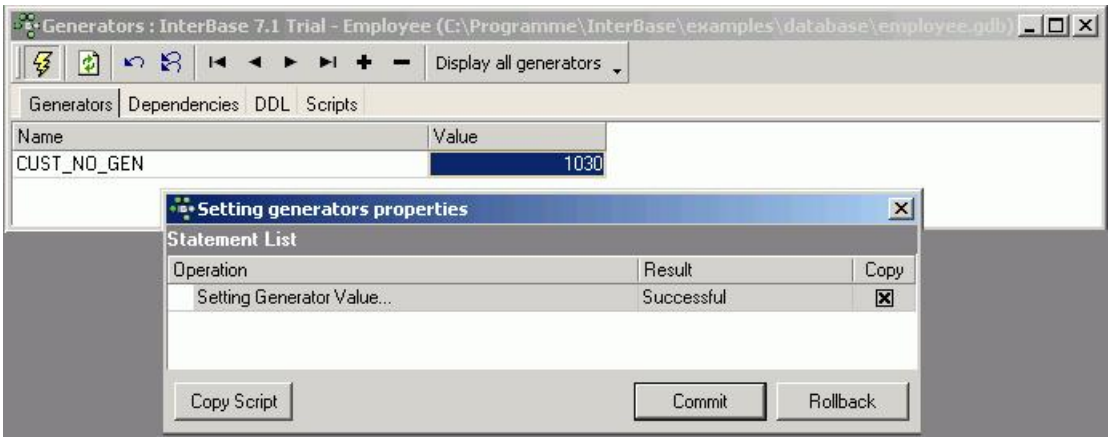
As soon as the function GEN_ID() enters or alters a value in a table column, this value is calculated from the int_value plus the increment defined by the GEN_ID() step parameter.

Example

```
SET GENERATOR CUST_ID_GEN TO 1030;
```

Assuming that the step parameter in the function GEN_ID() is given the value 1, the next customer would receive the customer number 1031.

This statement can also be easily and quickly performed using IBExpert's Generator Editor (please refer to Alter Generator for further information):



SET NAMES

The SET NAMES statement specifies an active character set to use for subsequent database attachments. Available in gpre, and isql.

Syntax

```
SET NAMES [charset | :var];
```

charset	Name of a character set that identifies the active character set for a given process; default: NONE.
:var	Host variable containing string identifying a known character set name. Must be declared as a character set name. SQL only.

SET NAMES specifies the character set to use for subsequent database attachments in an application. It enables the server to translate between the default character set for a database on the server and the character set used by an application on the client.

SET NAMES must appear before the SET DATABASE and CONNECT statements it is to affect.

Tip: Use a host-language variable with SET NAMES in an embedded application to specify a character set interactively.

Choice of character sets limits possible collation orders to a subset of all available collation orders. Given a specific character set, a specific collation order can be specified when data is selected, inserted, or updated in a column. If a default character set is not specified, the character set defaults to NONE.

Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as it is originally entered. You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a different character set. No transliteration is performed between the source and destination character sets, so in most cases, errors occur during assignment.

Example

The following statements demonstrate the use of SET NAMES in an embedded SQL application:

```
EXEC SQL
   SET NAMES ISO8859_1;

EXEC SQL
```

```
    SET DATABASE DB1 = 'employee.gdb';

EXEC SQL
    CONNECT;
```

The next statements demonstrate the use of `SET NAMES` in isql:

```
SET NAMES LATIN1;
    CONNECT 'employee.gdb';
```

SET SQL DIALECT

`SET SQL DIALECT` declares the SQL dialect for database access.

`n` is the SQL dialect type, either `1`, `2`, or `3`. If no dialect is specified, the default dialect is set to that of the specified compile-time database. If the default dialect is different than the one specified by the user, a warning is generated and the default dialect is set to the user-specified value. Available in gpre and isql.

Syntax

```
SET SQL DIALECT n;
```

where `n` is the SQL dialect type, either `1`, `2`, or `3`.

SQL Dialect	Used for
1	InterBase 5 and earlier compatibility.
2	Transitional dialect used to flag changes when migrating from dialect `1` to dialect `3`.
3	Current InterBase/Firebird; allows you to use delimited identifiers, exact `NUMERICS`, and `DATE`, `TIME`, and `TIMESTAMP` datatypes.

SET STATISTICS

`SET STATISTICS` enables the selectivity of an index to be recomputed. Index selectivity is a calculation, based on the number of distinct rows in a table, which is made by the InterBase/Firebird optimizer when a table is accessed. It is cached in memory, where the optimizer can access it to calculate the optimal retrieval plan for a given query. For tables where the number of duplicate values in indexed columns radically increases or decreases, periodically recomputing index selectivity can improve performance. Available in gpre, DSQL, and isql.

Only the creator of an index can use `SET STATISTICS`.

Note: `SET STATISTICS` does not rebuild an index. To rebuild an index, use `ALTER INDEX`.

Syntax:

```
SET STATISTICS INDEX name;
```

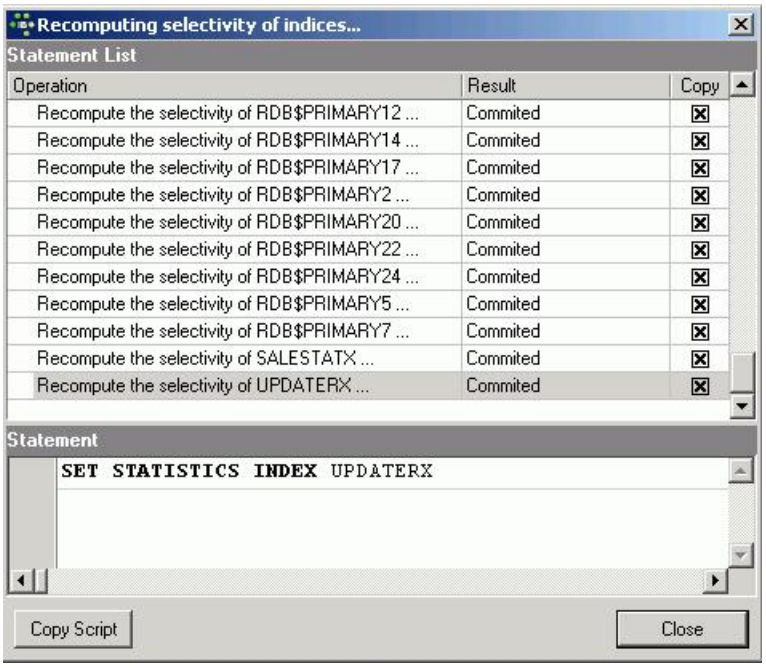
name	Name of an existing index for which to recompute selectivity.

Example:

The following embedded SQL statement recomputes the selectivity for an index:

```
EXEC SQL
    SET STATISTICS INDEX MINSALX;
```

It is possible to recompute the selectivity for all indices using the IBExpert Database menu item Recompute selectivity of all indices.



SET TRANSACTION

`SET TRANSACTION` starts a transaction, and optionally specifies its database access, lock conflict behavior, and level of interaction with other concurrent transactions accessing the same data. It can also reserve locks for tables. As an alternative to reserving tables, multiple database SQL applications can restrict a transaction's access to a subset of connected databases. Available in gpre, DSQL, and isql.

Important: applications preprocessed with the gpre `-manual` switch must explicitly start each transaction with a `SET TRANSACTION` statement.

Syntax

```
SET TRANSACTION [NAME transaction]
   [READ WRITE | READ ONLY]
   [WAIT | NO WAIT]
   [[ISOLATION LEVEL] {SNAPSHOT [TABLE STABILITY]
      | READ COMMITTED [[NO] RECORD_VERSION]}]
   [RESERVING reserving_clause
      | USING dbhandle [, dbhandle …]];
<reserving_clause> = table [, table …]
   [FOR [SHARED | PROTECTED] {READ | WRITE}] [, reserving_clause]
```

`NAME transaction`	Specifies the name for this transaction. Transaction is a previously declared and initialized host-language variable. SQL only.
`READ WRITE [Default]`	Specifies that the transaction can read and write to tables.
`READ ONLY`	Specifies that the transaction can only read tables.
`WAIT [Default]`	Specifies that a transaction wait for access if it encounters a lock conflict with another transaction.
`NO WAIT`	Specifies that a transaction immediately return an error if it encounters a lock conflict.
`ISOLATION LEVEL`	Specifies the isolation level for this transaction when attempting to access the same tables as other simultaneous transactions; default: `SNAPSHOT`.
`RESERVING reserving_clause`	Reserves lock for tables at transaction start.
`USING dbhandle [, dbhandle …]`	Limits database access to a subset of available databases; SQL only.

Examples

The following embedded SQL statement sets up the default transaction with an isolation level of `READ COMMITTED`. If the transaction encounters an update conflict, it waits to get control until the first (locking) transaction is committed or rolled back.

```
EXEC SQL
   SET TRANSACTION WAIT ISOLATION LEVEL READ COMMITTED;
```

The next embedded SQL statement starts a named transaction:

```
EXEC SQL
   SET TRANSACTION NAME T1 READ COMMITTED;
```

The following embedded SQL statement reserves three tables:

```
EXEC SQL
   SET TRANSACTION NAME TR1
   ISOLATION LEVEL READ COMMITTED
   NO RECORD_VERSION WAIT
```

```
RESERVING TABLE1, TABLE2 FOR SHARED WRITE,
   TABLE3 FOR PROTECTED WRITE;
```

See also:
SET NAMES
COMMIT
ROLLBACK

WHENEVER

WHENEVER traps for SQLCODE errors and warnings. Every executable SQL statement returns a SQLCODE value to indicate its success or failure. If SQLCODE is zero, statement execution is successful. A non-zero value indicates an error, warning, or not found condition. Available in gpre.

If the appropriate condition is trapped, WHENEVER can:

- Use GOTO label to jump to an error-handling routine in an application.
- Use CONTINUE to ignore the condition.

WHENEVER can help limit the size of an application, because the application can use a single suite of routines for handling all errors and warnings.

WHENEVER statements should precede any SQL statement that can result in an error. Each condition to trap for requires a separate WHENEVER statement. If WHENEVER is omitted for a particular condition, it is not trapped.

Tip: Precede error-handling routines with WHENEVER … CONTINUE statements to prevent the possibility of infinite looping in the error-handling routines.

Syntax

```
WHENEVER {NOT FOUND | SQLERROR | SQLWARNING}
   {GOTO label | CONTINUE};
```

NOT FOUND	Traps SQLCODE = 100, no qualifying rows found for the executed statement.
SQLERROR	Traps SQLCODE < 0, failed statement.
SQLWARNING	Traps SQLCODE > 0 AND < 100, system warning or informational message.
GOTO label	Jumps to program location specified by label when a warning or error occurs.
CONTINUE	Ignores the warning or error and attempts to continue processing.

Example

In the following code from an embedded SQL application, three WHENEVER statements determine which label to branch to for error and warning handling:

```
EXEC SQL
   WHENEVER SQLERROR GO TO Error; /* Trap all errors. */

EXEC SQL
   WHENEVER NOT FOUND GO TO AllDone; /* Trap SQLCODE = 100 */

EXEC SQL
   WHENEVER SQLWARNING CONTINUE; /* Ignore all warnings.
```

See also:
Firebird 2.0.4 Release Notes: Data Definition Language
SQL basics

Data Transaction

COMMIT and ROLLBACK interact with areas such as transaction control and locking. Strictly, both terminate any open transaction and release any locks held on data. In the absence of a BEGIN or similar statement, the semantics of SQL are implementation-dependent.

COMMIT

The COMMIT command makes a transaction's changes to the database permanent. It is used to start all transactions.

COMMIT is used to end a transaction and:

- Write all updates to the database.
- Make the transaction's changes visible to subsequent SNAPSHOT transactions or READ COMMITTED transactions.
- Close open cursors, unless the RETAIN argument is used.

After executing a transaction with [F9] or the

icon, and all operations in the transaction have been successfully performed by the server, the changes to the database must be explicitly committed. This can be done using [Ctrl + Alt + C] or the

icon.

Of course, those competent in SQL can also enter the command directly in SQL Editor.

Syntax

```
COMMIT [WORK] [TRANSACTION name] [RELEASE] [RETAIN [SNAPSHOT]];
```

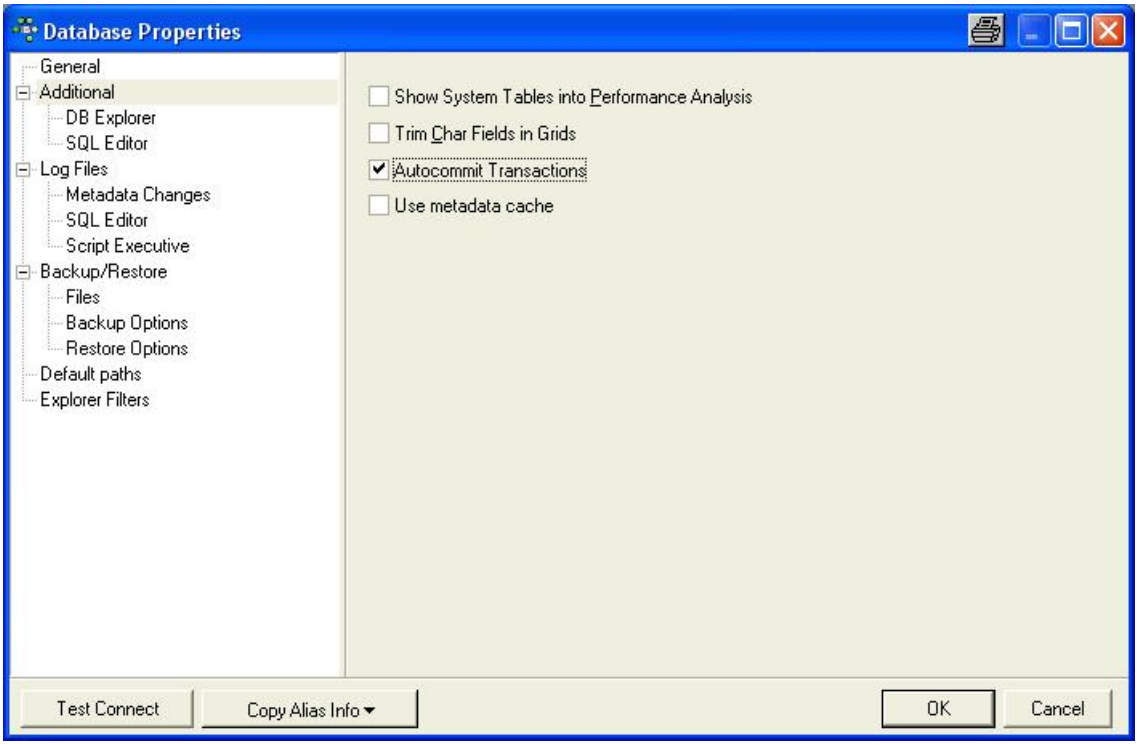
WORK	An optional work used for compatibility with other relational databases that require it.
TRANSACTION name	Commits a transaction name to database. Without this option, COMMIT affects the default transaction.
RELEASE	Available for compatibility with earlier versions of InterBase/Firebird.
RETAIN [SNAPSHOT]	Commits changes and retains current transaction context.

The transaction name is only valid in an embedded SQL application using SQL or DSQL, where more than one transaction can be active at a time.

A transaction ending with COMMIT is considered a successful termination. Always use COMMIT or ROLLBACK to end the default transaction. Tip: after read-only transactions, which make no database changes, use COMMIT rather than ROLLBACK. The effect is the same, but the performance of subsequent transactions is better and the system resources used by them are reduced.

This statement is not valid inside a trigger, because a trigger is started automatically as part of a larger transaction, with other triggers perhaps firing after it. It is also not valid inside a stored procedure because the procedure might be invoked from a trigger.

In IBExpert it is possible to force all commands to be automatically committed, by checking the *Autocommit Transactions* box in the Database Properties dialog / Additional (menu item: Database / Database Registration Info...):



However, this is *NOT* recommended, as it is all too easy to accidentally drop a database (instead of a database field for example), as the developer is no longer asked for confirmation before committing.

ROLLBACK

If a transaction's operations did not all complete successfully or satisfactorily, it is possible to roll back the transaction. A rollback restores the data to the state it was in before the transaction started. All changes made by insertions, updates and deletions are reversed.

The `ROLLBACK` is performed in IBExpert using the

icon or [Ctrl + Alt + R].

Rolling back can of course also be specified by issuing the following statement:

```
ROLLBACK [TRANSACTION name];
```

The transaction name is only required in embedded SQL applications using SQL or DSQL, where more than one transaction can be active at any one time.

It is important to note that when a transaction is rolled back, the changes performed by that transaction are not immediately deleted. Instead, InterBase flags the transaction associated with that entry as having been rolled back in the Transaction Inventory Page (TIP). Subsequent queries must then reconstruct the row using the version history.

When InterBase/Firebird performs a garbage collection or database sweep, the server detects that the row entry for the current version does not in fact contain the complete current version. It is then updated and the various data segments and version history relinked to ensure that the current version of the row is stored in the correct place, so that back versions do not need to be read each time.

See also:
DCL - Data Control Language
DDL - Data Definition Language
DML - Data Manipulation Language
Data Retrieval
Compile, Commit, Rollback

DCL - Data Control Language

The third group of SQL keywords is the Data Control Language (DCL). DCL handles the authorisation aspects of data and permits the user to control who has access to see or manipulate data within the database.

Its two main keywords are:

- **GRANT:** - authorises a user to perform an operation or a set of operations e.g. grant all privileges to `user X`.
- **REVOKE:** - removes or restricts the capability of a user to perform an operation or a set of operations.

GRANT

`GRANT` is the SQL statement, used to assign privileges to database users for specified database objects.

Grants can be assigned and revoked using the IBExpert Grant Manager, the relevant object editors' Grants pages, or the SQL Editor.

InterBase/Firebird offers the following access privileges at database object level:

Privilege	Allows user to:
`SELECT`	Read data.
`INSERT`	Write new data.
`UPDATE`	Modify existing data.
`DELETE`	Delete data.
`ALL`	Select, insert, update, delete data, and reference a primary key from a foreign key. (*Note:* does not include references or code for InterBase 4.0 or earlier).
`EXECUTE`	Execute or call a stored procedure.
`REFERENCES`	Reference a primary key with a foreign key.
`role`	Use all privileges assigned to the role (please refer to Role for further information).

`PUBLIC` is used to assign a set of privileges to every user of the database. Using the `PUBLIC` keyword does not grant the specified rights to stored procedures, only to all database users. Procedures need to be specified explicitly. Please note: `PUBLIC` is really public! This `GRANT` option enables all users to access and manipulate a database object with `PUBLIC` rights, even certain system files.

Table Interactions

Many operations require that the user has rights to linked tables, in order for InterBase/Firebird to process updates.

1. If foreign key constraints exist between two tables, then an `UPDATE`, `DELETE` or `INSERT` operation on the first table requires `SELECT` or `REFERENCES` privileges on the referenced table. *Tip:* Make it easy: if read security is not an issue, `GRANT REFERENCES` on the primary key table to `PUBLIC`. If you grant the `REFERENCES` privilege, it must, at a minimum, be granted to all columns of the primary key. When `REFERENCES` is granted to the entire table, columns that are not part of the primary key are not affected in any way. When a user defines a foreign key constraint on a table owned by someone else, InterBase/Firebird checks that the user has `REFERENCES` privileges on the referenced table. The privilege is used at runtime to verify that a value entered in a foreign key field is contained in the primary key table. You can grant `REFERENCES` privileges to roles.
2. If there is a check constraint within a table, an `UPDATE` or `INSERT` operation also requires `SELECT` privileges on the same table.
3. If a constraint includes one or more queries, an `UPDATE` or `INSERT` operation also requires `SELECT` privileges on the table or tables used in the `SELECT`.

IBExpert allows privileges to be granted on objects at the time of creation directly in the objects editor's *Grants* page (please refer to Table Editor / Grants for further details). Dependencies upon or from other objects are also displayed in the individual object editors, to show visually any object interactions, which may need to be taken into consideration when assigning user permissions. Refer to Table Editor / Dependencies for further information. All objects or a filtered selection of objects can be displayed and processed in the IBExpert Grant Manager.

Privileges can be granted to a role as well as to users or stored procedures, tables, views and triggers.

The `GRANT` statement can be used in gpre, DSQL and isql.

Syntax

```
GRANT privileges ON [TABLE] {tablename | viewname}
       TO {object|userlist [WITH GRANT OPTION]|GROUP UNIX_group}
    | EXECUTE ON PROCEDURE procname TO {object | userlist}
    | role_granted TO {PUBLIC | role_grantee_list}[WITH ADMIN OPTION];

<privileges> = ALL [PRIVILEGES] | privilege_list

<privilege_list> = {
      SELECT
    | DELETE
    | INSERT
    | UPDATE [(col [, col...])]
    | REFERENCES [(col [, col...])]
}[, privilege_list...]
```

```
<object> = {
     PROCEDURE procname
   | TRIGGER trigname
   | VIEW viewname
   | PUBLIC
}[, object...]

<userlist> = {
     [USER] username
   | rolename
   | UNIX_user
}[,userlist...]

<role_granted> = rolename [, rolename...]

<role_grantee_list> = [USER] username [, [USER] username...]
```

privilege_list	Name of privilege to be granted; valid options are SELECT, DELETE, INSERT, UPDATE, and REFERENCES.
col	Column to which the granted privileges apply.
tablename	Name of an existing table for which granted privileges apply.
viewname	Name of an existing view for which granted privileges apply.
GROUP unix_group	On a UNIX system, the name of a group defined in /etc/group.
object	Name of an existing procedure, trigger, or view; PUBLIC is also a permitted value.
userlist	A user in the InterBase/Firebird security database or a role name created with CREATE ROLE.
WITH GRANT OPTION	Passes GRANT authority for privileges listed in the GRANT statement to userlist (please refer to GRANT AUTHORITY for further information).
rolename	An existing role created with the CREATE ROLE statement.
role_grantee_list	A list of users to whom rolename is granted; users must be in the Firebird/InterBase .
WITH ADMIN OPTION	Passes grant authority for roles listed to role_grantee_list.

Important: In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in isql, the semicolon is a terminating symbol for the statement, so it must be included.

To grant privileges to a group of users, create a role using the CREATE ROLE statement. Please refer to New Role for details.

On UNIX systems, privileges can be granted to groups listed in /etc/groups and to any UNIX user listed in /etc/passwd on both the client and server, as well as to individual users and to roles.

Examples

```
GRANT insert, update, delete
   ON customer
   TO Janet, John
   WITH GRANT OPTION;
```

or:

```
GRANT references
   ON customer
   TO PUBLIC;
```

If different levels of access are to be assigned to different objects and different people, separate GRANT statements have to be used.

This embedded SQL statement grants EXECUTE privileges for a procedure to another procedure and to a user:

```
EXEC SQL
   GRANT EXECUTE ON PROCEDURE GET_EMP_PROJ
   TO PROCEDURE ADD_EMP_PROJ, LUIS;
```

The following example creates a role called administrator, grants UPDATE privileges on table1 to that role, and then grants the role to user1, user2, and user3. These users then have UPDATE and REFERENCES privileges on table1:

```
CREATE ROLE administrator;
GRANT UPDATE ON table1 TO administrator;
GRANT administrator TO user1, user2, user3;
```

REVOKE

REVOKE is the SQL statement, used to withdraw those rights already assigned to database users or objects for database objects. Rights can be revoked using the IBExpert Grant Manager, the relevant object editors' Grants pages, or the SQL Editor.

The following rules apply when revoking user privileges:

1. Only the user who granted the privilege or the SYSDBA may revoke it.

2. Revoking a privilege has no effect on any other privileges granted by other users. However, if multiple users have the ability to grant privileges, one user might have received a specific privilege from more than one source. If only one of them is revoked, the other remains in effect.
3. If a privilege, which was originally granted using the `WITH GRANT OPTION` clause, is revoked, any subsequent users to which the same privilege had been granted in turn lose their privileges too.
4. The `ALL` keyword can be used to revoke all granted privileges to an object, even if the user has not been granted all available privileges in the first place. `REVOKE ALL` however has no effect on the `EXECUTE` privilege, which must always be explicitly revoked.
5. If a privilege is granted to all users using the `PUBLIC` option, this grant can only be revoked using the same `PUBLIC` option.

Syntax

```
REVOKE [GRANT OPTION FOR] privilege ON [TABLE] {tablename | viewname}
     FROM {object | userlist | rolelist | GROUP UNIX_group}
   | EXECUTE ON PROCEDURE procname FROM {object | userlist}
   | role_granted FROM {PUBLIC | role_grantee_list}};
<privileges> = ALL [PRIVILEGES] | privilege_list
<privilege_list> = {
     SELECT
   | DELETE
   | INSERT
   | UPDATE [(col [, col ...])]
   | REFERENCES [(col [, col ...])]
   }[, privilege_list ...]
<object> = {
     PROCEDURE procname
   | TRIGGER trigname
   | VIEW viewname
   | PUBLIC
   }[, object ...]
<userlist> = [USER] username [, [USER] username ...]
<rolelist> = rolename [, rolename]
<role_granted> = rolename [, rolename ...]
<role_grantee_list> = [USER] username [, [USER] username ...]
```

`privilege_list`	Name of privilege to be granted; valid options are `SELECT`, `DELETE`, `INSERT`, `UPDATE` and `REFERENCES`.
`GRANT OPTION FOR`	Removes grant authority for privileges listed in the `REVOKE` statement from userlist; cannot be used with object.
`col`	Column for which the privilege is revoked.
`tablename`	Name of an existing table for which privileges are revoked.
`viewname`	Name of an existing view for which privileges are revoked.
`GROUP unix_group`	On a UNIX system, the name of a group defined in `/etc/group`.
`object`	Name of an existing database object from which privileges are to be revoked.
`userlist`	A list of users from whom privileges are to be revoked.
`rolename`	An existing role created with the `CREATE ROLE` statement.
`role_grantee_list`	A list of users to whom `rolename` is granted; users must be in the Firebird/InterBase .

Examples

To revoke `INSERT` and `UPDATE` privileges from Janet and John:

```
REVOKE INSERT, UPDATE
   ON PROJ_DEPT_BUDGET
   FROM Janet, John
```

To revoke all privileges from every user, use the `PUBLIC` option, for example:

```
REVOKE ALL
   ON PROJ_DEPT_BUDGET
   FROM PUBLIC;
```

See also:
Grant Manager
User Manager
DDL - Data Definition Language
DML - Data Manipulation Language
Data Retrieval
Data Transaction

JOIN

In practice it seldom occurs that all relevant information can be found in a single database table. It is much more often the case that the data required is distributed across several tables and linked by relations. Indeed, information in a normalized database should be spread across multiple tables!

In a fully normalized database, the vast majority of tables have a primary key consisting of one or two columns only. If a referential integrity relationship exists, these primary key columns are replicated in other tables to ensure consistency in the data. These are the columns that allow you to establish logical links between these tables. When queries are performed, tables are commonly joined on these columns.

There is actually no restriction by design to the number of tables that may be joined. However the task of joining tables is exponential in relation to the number of tables in the join. The largest practical number of tables in a join is about 16, but experiment with your application and a realistic volume of data to find the most complex join that has an acceptable performance.

When you establish a join, InterBase/Firebird looks for matching values in the designated columns of each table. It does not care if a value appears once on one side of the join and multiple times on the other side, as is often the case.

In this instance, InterBase/Firebird joins each matching row in `TableB` to the single matching row in `TableA`, thereby creating what is known as a virtual row. Each `TableB` row can logically be linked to a single unambiguous row in `TableA`.

InterBase/Firebird also provides options for establishing a relationship where a value can appear on one side of the join instead of both. This is known as an `OUTER JOIN`.

The following statement selects from both `TableA` and `TableB` tables:

```
SELECT column_list
FROM TableA, TableB;
```

When you select from two or more tables, these tables are normally joined on a common column. For example, you might join `TableA` and `TableB` tables on the column that is common to each of them, the `TableA_ID`.

Theoretically it is not necessary to specify a join column. If you do not specify one, InterBase/Firebird performs a Cartesian product between the two tables, joining each row in one table to each row in the other. So, for example, if the first table had 100 rows, and the second had 20, the result set would have 2000 rows. Such a join normally makes no sense because the row information in one table is not logically related to the row information in the other table, except where column and field values are shared between the tables.

InterBase/Firebird does not prevent you from establishing a meaningless join. You can issue an SQL statement that joins, for example, `Orders.PaymentMethod` with `ustomer.Country`, and InterBase/Firebird processes the statement! But the result set is always empty because there are no matching values in either column.

`JOIN` syntax

InterBase/Firebird currently supports two methods to link two or more tables via a common column:

- the traditional SQL syntax, and
- the SQL '92 syntax.

The traditional SQL syntax integrates the link in the `WHERE` clause:

```
SELECT <ColumnList>
     FROM Table1 Synonym1 , Table2 Synonym2
     WHERE Synonym1.JoinColumn = Synonym2.JoinColumn
     AND <Other_WHERE_Conditions> ;
```

The following example illustrates this syntax:

```
SELECT C.Name, C.Country, O.OrderID, O.SaleDate, O.TotalInvoice
     FROM Customer C, Orders O
     WHERE C.CustomerID = O.CustomerID
     AND C.Country != 'U.S.A.'
     ORDER BY C.Name, O.OrderID;
```

As opposed to traditional SQL syntax, the SQL 92 syntax detaches the link from the `WHERE` clause and relocates it in the `FROM` clause, i.e. that area, in which the tables to be used are defined:

```
SELECT <ColumnList>
     FROM Table1 Alias1 JOIN Table2 Alias2
              ON Alias1.Column = Alias2.Column
     WHERE <Where_Conditions> ;
```

Example

```
SELECT C.Name, C.Country, O.OrderID, O.SaleDate, O.TotalInvoice
     FROM Customer C JOIN Orders O
```

```
        ON   C.CustomerID = O.CustomerID )
    WHERE C.Country != 'U.S.A.'
  ORDERBY C.Name, O.OrderID;
```

Either syntax can be used at any time; they are virtually interchangeable. The difference is that the SQL 92 syntax permits `OUTER JOINS`, whereas the traditional syntax does not.

Specifying columns and rows

When two or more tables are joined, rows can be included from either table in the result. It is also possible to specify `WHERE` conditions to limit the rows in either table that are considered for the join.

For example, the following statement asks for customers in Florida who placed orders in 1994 with a total invoice of more than $5,000 for the order:

```
SELECT C.Name, C.City, O.SaleDate, O.TotalInvoice
FROM Customer C JOIN Orders O
ON C.CustomerID = O.CustomerID
WHERE C.State_Province = 'FL'
AND O.SaleDate BETWEEN '1/1/94' AND '12/31/94'
AND O.TotalInvoice > 5000;
```

Please refer to Joining more than two tables for further information.

INNER JOIN

When you join two tables, the result set includes only those rows where the joining value appears in both tables.

Syntax

```
TableA JOIN TableB
```

The join applies to the table written to the left of the command.

For example, the following query joins `Stock` to `LineItem` to find out many orders included each stock item:

```
SELECT S.StockID, COUNT( L.OrderID )
FROM Stock S JOIN Lineitem L
ON S.StockID = L.StockID
GROUP BY S.StockID
```

From a theoretical standpoint, this is known as an `INNER JOIN`, but the `INNER` keyword is optional. What if you also want to include those stock items that have not yet been ordered, so that the result set shows all stock items. These items do not appear in the `LineItem` table at all. The solution lies in performing an `OUTER JOIN`. An outer join includes every column in one table and a subset of columns in the other table.

OUTER JOIN

When you join two tables, the result set includes only those rows where the joining value appears in both tables.

There are three types of outer joins:

SQL92 syntax permits outer joins, whereas the traditional syntax does not.

Types of outer joins

- `LEFT OUTER JOIN`, which includes all rows from the table on the left side of the join expression.
- `RIGHT OUTER JOIN`, which includes all rows from the table on the right side of the join expression.
- `FULL OUTER JOIN`, which includes all rows from both tables.

Syntax

```
TableA LEFT OUTER JOIN TableB
```

The join applies to the table written to the left of the command.

```
TableA RIGHT OUTER JOIN TableB
```

The join applies to the table written to the right of the command.

When your tables are linked in a referential relationship on a foreign key column, only the `LEFT OUTER JOIN` usually makes sense. For example, every order includes a customer from the `Customer` table. If you join `Customer` to `Orders` with a `RIGHT OUTER JOIN`, the result is the same as if you had performed an `INNER JOIN`.

The following query modifies the preceding example to include all stock items, even the one that have not yet been ordered:

```
SELECT S.StockID, COUNT( L.OrderID )
FROM Stock S LEFT OUTER JOIN Lineitem L
```

```
ON S.StockID = L.StockID
GROUP BY S.StockID
```

Adding selection criteria

If two tables are joined using an outer join, and there are also selection criteria in the table where the inclusion operator is placed, it would appear as first glance that you are asking two conflicting questions.

Consider the following query, which asks for the value of all orders placed by customers located in California, including those customers who might not have placed an order.

```
SELECT C.Name, SUM( O.TotalInvoice )
FROM Customer C LEFT OUTER JOIN Orders O
ON C.CustomerID = O.CustomerID
WHERE C.State_Province = 'CA'
GROUP BY C.Name;
```

On the one hand, the `LEFT OUTER JOIN` is asking InterBase/Firebird to include all customers in the result set, whether or not that customer has also placed any orders. On the other hand, the query is also asking InterBase/Firebird to limit the query to only those customers located in California.

InterBase/Firebird resolves this apparent conflict by always processing the `WHERE` clause before processing any outer joins. The `Customer` table is first limited to those customers in California, and this intermediate result is then joined to the `Orders` table to which of the California customers have placed orders.

CROSS JOIN

`CROSS JOIN` was introduced in Firebird 2.0. Logically, this syntax pattern:

```
A CROSS JOIN B
```

is equivalent to either of the following:

```
A INNER JOIN B ON 1 = 1
```

or, simply:

```
FROM A, B
```

Joining more than two tables

The SQL92 join syntax provides for joins that reference more than two tables. The trick is to establish the join with the first pair of tables, then join this product with the third table, and so on.

For example, the following query finds customers and the order details, where the order included a specific stock item:

```
SELECT C.Name, O.SaleDate, L.Quantity
FROM Customer C JOIN Orders O
ON ( C.CustomerID = O.CustomerID )
JOIN LineItem L
ON ( O.OrderID = L.OrderID )
WHERE L.StockID = '5313';
```

This syntax can be extended to any number of tables. You can even create a circular join. For example, the following statement asks for customers who have ordered products that were made by vendors in the same state as the customer. This query requires a series of joins from `Customer` to `Orders` to `LineItem` to `Stock` to `Vendors`, and another join from the `Customer` state to the `Vendor`'s state.

```
SELECT DISTINCT C.Name, V.VendorName, C.State_Province
FROM Customer C JOIN Orders O
ON ( C.CustomerID = O.CustomerID )
JOIN LineItem L
ON ( O.OrderID = L.OrderID )
JOIN Stock S
ON ( L.StockID = S.StockID )
JOIN Vendors V
ON ( S.VendorID = V.VendorID )
AND ( C.State_Province = V.State_Province );
```

Note an important limitation in this `SELECT` statement: tables are added to the `JOIN` expression one at a time. You cannot reference columns from a table until the table has been joined to the expression. For example, the condition linking the `Customer` and `Vendor` tables on their `State` columns cannot be specified until the `Vendor` table has been added to the expression and correctly joined.

Self joins / reflexive joins

A self-join, also known as a reflexive join, is a join in which a table is joined to itself. It compares rows of data within a single table. For example, we could add another column to the employee table in the sample employee database that would contain the employee's manager number. Since managers are also stored in the employee table, we could create a self-join on the employee table to determine the name of each employee's manager.

```
SELECT e1.full_name AS Employee, e2.full_name AS Manager
FROM employee e1 JOIN employee e2
ON e1.mng_id = e2.emp_no;
```

See also:
View
Query Builder

Stored procedure and trigger language

The InterBase/Firebird procedure and trigger language includes all the constructs of a basic structured programming language, as well as statements unique to working with table data. The SQL SELECT, INSERT, UPDATE and DELETE statements can be used in stored procedures exactly as they are used in a query, with only minor syntax changes. Local variables or input parameters can be used for all of these statements in any place that a literal value is allowed. Certain constructs, including all DDL (Data Definition Language) statements, are omitted.

Firebird 2.0 introduced high performance cursor processing, for cursors originating from a SELECT query and for cursors originating from a selectable stored procedure.

Because PSQL programs run on the server, data transfer between the relational core and the PSQL engine is very fast, much faster than transfer to a client application.

Other statements that are specific to stored procedures include, among others, error handling and raising exceptions. Please refer to the relevant sections for further information.

Note that the string concatenation operator in InterBase/Firebird procedure and trigger language is || (a double vertical bar, or pipe), and not the + that is used in many programming languages. Please refer to concatenation of strings for further information.

Within a trigger or stored procedure, statements are separated by semicolons.

For further reading, particularly for those new to PSQL, please refer to *Writing stored procedures and triggers*.

Summary of PSQL commands

Command	Description
BEGIN <statements> END	Compound statement like in PASCAL.
variable = expression	Assignment. variable can be a local variable, an in or an out parameter.
compound_statement	A single command or a BEGIN/END block.
select_statement	Normal SELECT statement. The INTO clause must be present at the end of the statement. Variable names can be used with a colon preceding them. Example: SELECT PRICE FROM ARTICLES WHERE ARTNO = :ArticleNo INTO :EPrice
/* Comment */	Comment, like in C.
-- Comment	Single line SQL comment.
DECLARE VARIABLE name datatype [= startval]	Variable declaration. After AS, before the first BEGIN.
EXCEPTION	Re-fire the current exception. Only makes sense in a WHEN clause.
EXCEPTION name [message]	Fire the specified exception. Can be handled with WHEN.
EXECUTE PROCEDURE name arg, arg RETURNING_VALUES arg, arg	Calling a procedure. arg's must be local variables. Nesting and recursion allowed.
EXIT	Leaves the procedure (like in PASCAL).
FOR select_statement DO compound_statement	Executes compound_statement for every line that is returned by the SELECT statement.
IF (condition) THEN compound_statement [ELSE compound_statement]	IF statement, like in PASCAL.
POST_EVENT name	Posts the specified event.
SUSPEND	Only for SELECT procedures which return tables: Waits for the client to request the next line. Returns the next line to the client.
WHILE (condition) DO compound_statement	WHILE statement. Like in PASCAL.

WHEN {EXCEPTION a | SQLCODE x | ANY} DO compound_statement	Exception handling. WHEN statements must be at the end of the procedure, directly before the final END.
EXECUTE STATEMENT stringvalue	Executes the DML statement in stringvalue.
EXECUTE STATEMENT stringvalue INTO variable_list	Executes the statement and returns variables (singleton).
FOR EXECUTE STATEMENT stringvalue INTO variable_list DO compound_statement	Executes the statement and iterates through the resulting lines.

(Source: Stored Procedures in Firebird by Stefan Heymann, 2004)

A complete Firebird 2.0 PSQL Language Reference including expressions, conditions and statements can be found at: http://www.janus-software.com/fbmanual/index.php?book=psql.

The most important items are listed in detail below.

Supported Firebird 2 features

Since IBExpert version 2005.03.12 the following Firebird 2 features are also supported:

- DECLARE <cursor_name> CURSOR FOR ...
- OPEN <cursor_name>
- FETCH <cursor_name> INTO ..
- CLOSE <cursor_name>
- LEAVE <label>
- NEXT VALUE FOR <generator>

There are a number of further enhancements to PSQL in Firebird 2.0. Please refer to the Firebird 2.0.4 Release Notes chapter, Stored Procedure Language (PSQL), for details.

Using DML statements

The SQL Data Manipulation Language (DML), consists primarily of the SELECT, INSERT, UPDATE and DELETE statements.

Statements that are not recognized or permitted in the stored procedures and trigger language include DDL statements such as CREATE, ALTER, DROP, and SET as well as statements such as GRANT, REVOKE, COMMIT, and ROLLBACK.

Wherever a literal value is specified in an INSERT, UPDATE or DELETE statement, an input or local variable can be substituted in place of this literal. For example, variables can be used for the values to be inserted into a new row, or the new values in an UPDATE statement. They can also be used in a WHERE clause, to specify the rows that are to be updated or deleted.

Since Firebird 2.0, the SQL language extension EXECUTE BLOCK makes "dynamic PSQL" available to SELECT specifications. It has the effect of allowing a self-contained block of PSQL code to be executed in dynamic SQL as if it were a stored procedure. For further information please refer to EXECUTE BLOCK statement.

Using SELECT statements

InterBase/Firebird supports an extension to the standard SELECT statement, to solve the problem of what to do with the results when using a SELECT statement inside a stored procedure. The INTO clause appoints variables that receive the results of the SELECT statement. The syntax is as follows:

```
SELECT <result1, result2, ..., resultN>
FROM ...
WHERE ...
GROUP BY ...
INTO : <Variable1, : Variable2, ..., VariableN>;
```

The INTO clause must be the final clause in the SELECT statement. A variable must be given for each result generated by the statement. *Important:* this form of SELECT statement can generate only one row. Therefore the ORDER BY clause is unnecessary here.

To use a SELECT that generates more than one row within a stored procedure, use the FOR SELECT statement.

New to Firebird 2.0: support for derived tables in DSQL (subqueries in FROM clause) as defined by SQL200X. A derived table is a set, derived from a dynamic SELECT statement. Derived tables can be nested, if required, to build complex queries and they can be involved in joins as though they were normal tables or views.

Syntax

```
SELECT
   <select list>
FROM
   <table reference list>

   <table reference list> ::= <table reference> [{<comma> <table reference>}...]

   <table reference> ::=
      <table primary>
    | <joined table>
```

```
<table primary> ::=
   <table> [[AS] <correlation name>]
     | <derived table>

<derived table> ::=
   <query expression> [[AS] <correlation name>]
     [<left paren> <derived column list> <right paren>]

<derived column list> ::= <column name> [{<comma> <column name>}...]
```

Examples can be found in the Data Manipulation Language chapter.

Points to Note

- Every column in the derived table must have a name. Unnamed expressions like constants should be added with an alias or the column list should be used.
- The number of columns in the column list should be the same as the number of columns from the query expression.
- The optimizer can handle a derived table very efficiently. However, if the derived table is involved in an inner join and contains a subquery, then no join order can be made.

SET TERM terminator or terminating character

Normally InterBase processes a script step by step and separates two statements by a semicolon. Each statement between two semicolons is parsed, interpreted, converted into an internal format and executed. This is not possible in the case of stored procedures or triggers where there are often multiple commands which need to be successively executed, i.e. there are several semicolons in their source codes. So if `CREATE PROCEDURE ...` was called, InterBase/Firebird assumes that the command has finished when it arrives at the first semi colon.

In order for InterBase/Firebird to correctly interpret and transfer a stored procedure to the database, it is necessary to temporarily alter the terminating character using the `SET TERM` statement. The syntax for this is as follows (Although when using the IBExpert templates this is not necessary, as IBExpert automatically inserts the `SET TERM` command):

```
SET TERM NEW_TERMINATOR OLD_TERMINATOR
```

Example

```
SET TERM ^;
CREATE PROCEDURE NAME
   AS
     BEGIN
     <procedure body>;
   END^
SET TERM ;^
```

Before the first `SET TERM` statement appears, InterBase/Firebird regards the semicolon as the statement terminating character and interprets and converts the script code up until each semicolon.

Following the first `SET TERM` statement, the terminator is switched and all following semicolons are no longer interpreted as terminators. The `CREATE PROCEDURE` statement is then treated as one statement up until the new terminating character, and parsed and interpreted. The final `SET TERM` statement is necessary to change the terminating character back to a semicolon, using the syntax:

```
SET TERM OLD_TERMINATOR NEW_TERMINATOR
```

(refer to above example: `SET TERM ;^`).

The statement must be concluded by the previously defined temporary termination character. This concluding statement is again interpreted as a statement between the two last termination characters. Finally the semicolon becomes the termination character for use in further script commands.

It is irrelevant which character is used to replace the semi colon; however it should be a seldom-used sign to prevent conflicts e.g. `^`, and not `*` or `+` (used in mathematical formulae) or `!` (this is used for "not equal": `A!=B`).

SUSPEND

`SUSPEND` is used in stored procedures; It is used to return a row of data from a procedure to its caller. It acts as if it was a data set, i.e. returns the named data set visually as a result.

It suspends procedure execution until the next `FETCH` is issued by the calling application and returns output values, if there are any, to the calling application. It prevents the stored procedure from terminating until the client has fetched all the results. This statement is not recommended for executable procedures.

Syntax

```
<suspend_stmt> ::=
   SUSPEND ;
```

Suspends execution of a PSQL routine until the next value is requested by the calling application, and returns output values, if any, to the calling application. If the procedure is called from a `SELECT` statement, processing will continue following `SUSPEND` when the next row of data is needed. Use the `EXIT` statement or let the code path end at the final `END` of the body to signal that there are no more rows to return.

If the procedure is called from a `EXECUTE PROCEDURE` statement, then `SUSPEND` has the same effect as `EXIT`. This usage is legal, but not recommended.

BEGIN and END statement

As well as defining the contents of the stored procedure, these keywords also delimit a block of statements which then executes as a single statement. This means that BEGIN and END can be used to enclose several statements and so form a simple compound statement. Unlike all other PSQL statements, a BEGIN ... END block is not followed by a semicolon.

DECLARE VARIABLE

Please refer to local variables.

FOR EXECUTE INTO

Use the FOR EXECUTE INTO statement to execute a (can also be dynamically created) SELECT statement contained in a string and process all its result rows.

The execute SQL statement allows the execution of dynamically constructed SELECT statements. The rows of the result set are sequentially assigned to the variables specified in the INTO clause, and for each row the statement in the DO clause is executed.

To work with SELECT statements that return only a single row, consider using the EXECUTE INTO statement.

It is not possible to use parameter markers (?) in the SELECT statement as there is no way to specify the input actuals. Rather than using parameter markers, dynamically construct the SELECT statement, using the input actuals as part of the construction process.

FOR SELECT ... DO ...

The FOR SELECT DO statement allows the compact processing of a SELECT statement. The rows of the result set are sequentially assigned to the variables specified in the INTO clause, and for each row the statement in the DO clause is executed.

If the AS CURSOR clause is present, the select statement is assigned a cursor name. The current row being processed by the FOR SELECT DO statement can be referred to in DELETE and UPDATE statements in the body of the FOR SELECT DO by using the WHERE CURRENT OF clause of those statements.

Examples can be found in *Writing stored procedures and triggers*.

IF THEN ELSE

A condition is evaluated and if it evaluates to TRUE the statement in the THEN clause is executed. If it is not TRUE, i.e. It evaluates to FALSE or to NULL, and an ELSE clause is present, then the statement in the ELSE clause is executed.

IF statements can be nested, i.e. The statements in the THEN or ELSE clauses can be IF statements also. If the THEN clause contains a IF THEN ELSE statement, then that ELSE clause is deemed to be part of the nested IF, just as in nearly all other programming languages. Enclose the nested IF in a compound statement if you want the ELSE clause to refer to the enclosing IF statement.

```
variable = expression;
```

The variable can be an input or output parameter, or a local variable defined in a DECLARE VARIABLE statement. The expression needs to be concluded with a semicolon. The syntax for the IF statement is as follows:

```
IF <conditional_test>
THEN
<statements>;
ELSE
<statements>;
```

Any of the standard comparison operators available in SQL an be used (please refer to comparison operators for a full list).

The value can be a constant or one of the input parameters, output parameters or local variables used in the procedure.

If a single statement is placed after the THEN or ELSE clauses, it should be terminated with a semicolon.

If multiple statements need to be placed after one of these clauses, use the BEGIN and END keywords as follows:

```
IF <conditional_test> THEN
BEGIN
<statement1>;
<statement2>;
...
<statementN>;
END
ELSE
etc.;
```

WHILE and DO

The WHILE … DO statement provides a looping capability. The syntax for this statement is as follows:

```
WHILE
<conditional_test>
```

```
DO
<statements>;
```

InterBase/Firebird evaluates the conditional test. If it is TRUE, the statements following the WHILE are executed. If it is FALSE, the statements are ignored. If only one statement is placed after the DO clause, it should be terminated with a semicolon. If multiple statements are used after one of these clauses, use the BEGIN and END keywords. Brackets need to be put around the conditional test.

OPEN CURSOR

New to Firebird 2.0, the OPEN statement allows you to open a local cursor.

Syntax

```
<open_stmt> ::=
   OPEN <cursor_name>;

<cursor_name> ::=   <identifier>
```

where cursor_name is the name of a local cursor.

The OPEN statement opens a local cursor. Opening a cursor means that the associated query is executed and the that the result set is kept available for subsequent processing by the FETCH statement. The cursor must have been declared in the declarations section of the PSQL program.

Attempts to open a cursor that is already open, or attempts to open a named FOR SELECT cursor will fail and generate a runtime exception. All cursors which were not explicitly closed will be closed automatically on exit from the current PSQL program.

Please also refer to *Explicit cursors* in the *Firebird 2.0.4 Release Notes*.

See also:
Comments
Comparison Operators
Conditional Test
Writing stored procedures and triggers
Firebird 2 SQL Reference Guide

Firebird 2 Quick Start Guide

IBPhoenix Editors
Firebird Project members
8 April 2008, document version 3.7 — covers Firebird 2.0–2.0.4 and 2.1

About this guide

The Firebird Quick Start Guide is an introduction for the complete newcomer to a few essentials for getting off to a quick start with a Firebird binary kit. The guide first saw the light as Chapter 1 of the *Using Firebird* manual, sold on CD by http://www.IBPhoenix.com. Later it was published separately on the Internet. In June 2004, IBPhoenix donated it to the Firebird Project. Since then it is maintained, and regularly updated, by members of the Firebird documentation project.

Important Before you read on, verify that this guide matches your Firebird version. This guide covers versions 2.0–2.0.4 and 2.1. For all other Firebird versions, get the corresponding Quick Start Guide at http://www.firebirdsql.org/?op=doc.

Some warnings before you start

- Firebird 2.0.2 was recalled due to a regression; if you use it, upgrade to 2.0.3 or higher ASAP and make sure to read your new version's Release Notes.
- If you want to rely on Linux forced writes to work correctly, upgrade to at least 2.0.4.

What is in the kit?

All of the kits contain all of the components needed to install the Firebird server:

- The Firebird server executable.
- One or more client libraries.
- The command-line tools.
- The standard user-defined function libraries.
- A sample database.
- The C header files (not needed by beginners).
- Release notes – ESSENTIAL READING!

Classic or Superserver?

Firebird comes in two flavours, called *architectures*: Classic Server and Superserver. Which one should you install? That depends on your situation. A short overview of the most important differences follows.

Table 1. Firebird 2 Classic Server vs. Superserver

	Classic Server	Superserver
Processes	Creates a separate process for every client connection, each with its own cache. Less resource use if the number of connections is low.	A single process serves all connections, using threads to handle requests. Shared cache space. More efficient if the number of simultaneous connections grows.
Local connections	Permits fast, direct I/O to database files for local connections on Linux. The client process must have filesystem-level access rights to the database for this to work.	On Linux, all local connections are made via the network layer, using localhost (often implicitly). Only the server process needs access rights to the database file.
	On Windows, both architectures now support safe and reliable local connections, with only the server process requiring access rights to the database file.	
Multiprocessor	SMP (symmetrical multi-processor) support. Better performance in case of a small number of connections that do not influence each other.	No SMP support. On multi-processor Windows machines, performance can even drop dramatically as the OS switches the process between CPUs. To prevent this, set the `CpuAffinityMask` parameter in the configuration file `firebird.conf`.
Guardian	When run as a Windows application (as opposed to a service) you can't use the Firebird Guardian. Note that running Firebird as an application is the only option on Windows 9x–ME.	Can be used with the Guardian on Windows, whether run as an application or as a service.

As you can see, neither of the architectures is better in all respects. This is hardly surprising: we wouldn't maintain two separate architectures if one of them was an all-fronts loser.

If you're still not sure what to choose (maybe you find all this tech talk a little overwhelming), use this rule of thumb:

- On Windows, choose Superserver.
- On Linux, just pick one or the other. In most circumstances, chances are that you won't notice a performance difference.

Note that you can always switch to the other architecture later; your applications and databases will keep functioning like before.

For Linux, Superserver download packages start with FirebirdSS, Classic packages with FirebirdCS. For Windows, there is a combined installation package; you choose the architecture during the installation process.

Embedded Server for Windows

On Windows platforms only, Firebird offers a third flavor: *Embedded Server*, a client and server rolled into one DLL for ease of deployment. While very practical, it lacks most of Firebird's usual security features. For more information on Firebird Embedded Server, consult the *Clients and Servers* chapter in *Using Firebird*:

http://www.firebirdsql.org/manual/ufb-cs-embedded.html (HTML)
http://www.firebirdsql.org/pdfmanual/Using-Firebird_(wip).pdf (PDF)

The Embedded Server comes in a separate download package.

Default disk locations

Linux

The following table shows the default component locations of a Firebird installation on Linux. Some of the locations may be different on other Unix-like systems.

Table 2. Firebird 2 component locations on Linux

Component	File Name	Default Location
Installation directory (referred to hereafter as `<InstallDir>`)	–	`/opt/firebird`
Release Notes and other documentation	various files	`<InstallDir>/doc`
Firebird server	`fbserver` (SS) or `fb_inet_server` (CS)	`<InstallDir>/bin`
Command-line tools	`isql, gbak, nbackup, gsec, gfix, gstat, etc.`	`<InstallDir>/bin`
Sample database	`employee.fdb`	`<InstallDir>/examples/empbuild`
UDF libraries	`ib_udf.so, fbudf.so`	`<InstallDir>/UDF`
Additional server-side libraries	`libicu*.so, libib_util.so`	`<InstallDir>/bin`
Client libraries	`libfbclient.so.2.`*`m.n#` (network client) `libfbembed.so.2.`*`m.n` (local client with embedded engine, Classic only) The usual symlinks (`*.so.2`,`*.so`) are created. Legacy `libgds.*` symlinks are also installed.	`/usr/lib` (actually, the real stuff is in `<InstallDir>/lib`, but you should use the links in `/usr/lib`)

Windows

In the table below, `<ProgramDir>` refers to the Windows programs folder. This is usually `C:\Program Files` but may also be a different path, e.g. `D:\Programmi`. Likewise, `<SystemDir>` refers to the Windows system directory. Be sure to read the notes below the table, especially if you're running Firebird on a 64-bit Windows system.

Table 3. Firebird 2 component locations on Windows

Component	File Name	Default Location
Installation directory (referred to hereafter as `<InstallDir>`)	—	`<ProgramDir>\Firebird\Firebird_2_0`
Release Notes and other documentation	Various files	`<InstallDir>\doc`
Firebird server	`fbserver.exe` (SS) or `fb_inet_server.exe` (CS)	`<InstallDir>\bin`
Command-line tools	`isql.exe`, `gbak.exe`, `nbackup.exe`, `gsec.exe`, `gfix.exe`, `gstat.exe`, etc.	`<InstallDir>\bin`
Sample database	`employee.fdb`	`<InstallDir>\examples\empbuild`
User-defined function (UDF) libraries	`ib_udf.dll`, `fbudf.dll`	`<InstallDir>\UDF`
Additional server-side libraries	`icu*.dll`, `ib_util.dll`	`<InstallDir>\bin`
Client libraries	`fbclient.dll` (with an optional `gds32.dll`, to support legacy apps)	`<InstallDir>\bin` (with an optional copy in `<SystemDir>` – see note below table)

The Windows system directory

The exact path to the Windows System directory depends on your Windows version. Typical locations on 32-bit systems are:

- for Windows 95/98/ME: `C:\Windows\System`
- for Windows NT/2000: `C:\WINNT\System32`
- for Windows XP: `C:\Windows\System32`

For 64-bit systems, read the next note.

Important notice for 64-bit Windows users

On 64-bit Windows systems, the "`Program Files`" directory is reserved for 64-bit programs. If you try to install a 32-bit application into that folder, it will be auto-redirected to a directory which – in English versions – is called "`Program Files (x86)`". In other language versions the name may be different.

In the same vein, the `System32` directory is reserved for 64-bit libraries. 32-bit libraries go into `SysWOW64`. That's right: 64-bit libraries are in `System32`, 32-bit libraries in `SysWOW64`.

If you're not aware of this, you may have a hard time locating your 32-bit Firebird components on a 64-bit Windows system.

(Incidentally, WOW stands for Windows on Windows. Now you can also work out what LOL means.)

Installing Firebird

The instructions given below for the installation of Firebird on Windows and Linux should be sufficient for the vast majority of cases. However, if you experience problems or if you have special needs not covered here, be sure to read the *INSTALLATION NOTES* chapter in the *Release Notes*. This is especially important if you are upgrading from a previous version or if there are remnants of an old (and maybe long gone) InterBase or Firebird installation floating around your system (DLLs, Registry entries, environment variables...).

Installing the Firebird server

Installation drives

Firebird server – and any databases you create or connect to – must reside on a hard drive that is physically connected to the host machine. You cannot locate components of the server, or any database, on a mapped drive, a filesystem share or a network filesystem.

Note: You can mount a read-only database on a CD-ROM drive but you cannot run Firebird server from one.

Installation script or program

Although it is possible to install Firebird by a filesystem copying method – such as "untarring" a snapshot build or decompressing a structured `.zip` archive – it is strongly recommended that you use the distributed release kit (`.exe` for Windows, `.rpm` for Linux), especially if this is the first time you install Firebird. The Windows installation executable, the Linux `rpm` program and the `install.sh` script in the official `.tar.gz` for various Posix platforms all perform some essential setup tasks. Provided you follow the installation instructions correctly, there should be nothing for you to do upon completion but log in and go!

Installing on Windows

The Firebird installer lets you choose between Superserver and Classic Server installation. Both are fully mature and stable and there is no reason to categorically prefer one to the other. Of course you may have your own specific considerations.

If you install Firebird under Windows 95/98/ME, uncheck the option to install the *Control Panel* applet. It doesn't work on these platforms. You'll find a link to a usable applet further down. (Note: the option to install the applet is only available for Superserver.)

On Windows server platforms – NT, 2000, 2003 and XP – Firebird will run as a system service by default, but during the installation you can also choose to let it run as an application. Non-server Windows systems – 95, 98 and ME – don't support services; running as an application is the only option there.

Use the Guardian?

The Firebird Guardian is a utility that monitors the server process and tries to restart it if it terminates abnormally. The Guardian does not work with Firebird Classic Server on Windows if run as an application. This is due to a known bug, which will be fixed later. Currently the Firebird 2 installer doesn't give you the option to include the Guardian at all with a Classic Server, even if you install it as a service.

The Guardian works correctly with Superserver, whether run as an application or as a service.

If you run Firebird *as a service* on Windows 2000, 2003 or XP, the Guardian is a convenience rather than a necessity, since these operating systems have the facility to watch and restart services. It is recommended that you keep the Guardian option on (if possible) in all other situations.

Warning

If you install Firebird 2.0.3 (and probably earlier 2.0 versions too) on Windows without the Guardian, the installer doesn't correctly detect an already running server. This leads to errors when it tries to overwrite existing DLLs and executables. So, in the above case, make sure to uninstall any existing Firebid server before attempting to install the new one. This bug has been fixed in versions 2.0.4 and 2.1.

Installing on Linux and other Unix-like platforms

In all cases, read the *Release Notes* that came with your Firebird package (chapter *Installation Notes*, section *Posix Platforms*). There may be significant variations from release to release of any Posix operating system, especially the open source ones. Where possible, the build engineers for each Firebird version have attempted to document any known issues.

If you have a Linux distribution that supports rpm installs, consult the appropriate platform documentation for instructions about using RPM Package Manager. In most distributions you will have the choice of performing the install from a command shell or through a GUI interface.

For Linux distributions that cannot process rpm programs, and for Unix flavours for which no `.rpm` kit is provided, use the `.tar.gz` kit. You will find detailed instructions in the *Release Notes*. Shell scripts have been provided. In some cases, the Release Notes may instruct you to edit the scripts and make some manual adjustments.

Installing multiple servers

Firebird 2 allows the operation of multiple servers on a single machine. It can also run concurrently with Firebird 1.x or InterBase servers. Setting this up is not a beginner's task though. If you need to run multiple servers, consult the *Installation Notes* chapter of the *Release Notes*, and have the *Firebird 1.5 Release Notes* handy too – you will be directed to them at a certain point during your reading of the *Installation Notes*.

Testing the installation

If everything works as designed, the Firebird server process will be running on your server machine upon completion of the installation. It will also start up automatically whenever you restart your computer.

Before testing the Firebird server itself, it is advisable to verify if the server machine is reachable from the client at all. At this point, it is assumed that you will use the recommended TCP/IP network protocol for your Firebird client/server connections.

Notes:

- If you have installed a Classic Server on Linux/Unix or any Firebird server on Windows, it is possible to connect directly to the local server, without using a network layer. If you intend to use Firebird for this type of connection only, you can skip the Pinging the server section below.
- For information about using the NetBEUI protocol in an all-Windows environment, refer to the *Network Configuration* chapter in the *Using Firebird* manual sold by IBPhoenix, or consult the *InterBase 6 Operations Guide* (http://www.ibphoenix.com/downloads/60OpGuide.zip).
- Firebird does not support IPX/SPX networks.

Pinging the server

The ping command – available on most systems – is a quick and easy way to see if you can connect to a server machine via the network. For example, if your server's IP address in the domain that is visible to your client is 192.13.14.1, go to a command shell on the client machine and type the command

```
ping 192.13.14.1
```

substituting this example IP address with the IP address that your server is broadcasting. If you are on a managed network and you don't know the server's IP address, ask your system administrator. Of course you can also ping the server by its name, if you know it:

```
ping vercingetorix
```

If you are connecting to the server from a local client – that is, a client running on the same machine as the server – you can ping the virtual TCP/IP loopback server:

```
ping localhost –or– ping 127.0.0.1
```

If you have a simple network of two machines linked by a crossover cable, you can set up your server with any IP address you like except 127.0.0.1 (which is reserved for a local loopback server) and, of course, the IP address which you are using for your client machine. If you know the “native” IP addresses of your network cards, and they are different, you can simply use those.

Once you have verified that the server machine is reachable from the client, you can go on to the next step.

Checking that the Firebird server is running

After installation, Firebird server should be running:

On Linux or other Unix-like systems: As a service.

On Windows server systems (NT, 2000, 2003, XP): As a service or as an application. Service is default and highly recommended.

On Windows non-server systems (95, 98, ME): As an application.

The following sections show you how to test the server in each of these situations.

Server check: Linux and other Unices

Use the top command in a command shell to inspect the running processes interactively. If a Firebird Superserver is running, you should see a process named fbguard. This is the Guardian process. Further, there will be one main and zero or more child processes named `fbserver`.

The following screen shows the output of top, restricted by grep to show only lines containing the characters `fb`:

```
frodo:/inkomend/firebird # top -b -n1 | grep fb
2587 firebird 24 0 1232 1232 1028 S 0.0 0.3 0:00.00 fbguard
2588 firebird 15 0 4124 4120 2092 S 0.0 0.9 0:00.04 fbserver
2589 firebird 15 0 4124 4120 2092 S 0.0 0.9 0:00.00 fbserver
2604 firebird 15 0 4124 4120 2092 S 0.0 0.9 0:00.00 fbserver
2605 firebird 15 0 4124 4120 2092 S 0.0 0.9 0:00.02 fbserver
2606 firebird 15 0 4124 4120 2092 S 0.0 0.9 0:00.00 fbserver
2607 firebird 15 0 4124 4120 2092 S 0.0 0.9 0:00.00 fbserver
```

As an alternative to top, you can use `ps -ax` or `ps -aux` and pipe the output to `grep`.

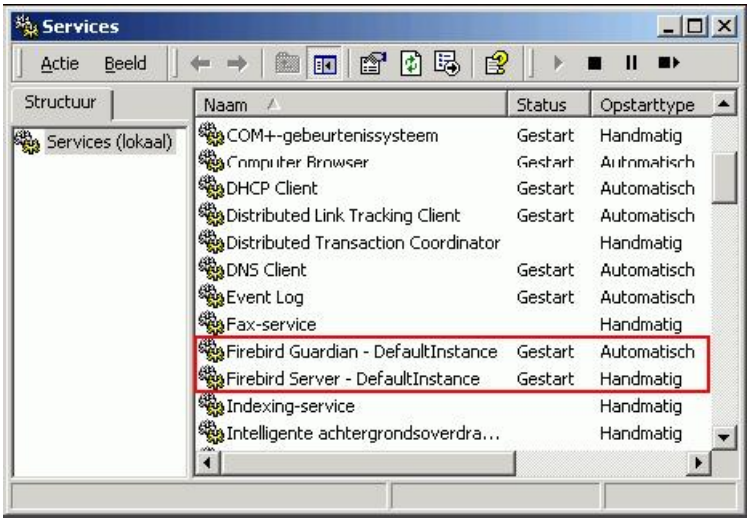
For Classic Server versions, the process name is `fb_inet_server`. There will be one instance of this process running for each network connection. Note that if there are no active connections, or if there are only direct local connections, you won't find `fb_inet_server` in the process list. `fb_lock_mgr` should be present though as soon as any kind of Classic connection has been established.

Other ways to test a Firebird server immediately after installation include connecting to a database, creating a database, and launching the `gsec` utility. All these operations are described later on in this guide.

Server check: Windows, running as service

Open *Control Panel -> Services (NT)* or *Control Panel -> Administrative Tools -> Services (2000, XP)*.

This illustration shows the Services applet display on Windows 2000. The appearance may vary from one Windows server edition to another. Also, service names may vary with the Firebird version.



You should at least find the Firebird server in the services listing. The Guardian may or may not be running, depending on the choices you made during installation.

Server check: Windows, running as application

If Firebird is up and running as an application, it is represented by an icon in the system tray:

- A green and grey server symbol if controlled by the Guardian;
- A round yellow and black graphic if running standalone.

A flashing icon indicates that the server is in the process of starting up (or at least trying to do so). A red icon, or an icon with an overlying red stop sign, indicates that startup has failed.

One way to make 100% sure if the server is running or not is to press [Ctrl-Alt-Del] and look for the `fbserver` or `fb_inet_server` process (and possibly `fbguard`) in the task list.

On some occasions, you may need to start the Guardian or server once explicitly via the Start menu even if you opted for "Start Firebird now## at the end of the installation process. Sometimes a reboot is necessary.

If you're desperately trying to start Firebird and nothing seems to work, ask yourself if you've installed Firebird 2 Classic server with the Guardian option enabled (the installation program doesn't offer this possibility anymore, but there are other ways). As said before, the combination Classic + Guardian currently doesn't work if Firebird runs as an application. Uninstall Firebird if necessary and reinstall Classic without Guardian, or Superserver with or without Guardian.

You can shut the server down via the menu that appears if you right-click on the tray icon. Notice that this also makes the icon disappear; you can restart Firebird via the Start menu.

Note: Windows Classic Server launches a new process for every connection, so the number of `fb_inet_server` processes will always equal the number of client connections plus one. Shutdown via the tray icon menu only terminates the first process (the *listener*). Other processes, if present, will continue to function normally, each terminating when the client disconnects from the database. Of course, once the listener has been shut down, new connections can't be made.

Performing a client-only install

Each remote client machine needs to have the client library – `libfbclient.so` on Posix clients, `fbclient.dll` on Windows clients – that matches the release version of the Firebird server.

Firebird versions from 1.5 onward can install symlinks or copies named after the 1.0 libs (with the “old” Inter-Base names), to maintain compatibility with third-party products which need these files.

Some extra pieces are also needed for the client-only install.

Windows

At present, no separate installation program is available to install only the client pieces on a Windows machine. If you are in the common situation of running Windows clients to a Linux or other Unix-like Firebird server (or another Windows machine), you need to download the full Windows installation kit that corresponds to the version of Firebird server you install on your server machine.

Fortunately, once you have the kit, the Windows client-only install is easy to do. Start up the installation program just as though you were going to install the server, but select one of the client-only options from the installation menu.

Linux and some other Posix clients

A small-footprint client install program for Linux clients is not available either. Additionally, some Posix flavours – even within the Linux constellation – have somewhat idiosyncratic requirements for filesystem locations. For these reasons, not all `*x` distributions for Firebird even contain a client-only install option.

For most Linux flavours, the following procedure is suggested for a Firebird client-only install. Log in as root for this.

1. Look for `libfbclient.so.2.m.n` (`m.n` being the minor plus patch version number) in `/opt/firebird/lib` on the machine where the Firebird server is installed. Copy it to `/usr/lib` on the client.
2. Create chained symlinks using the following commands:

   ln -s /usr/lib/libfbclient.so.2.m.n /usr/lib/libfbclient.so.2 ln -s /usr/lib/libfbclient.so.2 /usr/lib/libfbclient.so

   ...replacing `2.m.n` with your version number, e.g. `2.0.0` or `2.1.0`

If you're running applications that expect the legacy libraries to be present, also create the following symlinks:

```
ln -s /usr/lib/libfbclient.so /usr/lib/libgds.so.0
ln -s /usr/lib/libfbclient.so /usr/lib/libgds.so
```

3. Copy `firebird.msg` to the client machine, preferably into the `/opt/firebird` directory. If you place it somewhere else, create a system-wide permanent `FIREBIRD` environment variable pointing to the right directory, so that the API routines can locate the messages.
4. Optionally copy some of the Firebird command-line tools – e.g. isql – to the client machine. *Note:* always copy the tools from a Superserver kit, regardless of the architecture of the server(s) you're planning to connect to. Tools from Classic distributions terminate immediately if they can't find the `libfbembed` library (which is useless for network connections) upon program start.

Instead of copying the files from a server, you can also pull them out of a Firebird `tar.gz` kit. Everything you need is located in the `/opt/firebird` tree within the `buildroot.tar.gz` archive that's packed inside the kit.

Server configuration and management

There are several things you should be aware of – and take care of – before you start using your freshly installed Firebird server. This part of the manual introduces you to some useful tools and shows you how to protect your server and databases.

User management: gsec

Firebird comes with a command-line user management tool called `gsec`. Although its functions can also be performed by a number of third-party GUI utilities, you should at least have a basic knowledge of `gsec`, since this is the official tool and it's present in every Firebird server installation. In the next sections you will use `gsec` to execute two tasks: changing the SYSDBA password and adding a Firebird user. First though, some points of attention:

Permission to run `gsec`

With some Firebird installations, you can only run `gsec` if you are logged into the operating system as Superuser (root on Linux) or as the user the Firebird server process runs under. On Windows server platforms, you typically need to be in the Power User group or higher to run `gsec` successfully.

Trouble running `gsec`

If you have enough privileges but invoking `gsec` results in a message like *cannot attach to password database - unable to open database*:

- You may be running Firebird on Windows and for some reason the local protocol isn't working. One rather common cause for this is running Windows Vista, 2003 or XP with terminal services enabled. To enable the local protocol, open `firebird.conf`, uncomment the `IpcName` parameter and set it to `Global\FIREBIRD`. Then restart the server.
  Note: In Firebird 2.0.1 and up, `Global\FIREBIRD` is already the default on TS-enabled Windows systems.
- If the above doesn't apply to you, you can at least circumvent the problem by "tricking" `gsec` into using TCP/IP. Add the following parameter to the command line, adjusting the path if necessary:

  ```
  -database "localhost:C:\Program Files\Firebird\Firebird_2_0\security2.fdb"
  ```

  The file `security2.fdb` is the security database, where Firebird keeps its user account details. It is located in your Firebird installation directory.
- Maybe your security database is a renamed `security.fdb` from Firebird 1.5. Of course this can't be the case immediately after installation. Someone (you?) must have put it there, in order to keep the existing accounts available. Consult the *Release Notes* for instructions on how to upgrade old security databases.

If the error message starts with *Cannot attach to services manager*, the server may not be running at all. In that case, go back to Testing your installation and fix the problem.

Calling `gsec` on Linux

On `**nix` systems, if you call `gsec` from its own directory, you should type `./gsec` instead of just `gsec`. The current directory is usually not part of the search path, so plain `gsec` may either fail or launch a "wrong" `gsec`.

Changing the SYSDBA password

One Firebird account is created automatically as part of the installation process: SYSDBA. This account has all the privileges on the server and cannot be deleted. Depending on version, OS, and architecture, the installation program will either

- install the SYSDBA user with the password `masterkey` (actually, `masterke`: characters after the eighth are ignored), or
- ask you to enter a password during installation, or
- generate a random password and store that in the file `SYSDBA.password` within your Firebird installation directory.

If the password is `masterkey` and your server is exposed to the Internet at all – or even to a local network, unless you trust every user with the SYSDBA password – you should change it immediately using the `gsec` command-line utility. Go to a command shell, `cd` to the Firebird `bin` subdirectory and issue the following command to change the password to (as an example) `icuryy4me`:

```
gsec -user sysdba -pass masterkey -mo sysdba -pw icuryy4me
```

Notice that you specify "sysdba" twice in the command:

- With the `-user` parameter you identify yourself as SYSDBA. You also provide SYSDBA's current password in the `-pass` parameter.
- The `-mo[dify]` parameter tells `gsec` that you want to modify an account – which happens to be SYSDBA again. Lastly, `-pw` specifies the type of modification: the password.

If all has gone well, the new password `icuryy4me` is now encrypted and stored, and `masterkey` is no longer valid. Please be aware that unlike Firebird user names, passwords are case-sensitive.

Adding Firebird user accounts

Firebird allows the creation of many different user accounts. Each of them can own databases and also have various types of access to databases and database objects it doesn't own.

Using gsec, you can add a user account as follows from the command line in the Firebird bin subdirectory:

```
gsec -user sysdba -pass masterkey -add billyboy -pw sekrit66
```

Provided that you've supplied the correct password for SYSDBA, a user account called `billyboy` will now have been created with password `sekrit66`. Remember that passwords are case-sensitive.

Note: Since Firebird 2, users can change their own passwords. Previous versions required SYSDBA to do this.

Security

Firebird 2 offers a number of security options, designed to make unauthorised access as difficult as possible. Be warned however that some configurable security features default to the old, "insecure" behaviour inherited from InterBase and Firebird 1.0, in order not to break existing applications.

It pays to familiarise yourself with Firebird's security-related configuration parameters. You can significantly enhance your system's security if you raise the protection level wherever possible. This is not only a matter of setting parameters, by the way: other measures involve tuning filesystem access permissions, an intelligent user accounts policy, etc.

Below are some guidelines for protecting your Firebird server and databases.

Run Firebird as non-system user

On Unix-like systems, Firebird already runs as user firebird by default, not as `root`. On Windows server platforms, you can also run the Firebird service under a designated user account (e.g. `Firebird`). The default practice – running the service as the `LocalSystem` user – poses a security risk if your system is connected to the Internet. Consult `README.instsvc` in the `doc` subdir to learn more about this.

Change SYSDBA's password

As discussed before, if your Firebird server is reachable from the network and the system password is `masterkey`, change it.

Don't create user databases as SYSDBA

SYSDBA is a very powerful account, with full (destructive) access rights to all your Firebird databases. Its password should be known to a few trusted database administrators only. Therefore, you shouldn't use this super-account to create and populate regular databases. Instead, generate normal user accounts, and provide their account names and passwords to your users as needed. You can do this with gsec as shown above, or with any third-party Firebird administration tool.

Protect databases on the filesystem level

Anybody who has filesystem-level read access to a database file can copy it, install it on a system under his or her own control, and extract all data from it – including possibly sensitive information. Anybody who has filesystem-level write access to a database file can corrupt it or totally destroy it.

As a rule, only the Firebird server process should have access to the database files. Users don't need, and should not have, access to the files – not even read-only. They query databases via the server, and the server makes sure that users only get the allowed type of access (if at all) to any objects within the database.

Disable Classic local mode on Linux

An exception to the above rule is the so-called local or embedded access mode of Firebird Classic Server on Linux. This mode requires that users have proper access rights to the database file itself. They must also have read access to the security database `security2.fdb`. If this worries you, reserve filesystem access to the security database (and other databases, while you're at it) for the server process only. Users are then obliged to connect via the network layer. However, the `libfbembed.*` libraries should not be removed from your system, because the Firebird command-line tools refuse to run if they are not present.

(Another exception is the Windows Embedded Server, but that's outside the scope of this manual.)

Use database aliases

Database aliases shield the client from physical database locations. Using aliases, a client can e.g. connect to "`frodo:zappa`" without having to know that the real location is `frodo:/var/firebird/music/underground/mothers_of_invention.fdb`. Aliases also allow you to relocate databases while the clients keep using their existing connection strings.

Aliases are listed in the file `aliases.conf`, in this format on Windows machines:

```
poker = E:\Games\Data\PokerBase.fdb
blackjack.fdb = C:\Firebird\Databases\cardgames\blkjk_2.fdb
```

And on Linux:

```
books = /home/bookworm/database/books.fdb
zappa = /var/firebird/music/underground/mothers_of_invention.fdb
```

Giving the alias an `.fdb` (or any other) extension is fully optional. Of course if you do include it, you must also specify it when you use the alias to connect to the database.

Restrict database access

The `DatabaseAccess` parameter in `firebird.conf` can be set to `Restrict` to limit access to explicitly listed filesystem trees, or even to `None` to allow access to aliased databases only. Default is `All`, i.e. no restrictions.

Note that this is not the same thing as the filesystem-level access protection discussed earlier: when `DatabaseAccess` is anything other than `All`, the server will refuse to open any databases outside the defined scope even if it has sufficient rights on the database files.

There are more security parameters, but the ones not mentioned here are already set to an adequate protection level by default. You can read about them in the 1.5 and 2.0 *Release Notes* and in the comments in `firebird.conf` itself.

Windows Control Panel applets

Several control panel applets are available for use with Firebird. Whilst such applets are not essential, they do provide a convenient way to start and stop the server and check its current status.

Firebird Server Manager

The Firebird Server Manager applet is included in the Firebird distribution. The option to install this applet is only available for Superserver.

Note: The applet is also usable for Classic server, provided that it (the server, that is) runs as a service, not as an application. Since the installation dialogue won't give you the option to include the applet with a Classic server, you must, if you really want it:

- Install Superserver first;
- Copy the applet `Firebird2Control.cpl` from the Windows system folder to a safe place;
- Uninstall Superserver;
- Install Classic;
- Copy the applet back to the system directory.

This is a screenshot of the activated applet. Notice that the title bar says “Firebird Server Control”, although it is listed in the *Control Panel* as *Firebird 2.0 Server Manager*.

Unfortunately, the bundled applet only works on Windows NT, 2000/2003 and XP.

Firebird Control Center

If you want an applet that also works on Windows 9x or ME, visit this webpage: http://www.achim-kalwa.de/fbcc.phtml

...and download the Firebird Control Center `fbcc-0.2.7.exe`. Please note that, unlike the applet included with Firebird, the *Firebird Control Center* will not work with Classic servers at all.

The *Control Center* doesn't look anything like the Firebird applet shown in the screenshot, but offers the same functionality, and then some. *Attention:* if you run Firebird *as a service* and *without the Guardian*, the *Start/Stop* button will be labeled *Start* all the time, even when the server is already running. It functions as it should though. In all other configurations the button will say *Start* or *Stop* according to the situation.

Administration tools

The Firebird kit does not come with a GUI admin tool. It does have a set of command-line tools – executable programs which are located in the bin subdirectory of your Firebird installation. One of them, gsec, has already been introduced to you.

The range of excellent GUI tools available for use with a Windows client machine is too numerous to describe here. A few GUI tools written in Borland Kylix, for use on Linux client machines, are also in various stages of completion.

Inspect the page at for all of the options.

Remember: you can use a Windows client to access a Linux server and vice-versa.

Working with databases

In this part of the manual you will learn:

- how to connect to an existing database,
- how to create a database,
- and some things you should know about Firebird SQL.

In as much as remote connections are involved, we will use the recommended TCP/IP protocol.

Connection strings

If you want to connect to a database or create one you have to supply, amongst other things, a connection string to the client application (or, if you are a programmer, to the routines you are calling). A connection string uniquely identifies the location of the database on your computer, local network, or even the Internet.

Local connection strings

An explicit local connection string consists of the path + filename specification in the native format of the filesystem used on the server machine, for example

• on a Linux or other Unix-like server:

```
/opt/firebird/examples/empbuild/employee.fdb
```

• on a Windows server:

```
C:\Biology\Data\Primates\Apes\populations.fdb
```

Many clients also allow relative path strings (e.g. `..\examples\empbuild\employee.fdb`) but you should use them with caution, as it's not always obvious how they will be expanded. Getting an error message is annoying enough, but applying changes to another database than you thought you were connected to may be disastrous.

Instead of a file path, the local connection string may also be a database alias that is defined in `aliases.conf`, as mentioned earlier. The format of the alias depends only on how it's defined in the aliases file, not on the server filesystem. Examples are:

- `zappa`
- `blackjack.fdb`
- `poker`

Tip: If your local connections fail, it may be because the local protocol isn't working properly on your machine. If you're running Windows Vista, 2003 or XP with terminal services enabled, this can often be fixed by setting `IpcName` to `Global\FIREBIRD` in the configuration file `irebird.conf` (don't forget to uncomment the parameter and restart the server). In Firebird 2.0.1, `Global\FIREBIRD` is already the default on TS-enabled Windows systems.

If setting `IpcName` doesn't help and you don't get the local protocol enabled, you can always work around the problem by putting `localhost:` before your database paths or aliases, thus turning them into TCP/IP connection strings (discussed below).

TCP/IP connection strings

A TCP/IP connection string consists of:

1. a server name or IP address
2. a colon (":")
3. either the absolute path + filename on the server machine, or an alias defined on the server machine.

Examples

- On Linux/Unix:

pongo:/opt/firebird/examples/empbuild/employee.fdb bongo:fury 112.179.0.1:/var/Firebird/databases/butterflies.fdb localhost:blackjack.fdb

- On Windows:

  siamang:C:\Biology\Data\Primates\Apes\populations.fdb sofa:D:\Misc\Friends\Rich\Lenders.fdb 127.0.0.1:Borrowers

Notice how the aliased connection strings don't give any clue about the server OS. And they don't have to, either: you talk to a Linux Firebird server just like you talk to a Windows Firebird server. In fact, specifying an explicit database path is one of the rare occasions where you have to be aware of the difference.

Third-party programs

Please note that some third-party client programs may have different requirements for the composition of connection strings. Refer to their documentation or online help to find out.

Connecting to an existing database

A sample database named `employee.fdb` is located in the examples/empbuild subdirectory of your Firebird installation. You can use this database to "try your wings".

If you move or copy the sample database, be sure to place it on a hard disk that is physically attached to your server machine. Shares, mapped drives or (on Unix) mounted SMB (Samba) filesystems will not work. The same rule applies to any databases that you create or use.

Connecting to a Firebird database requires the user to authenticate with a user name and a valid password. In order to work with objects inside the database – such as tables, views, etc. – you also need explicit permissions on those objects, unless you own them (you own an object if you have created it) or if you're connected as SYSDBA. In the example database `employee.fdb`, sufficient permissions have been granted to `PUBLIC` (i.e. anybody who cares to connect) to enable you to view and modify data to your heart's content.

For simplicity here, we will look at authenticating as SYSDBA using the password `masterkey`. Also, to keep the lines in the examples from running off the right edge, we will work with local databases and use relative paths. Of course everything you'll learn in these sections can also be applied to remote databases, simply by supplying a full TCP/IP connection string.

Connecting with isql

Firebird ships with a text-mode client named isql (Interactive SQL utility). You can use it in several ways to connect to a database. One of them, shown below, is to start it in interactive mode. Go to the `bin` subdirectory of your Firebird installation and type `isql` (Windows) or `./isql` (Linux) at the command prompt.

[In the following examples, `#` means "hit Enter"]

```
C:\Program Files\Firebird\Firebird_2_0\bin>isql#

Use CONNECT or CREATE DATABASE to specify a database
SQL>CONNECT ..\examples\empbuild\employee.fdb user SYSDBA password masterkey;#
```

Important:

- In isql, every SQL statement must end with a semicolon. If you hit *Enter* and the line doesn't end with a semicolon, isql assumes that the statement continues on the next line and the prompt will change from `SQL>` to `CON>`. This enables you to split long statements over multiple lines. If you hit Enter after your statement and you've forgotten the semicolon, just type it after the `CON>` prompt on the next line and press *Enter* again.
- If you run Classic Server on Linux, a fast, direct local connection is attempted if the database path does not start with a hostname. This may fail if your Linux login doesn't have sufficient access rights to the database file. In that case, connect to `localhost:<path>`. Then the server process (with Firebird 2 usually running as user firebird) will open the file. On the other hand, network-style connections may fail if a user created the database in Classic local mode and the server doesn't have enough access rights.

Note: You can optionally enclose the path, the user name and/or the password in single (`'`) or double (`"`) quotes. If the path contains spaces, quoting is mandatory. At this point, isql will inform you that you are connected:

```
Database: ..\examples\empbuild\employee.fdb, User: sysdba
SQL>
```

You can now continue to play about with the `employee.fdb` database. With isql you can query data, get information about the metadata, create database objects, run data definition scripts and much more. To get back to the command prompt, type:

```
SQL>QUIT;#
```

You can also type `EXIT` instead of `QUIT`, the difference being that `EXIT` will first commit any open transactions, making your modifications permanent.

Connecting with a GUI client

GUI client tools usually take charge of composing the `CONNECT` string for you, using server, path (or alias), user name and password information that you type into prompting fields. Use the elements as described in the preceding topic.

Notes:

- It is quite common for such tools to expect the entire server + path/alias as a single connection string – just like isql does.
- Remember that file names and commands on Linux and other "Unix-ish" platforms are case-sensitive.

Creating a database using isql

There is more than one way to create a database with isql. Here, we will look at one simple way to create a database interactively –although, for your serious database definition work, you should create and maintain your metadata objects using data definition scripts.

Starting isql

To create a database interactively using the isql command shell, get to a command prompt in Firebird's `bin` subdirectory and type `isql` (Windows) or `./isql` (Linux):

```
C:\Program Files\Firebird\Firebird_2_0\bin>isql#
```

Use `CONNECT` or `CREATE DATABASE` to specify a database.

The CREATE DATABASE statement

Now you can create your new database interactively. Let's suppose that you want to create a database named `test.fdb` and store it in a directory named data on your D drive:

```
SQL>CREATE DATABASE 'D:\data\test.fdb' page_size 8192#
CON>user 'SYSDBA' password 'masterkey';#
```

Important:

- In the `CREATE DATABASE` statement it is mandatory to place quote characters (single or double) around `path`, `username` and `password`. This is different from the `CONNECT` statement.
- If you run Classic Server on Linux and you don't start the database path with a hostname, creation of the database file is attempted with your Linux login as the owner. This may or may not be what you want (think of access rights if you want others to be able to connect). If you prepend `localhost:` to the path, the server process (with Firebird 2 usually running as user firebird) will create and own the file.

The database will be created and, after a few moments, the SQL prompt will reappear. You are now connected to the new database and can proceed to create some test objects in it.

But to verify that there really is a database there, let's first type in this query:

```
SQL>SELECT * FROM RDB$RELATIONS;#
```

Although you haven't created any tables yet, the screen will fill up with a large amount of data! This query selects all of the rows in the system table `RDB$RELATIONS`, where Firebird stores the metadata for tables. An "empty" database is not really empty: it contains a number of system tables and other objects.

The system tables will grow as you add more user objects to your database.

To get back to the command prompt type `QUIT` or `EXIT`, as explained in the section on connecting.

Firebird SQL

Every database management system has its own idiosyncrasies in the ways it implements SQL. Firebird adheres to the SQL standard more rigorously than most other RDBMSes. Developers migrating from products that are less standards-compliant often wrongly suppose that Firebird is quirky, whereas many of its apparent quirks are not quirky at all.

Division of an integer by an integer

Firebird accords with the SQL standard by truncating the result (quotient) of an integer/integer calculation to the next lower integer. This can have bizarre results unless you are aware of it. For example, this calculation is correct in SQL:

```
1 / 3 = 0
```

If you are upgrading from an RDBMS which resolves integer/integer division to a float quotient, you will need to alter any affected expressions to use a float or scaled numeric type for either dividend, divisor, or both. For example, the calculation above could be modified thus in order to produce a non-zero result:

```
1.000 / 3 = 0.333
```

Things to know about strings

String delimiter symbol

Strings in Firebird are delimited by a pair of single quote (apostrophe) symbols: `'I am a string'` (ASCII code 39, not 96). If you used earlier versions of Firebird's relative, InterBase®, you might recall that double and single quotes were interchangeable as string delimiters. Double quotes cannot be used as string delimiters in Firebird SQL statements.

Apostrophes in strings

If you need to use an apostrophe inside a Firebird string, you can "escape" the apostrophe character by preceding it with another apostrophe. For example, this string will give an error:

```
'Joe´s Emporium'
```

because the parser encounters the apostrophe and interprets the string as 'Joe' followed by some unknown keywords. To make it a legal string, double the apostrophe character:

```
'Joes´´ Emporium'
```

Notice that this is TWO single quotes, not one double-quote.

Concatenation of strings

The concatenation symbol in SQL is two "pipe" symbols (ASCII 124, in a pair with no space between). In SQL, the "+" symbol is an arithmetic operator and it will cause an error if you attempt to use it for concatenating strings. The following expression prefixes a character column value with the string "`Reported by: `":

```
'Reported by: ' || LastName
```

Firebird will raise an error if the result of a string concatenation exceeds the maximum (var)char size of 32 Kb.

If only the potential result – based on variable or field size – is too long you'll get a warning, but the operation will be completed successfully. (In pre-2.0 Firebird, this too would cause an error and halt execution.)

See also the section below, Expressions involving `NULL`, about concatenating in expressions involving `NULL`.

Double-quoted identifiers

Before the SQL-92 standard, it was not legal to have object names (identifiers) in a database that duplicated keywords in the language, were case-sensitive or contained spaces. SQL-92 introduced a single new standard to make any of them legal, provided that the identifiers were defined within pairs of double-quote symbols (ASCII 34) and were always referred to using double-quote delimiters.

The purpose of this "gift" was to make it easier to migrate metadata from non-standard RDBMSes to standards-compliant ones. The down-side is that, if you choose to define an identifier in double quotes, its case-sensitivity and the enforced double-quoting will remain mandatory.

Firebird does permit a slight relaxation under a very limited set of conditions. If the identifier which was defined in double-quotes:

1. was defined as all upper-case,
2. is not a keyword, and
3. does not contain any spaces,

...then it can be used in SQL unquoted and case-insensitively. (But as soon as you put double-quotes around it, you must match the case again!)

Warning: Don't get too smart with this! For instance, if you have tables `"TESTTABLE"` and `"TestTable"`, both defined within double-quotes, and you issue the command:

```
SQL>select * from TestTable;
```

...you will get the records from `"TESTTABLE"`, not `"TestTable"`!

Unless you have a compelling reason to define quoted identifiers, it is usually recommended that you avoid them. Firebird happily accepts a mix of quoted and unquoted identifiers – so there is no problem including that keyword which you inherited from a legacy database, if you need to.

Warning: Some database admin tools enforce double-quoting of all identifiers by default. Try to choose a tool which makes double-quoting optional.

Expressions involving `NULL`

In SQL, `NULL` is not a value. It is a condition, or *state*, of a data item, in which its value is unknown. Because it is unknown, `NULL` cannot behave like a value. When you try to perform arithmetic on `NULL`, or involve it with values in other expressions, the result of the operation will almost always be `NULL`. It is not zero or blank or an "empty string" and it does not behave like any of these values.

Below are some examples of the types of surprises you will get if you try to perform calculations and comparisons with `NULL`.

The following expressions all return `NULL`:

- `1 + 2 + 3 + NULL`
- `not (NULL)`
- `'Home ' || 'sweet ' || NULL`

You might have expected 6 from the first expression and "Home sweet " from the third, but as we just said, `NULL` is not like the number 0 or an empty string – it's far more destructive!

The following expression:

- `FirstName || ' ' || LastName`

will return `NULL` if either `FirstName` or `LastName` is `NULL`. Otherwise it will nicely concatenate the two names with a space in between – even if any one of the variables is an empty string.

Tip: Think of `NULL` as *UNKNOWN* and these strange results suddenly start to make sense! If the value of `Number` is unknown, the outcome of `'1 + 2 + 3 + Number'` is also unknown (and therefore `NULL`). If the content of `MyString` is unknown, then so is `'MyString || YourString'` (even if `YourString` is non-`NULL`). Etcetera.

Now let's examine some PSQL (Procedural SQL) examples with if-constructs:

- `if (a = b) then`

```
MyVariable = 'Equal';
else
MyVariable = 'Not equal';
```

After executing this code, `MyVariable` will be `'Not equal'` if both `a` and `b` are `NULL`. The reason is that `'a = b'` yields `NULL` if at least one of them is `NULL`. If the test expression of an "`if`" statement is `NULL`, it behaves like false: the `'then'` block is skipped, and the `'else'` block executed.

Warning: Although the expression may behave like false in this case, it's still `NULL`. If you try to invert it using `not()`, what you get is another `NULL` – not "`true`".

- `if (a <> b) then`

```
MyVariable = 'Not equal';
else
MyVariable = 'Equal';
```

Here, `MyVariable` will be `'Equal'` if `a` is `NULL` and `b` isn't, or vice versa. The explanation is analogous to that of the previous example.

The `DISTINCT` keyword comes to the rescue!

Firebird 2 implements a new use of the `DISTINCT` keyword allowing you to perform (in)equality tests that take `NULL` into account. The semantics are as follows:

- Two expressions are `DISTINCT` if they have different values or if one is `NULL` and the other isn't;
- They are `NOT DISTINCT` if they have the same value or if both are `NULL`.

Notice that if neither operand is `NULL`, `DISTINCT` works exactly like the "`<>`" operator, and `NOT DISTINCT` like the "`=`" operator.

`DISTINCT` and `NOT DISTINCT` always return `true` or `false`, never `NULL`.

Using `DISTINCT`, you can rewrite the first PSQL example as follows:

```
if (a is not distinct from b) then
MyVariable = 'Equal';
else
MyVariable = 'Not equal';
```

And the second as:

```
if (a is distinct from b) then
MyVariable = 'Not equal';
else
MyVariable = 'Equal';
```

These versions will give you the results that a normal human being (untouched by SQL standards) would expect, whether there are `NULL`s involved or not.

More about `NULL`s

A lot more information about `NULL` behaviour can be found in the *Firebird Null Guide*, at these locations:

http://www.firebirdsql.org/manual/nullguide.html (HTML)
http://www.firebirdsql.org/pdfmanual/Firebird-Null-Guide.pdf (PDF)

An updated and greatly extended version of the *Null Guide* is available since January 2007.

See also:
Firebird 2 SQL Reference Guide

Preventing data loss

Backup

Firebird comes with two utilities for backing up and restoring your databases: `gbak` and `nbackup`. Both can be found in the `bin` subdirectory of your Firebird installation. Firebird databases can be backed up whilst users are connected to the system and going about their normal work. The backup will be taken from a snapshot of the database at the time the backup began.

Regular backups and occasional restores should be a scheduled part of your database management activity.

Warning

Except in `nbackup`'s lock mode, do not use external proprietary backup utilities or file-copying tools such as *WinZip*, *tar*, *copy*, *xcopy*, etc., on a database which is running. Not only will the backup be unreliable, but the disk-level blocking used by these tools can corrupt a running database.

Important

Study the warnings in the next section about database activity during restores!

More information about `gbak` can be found in *The Firebird Book*, the *Using Firebird* guide (a not-so-recent version is available through IBPhoenix, an updated version is currently in a state of growth on the Firebird site), or in the InterBase 6.0 manuals combined with the Firebird 1.5 and 2.0 *Release Notes*. See the links to these resources in How to get help.

The `nbackup` manual is here (HTML and PDF version, same content):

http://www.firebirdsql.org/manual/nbackup.html
http://www.firebirdsql.org/pdfmanual/Firebird-nbackup.pdf

How to corrupt a database

The following sections constitute a summary of things not to do if you want to keep your Firebird databases in good health.

Modifying metadata tables yourself

Firebird stores and maintains all of the metadata for its own and your user-defined objects in special tables, called system tables, right in the database itself. The identifiers for these system tables, their columns and several other types of system objects begin with the characters RDB$.

Because these are ordinary database objects, they can be queried and manipulated just like your user-defined objects. However, just because you can does not say you should. The Firebird engine implements a high-level subset of SQL (DDL) for the purpose of defining and operating on metadata objects, typically through `CREATE`, `ALTER` and `DROP` statements.

It cannot be recommended too strongly that you use DDL – not direct SQL operations on the system tables - whenever you need to alter or remove metadata. Defer the "hot fix" stuff until your skills in SQL and your knowledge of the Firebird engine become very advanced. A wrecked database is neither pretty to behold nor cheap to repair.

Disabling forced writes

Firebird is installed with forced writes (synchronous writes) enabled by default. Changed and new data are written to disk immediately upon posting.

It is possible to configure a database to use asynchronous data writes – whereby modified or new data are held in the memory cache for periodic flushing to disk by the operating system's I/O subsystem. The common term for this configuration is `forced writes off` (or `disabled`). It is sometimes resorted to in order to improve performance during large batch operations.

Disabling forced writes on Windows

The big warning here is: do not disable forced writes on a Windows server. It has been observed that the Windows server platforms do not flush the write cache until the Firebird service is shut down. Apart from power interruptions, there is just too much that can go wrong on a Windows server. If it should hang, the I/O system goes out of reach and your users' work will be lost in the process of rebooting.

Note

Windows 9x and ME do not support deferred data writes.

Disabling forced writes on Linux

Linux servers are safer for running an operation with forced writes disabled temporarily. Still, do not leave it disabled once your large batch task is completed, unless you have a very robust fall-back power system.

Warning

It was recently discovered that forced writes did not work at all under Linux. This is due to a bug in the `fcntl()` function on Linux and it affects all Firebird versions up to and including 2.0.3. The only known workaround is to mount the partition in question with the sync option — or upgrade to Firebird 2.0.4 or higher.

Other Unices don't seem to suffer from this bug. To make sure, test if you system's `fcntl()` can successfully set the `O_SYNC` flag. Set the flag on and off and read it back both times to make sure the change was actually written.

Restoring a backup to a running database

One of the restore options in the `gbak` utility (`gbak -rep[lace_database]`) allows you to restore a `gbak` file over the top of an existing database. It is possible for this style of restore to proceed without warning while users are logged in to the database. Database corruption is almost certain to be the result.

Note

Notice that the shortest form of this command is `gbak -rep`, not `gbak -r` as it used to be in previous Firebird versions.

What happened to `gbak -r`? It is now short for `gbak -recreate_database`, which functions the same as `gbak -c[reate]` and throws an error if the specified database already exists. You can force overwriting of the existing database by adding the `o[verwrite]` flag though. This flag is only supported with `gbak -r`, not with `gbak -c`.

These changes have been made because many users thought that the `-r` switch meant `restore` instead of `replace` – and only found out otherwise when it was too late.

Warning

Be aware that you will need to design your admin tools and procedures to prevent any possibility for any user (including SYSDBA) to restore to your active database if any users are logged in.

If is practicable to do so, it is recommended to restore to spare disk space using the `gbak -c[reate]` option and test the restored database using isql or your preferred admin tool. If the restored database is good, shut down the server. Make a filesystem copy of the old database and then copy the restored database file (or files) over their existing counterparts.

Allowing users to log in during a restore

If you do not block access to users while performing a restore using `gbak -rep[lace_database]` then users may be able to log in and attempt to do operations on data. Corrupted structures will result.

See also:
Database Corruption
Firebird for the database expert: Episode 3 - On Disk Consistency
Alternative database repair methods

How to get help

The community of willing helpers around Firebird goes a long way back, to many years before the source code for its ancestor, InterBase® 6, was made open source. Collectively, the Firebird community does have all the answers! It even includes some people who have been involved with it since it was a design on a drawing board in a bathroom in Boston.

- Visit the official Firebird Project site at http://www.firebirdsql.org and join the user support lists, in particular firebird-support. Look at http://www.firebirdsql.org/?op=lists for instructions.
- Use the Firebird documentation index at http://www.firebirdsql.org/?op=doc.
- Visit the Firebird knowledge site at http://www.ibphoenix.com to look up a vast collection of information about developing with and using Firebird. IBPhoenix also sells a Developer CD with the Firebird binaries and lots of documentation.
- Order the official *Firebird Book* at http://www.ibphoenix.com/main.nfs?a=ibphoenix&s=1093098777:149734&page=ibp_firebird_book, for more than 1100 pages jam-packed with Firebird information.
- As a last resort – since our documentation is still incomplete – you can consult the InterBase 6.0 beta manuals (the files whose names start with 60 at http://www.ibphoenix.com/downloads/) in combination with the Firebird 1.5 and 2.0 *Release Notes*.

Note

The IBPhoenix publications *Using Firebird* and *The Firebird Reference Guide*, though still on the Developer CD, are no longer actively maintained. However, most of the material contained in those documents is currently being brought up to date and added, bit by bit, to the official project documentation.

The Firebird Project

The developers, designers and testers who gave you Firebird and several of the drivers are members of the Firebird open source project at SourceForge, that amazing virtual community that is home to thousands of open source software teams. The Firebird project's address there is http://sourceforge.net/projects/firebird. At that site are the source code tree, the download packages and a number of technical files related to the development and testing of the codebases.

The Firebird Project developers and testers use an email list forum – firebird-devel@lists.sourceforge.net – as their "virtual laboratory" for communicating with one another about their work on enhancements, bug-fixing and producing new versions of Firebird.

Anyone who is interested in watching their progress can join this forum. However, user support questions are a distraction which they do not welcome. Please do not try to post your user support questions there! These belong in the firebird-support group.

Happy Firebirding!

Document History

The exact file history is recorded in the manual module in our CVS tree; see http://sourceforge.net/cvs/?group_id=9028

Revision History

0.0	2002	IBP	Published as Chapter One of *Using Firebird*.
1.0	2003	IBP	Published separately as a free *Quick Start Guide*.
1.x	June 2004	IBP	Donated to Firebird Project by IBPhoenix.
2.0	27 Aug 2004	PV	Upgraded to Firebird 1.5 Added Classic vs. Superserver section. Reorganised and corrected *Disk Locations Table*. Added (new) screenshots. Added section on security. Updated and completed information on Control Panel applets. Added more examples to Expressions involving `NULL`. Various other corrections and additions.
2.1	20 Feb 2005	PV	Enhanced GSEC section. Added more info to `CONNECT` and `CREATE DATABASE` sections. Added version number and document history.
2.1.1	1 Mar 2005	PV	Changed `gbak r[estore]` to `r[eplace]` in two places.
2.1.2	8 Apr 2005	PV	Reordered Firebird SQL subsections. Added links to *Firebird Null Guide*.
2.2 2	Dec 2005	PV	Removed "Using the books by IBPhoenix" as it doesn't make sense in the QSG. Promoted How to get help to 1st-level section and removed *Where to next* shell. Removed link to *UFB* and *RefGuide*; added a note instead explaining their current status. Updated/corrected classic-super comparison table. Moved a number of sections on installing, working with databases, and (un)safety into newly created top-level sections.
2.2.1	22 Dec 2005	PV	Corrected statement on SS thread usage in Classic-vs-Superserver table. Fixed broken link.
3.0	21 May 2006	PV	Creation of Firebird 2 Quick Start Guide, still equal to previous revision except for some version numbers, XML ids etc.
3.2	10 Aug 2006	PV	Promoted "Firebird Project members" to co-authors in articleinfo. Updated references to website (`firebird.sourceforge.net` -> http://www.firebirdsql.org). Removed "maturity" and "Service Manager" rows from Classic-vs-Super table; these things are no longer different in Firebird 2. Also changed the row on local connections: CS and SS now both allow safe, reliable local connections on Windows. Added row on Guardian. Prepended a column with feature names. Removed any and all remarks about Classic not having a (full) Service Manager. Removed 2nd paragraph of Default disk locations section. Removed notes stating that Classic/Win connections will fail without a host name. Updated location table and inserted rows for documentation. Edited the *Installation* sections; added sections on Guardian and installing multiple servers. Removed "if-you-do-not-find-the-release-notes" tip. Heavily edited and extended the Testing your installation sections. The *Other things you need* section is now gone and its contents distributed across other sections. Added a section on `gsec` (consisting partly of existing material). Greatly enhanced and extended the Security section, and moved it to another location. Extended and improved the Windows Control Panel applets section. Edited Working with databases. Added a special section on connection strings. Added information on access to database objects, the `EXIT` statement, and local vs. remote connections. Made some paths in the examples relative, to keep the lines short. Extended paragraph on metadata. Weakened the claim that Firebird is more SQL-compliant than any other RDBMS. Changed the Expressions involving NULL section. Added a subsection on `DISTINCT`. Changed More about `NULLS` subsection somewhat. Renamed "Safet *measures* to Preventing data loss. The Security subsection has been moved elsewhere. Extended Backup section to include `nbackup` information. Added links to other documentation. In the How to corrupt... part, changed `gbak -r` syntax to `-rep` and added explanatory note. Added the *IB6 plus rlsnotes* as last-resort option to How to get help. Also mentioned firebird support explicitly. Corrected more version numbers, paths, and stuff. Many sections have been reshuffled, moved up or down the hierarchy, etc. Many smaller modifications are not listed here. Added "Happy Firebirding!" to conclude the last section.
3.3	15 Oct	PV	Default disk locations table: added isql to command line tools; added row for additional server-side libs. Added introductory paragraph to Installing Firebird. Changed first sentence of "Installing on Linux... Changed and extended "Server check: Linux and other Unices". Corrected and extended the section on Linux client-only installs. Security section: moved last paragraph of the “Protect databases...” list item into a new item on Classic local mode. Connection strings: improved and extended introductory paragraph; added a subsection on third party program requirements. Changed 3rd and 4th paragraph of Connecting to an existing database. Used relative paths in connection examples. Updated/corrected note on the use of quote characters.

			Edited first “Important” item in The CREATE DATABASE statement. Updated the warning about concatenation of long strings. Extended the note in Restoring a backup to a running database. Updated last sentence of first paragraph in The Firebird Project.
3.4	25 Jan 2007	PV	About this guide: Changed note about versions and replaced HTML and PDF links with single link to new doc index page. Classic or Superserver?: Replaced note on Embedded Server with a proper subsection, containing more info and links to UFB. Default disk locations: Created two subsections (for Linux and Windows); also split table in two and removed first column. Introduced placeholders `<ProgramDir>` and `<SystemDir>`. Changed text around tables, changed existing note, and added note for Win64 users. Security: Removed statement that 1.5 Release Notes are included with 2.x packages. More about `NULLS`: Replaced note about the *Null Guide* being updated with a para announcing the availability of the new version. Backup: Updated information on UFB. How to get help: Updated documentation links and changed text here and there.
3.5	14 Mar 2007	PV	About this guide and *Important notice for 64-bit Windows users*: Minor rewordings. User management: `gsec` and Connection strings: Added information on enabling local protocol with `IpcName=Global\FIREBIRD`. Security:: *Use database aliases*: Changed type from `<database>` to `<literal>` to improve output.
3.6	21 Sep 2007	PV	About this guide: Mentioned 2.0.3. Warned against 2.0.2. Expressions involving `NULL`: Space added to expected concatenation result: "Home sweet ".
3.7	8 Apr 2008	PV	About this guide: Added 2.0.4 and 2.1 to covered versions. Mentioned forced writes bug. Installing the Firebird server :: *Use the Guardian?*: Added warning about Win installer not detecting existing server. How to corrupt a database: Gave subsections id attributes. Disabling forced writes on Windows: Created new parent section Disabling forced writes, with the Windows and Linux cases as subsections. Warned against Linux forced writes bug. License notice : Copyright end year now 2008.

License Notice

Firebird Development using IBExpert

This documentation introduces developers to Firebird development, with the emphasis on IBExpert as an aid to make your life easier. Even the more experienced Firebird developers will find a wealth of tips here.

Source: Firebird School at the Firebird Conference 2007 held in Hamburg, Germany

SQL basics

If you are really new to SQL, first check the definitions for Structured Query Language, and DSQL, ESQL, isql and PSQL. You can find a reference of the most important commands in the *SQL Language Reference*, and the full range of Firebird 2.0 commands in the *Firebird 2 SQL Reference Guide*. However you will find that the following are the most commonly used commands, with which you will be able to do the majority of your work:

SELECT INSERT UPDATE DELETE	These commands are known collectively as DML (Data Manipulation Language) commands. They are a collection of SQL commands, commonly known as SIUD, which can be used to manipulate a database's data. SIUD is the abbreviation for SELECT, INSERT, UPATE, DELETE.
CREATE ALTER DROP EXECUTE SET	These commands belong to the Data Definition Language (DDL) set of commands, which define and manipulate the database and its structure (known as metadata). A full explanation of these commands can be found in the *DDL - Data Definition Language* chapter.

Setting up a sample database

In order to gain follow the examples in this section and to offer the chance to play around with Firebird SQLs, we propose you install the demo database, `db1.fdb` supplied with IBExpert. Installation details can be found in the *IBExpertDemoDB* documentation.

Alternatively, Firebird also supplies a sample database, `employee.fdb`. However as this is the orginal sample database provided by InterBase in the 1990's it's potential for testing is unfortunately somewhat limited.

Simple SELECT commands

The most basic SELECT command is:

```
select * from <table_name>
```

where `*` is a so-called wildcard. Let's take an example using our demo database, and enter the query? in the IBExpert SQL Editor on the *Edit* page. If we want a list of all information in the `product` table:

```
select * from product
```

You will notice how IBExpert aids you when typing your database object name. When you enter `PR` the IBExpert Code Completion offers you a selection of all objects beginning with `PR`. When the key combination [Alt + Ctrl + T] is used, IBExpert offers a list of all tables beginning with `PR`.

If you've entered the object name correctly, for example the `product` table, IBExpert changes the text format (font color and underlined) if it recognizes the object, so you know immediately whether you have made a typing error (no change to text appearance) or not.

To run the query (`EXECUTE`) simply press the [F9] key or the green arrow icon:

The SQL Editor displays all resulting data sets found that meet the conditions of the query (in this case all fields of all data sets in the `product` table):

Please note that in IBExpert you can define whether you wish the results to appear on the same page as your query (i.e. below the editing area) or on a separate page, and whether IBExpert should immediately display this *Results* page after the query has been executed. Please refer to Environment Options / Tools / SQL Editor for further information.

Below the results you can see a summary of how Firebird attained the information.

If you wish to make your query more selective, you can specify which specific information you wish to see, instead of all of it. For example, the DVD title and leading actor of all products:

```
select title, actor from product
```

When you're writing a `select` it can become very tiresome repeatedly writing out the full names of commonly used objects correctly. It's helpful to abbreviate such objects, also reducing the amount of frequent typing errors. This is possible by defining a so-called alias. For example, if you wish to define an alias for the `product` table, type `select from product p`. That way the server knows that whenever you type a `p` in this SQL, you are referring to the `product` table.

IBExpert also recognizes the `p` as an alias and automatically offers me a list of all fields in the `product` table. By holding down the [Ctrl] key multiple fields can be selected, e.g. `title` and `actor`. By pressing the [Enter] key both fields are automatically inserted into the SQL with the alias prefix `p`.

Adding a `WHERE` clause

It is possible to set conditions on the information you want to see by adding a `WHERE` clause. For example:

```
select * from product p where p.category_id = 1
```

And if you only wish to see certain columns in the result sets:

```
select p.title, p.price, p.category from product p
 where p.category_id = 1
```

`SELECT`s can of course get a lot more complicated than this! It's important to try and keep it as simple as possible though. Because it's a mathematical notation, a complex SQL may look correct, but if you are not careful, you will get results that you did not really want. When you're working with many millions of data sets and you can't necessarily assess the values in the resulting statistical data, it's vital you're sure there are no mistakes or logical errors in your query. Build your statements up gradually, checking each stage - this is easy in the IBExpert SQL Editor, as you can execute query parts by simply marking the segment you wish to test and executing. Only if no query areas are selected by marking them, does the SQL Editor execute the whole statement.

It is of course possible to specify more than one condition, e.g.:

```
select * from product where special=1 and category_id=2
```

`CONTAINING`

```
select * from product where title containing 'HALLOWEEN'
```

This will supply all films with the word `HALLOWEEN` somewhere in the title. `CONTAINING` is case-insensitive, and never uses an index, as it searches for a string contained somewhere in the field, not necessarily at the beginning.

`ORDER BY`

If you need your results in a certain format, you can specify that the results be ordered, alphabetically or numerically, by a certain field. For example, order by price in ascending order (lowest first, highest last):

```
select * from product order by price
```

The ascending order is the so-called default; that means it is not necessary to specify it specifically. However, if you wish to specify a descending order, this needs to be explicitly specified:

```
select * from product order by price desc
```

SELECT across multiple tables

To combine data across multiple tables you can `JOIN` the tables together, giving you results that contains information from both. For example, each film is categorized according to genre.

[Table Editor/Data)

Now what we want to see is the category that these filns are associated with:

```
select p.title, c.txt
from product p
join category c on c.id=p.category_id
```

The `JOIN` is a flexible command. The above example is known as an `INNER JOIN`.

Theoretically there could be products that have not been categorized, or categories that have no products. If you want to include these products or these categories in your result list it is possible to define these using a so-called `LEFT OUTER JOIN` or a `RIGHT OUTER JOIN`.

The `LEFT OUTER JOIN` takes all information from the left-hand or first table (in our example `product`) and joins them to their categories. For example if you have a customer list with individual sales figures and you also want to see those cusotmers without any sales.

The `RIGHT OUTER JOIN` fetches all products with a category and also all categories.

If you wish to combine two different sets of data together, even if they have nothing in common, you can use the `CROSS JOIN`, introduced in Firebird 2.0:

```
select p.title, c.txt
from product p
cross join category c
```

From these simple building blocks you can construct very complex structures with extremely complex results. If you are just beginning with SQL, we recommend the IBExpert Query Builder. This enables you to compile your SQL by simply dragging and dropping your objects, and using point-and-click to specify which information you wish to see, set any conditions and sort the results.

Please refer to the IBExpert Tools menu item, Query Builder for further information.

Sub-SELECTs in fields and WHERE clauses

We can vary our query by replacing the second field by a sub-select:

```
select p.title,
  (select c.txt from category c
   where c.id=p.category_id)category_txt
from product
```

By replacing `c.txt` with `where c.id=p.category_id) category_txt` the `JOIN` is no longer necessary. This new second field is determined for each data set. As the sub-select is creating a new unnamed field, the field is given an alias, `category_txt`. You can name result columns as you like, particularly useful when columns with similar names from different tables are to be queried. For example, if you wish to see `c.id` and `p.id` in the same result set, you might want to rename `c.id category_id` and `p.id product_id`.

Physically this query is the same as the `JOIN` query, however this option offers more possibilities.

You can also insert a sub-select in a `WHERE` clause: select which fields you want from which tables and restrict it by adding a sub-select in the `WHERE` condition. For example, if you only want to see products from the first category:

```
select p.title, c.txt
  from product p
  join category c on c.id=p.category_id
  where c.id=(select first 1 id from category)
```

Be careful with this, as this is one of the areas of SQL where a lot of developers start to go wrong!

UNION SELECT

`SELECT`s are great and you can retrieve almost any information you want with a single `SELECT` statement. A classic example of when you might need a `UNION SELECT` is with a database system that stores its current data in one table and archive data in another table, and a report is required which includes both sets of data being evaluated and presented as a single set of information.

The syntax is simple: two `SELECT` statements with a `UNION` in between to fuse them together:

```
Select
  p.title,
  cast('Children' as varchar(20))
  from product p
  join category c on c.id=p.category_id
  where c.txt containing 'children'
union
Select
  p.title,
  cast('not for Children' as varchar(20))
  from product p
  join category c on c.id=p.category_id
  where c.txt not containing 'children'
```

Here all titles are being selected that belong to the category `children`. These results are then going to be combined with another set where the category does not contain the text `children@, and all these results (i.e. every other category that isn't explicitly for children) will contain the category text` not for Children@@, regardless of their genre. This artificial field supplies information that is not directly in the database in that form.

The rules regarding the joining together of two result sets is that you have to have columns with the same datatypes, i.e. you cannot mix `INTEGERS` and blobs in a single result column. You must have the same number of columns in the same layout, e.g. if you current orders table has 50 columns and the archive only 30 columns, you can only select common columns for the `UNION SELECT`.

IN operator

```
Select p.title,c.txt
  from product p
  join category c on c.id=p.category_id
  where c.id in (select first 5 id from category)þ
```

Here the value c.id is being limited to the first five, i.e. we only wish to see the first five resulting sets.

The `IN` operator is very powerful. Assume you wish to view film categories, `Action`, `Animation` and a couple of others and you had already retrieved the result that these categories were 1, 2, 5 and 7. Then you could query as follows:

```
Select p.title,c.txt
  from product p
```

```
  join category c on c.id=p.category_id
  where p.category_id in (1,2,5,7)
```

i.e. here it is asking for results where the `category_id` is in the specified set of values. The `IN` can be a set of values or a `SELECT`. You should be careful that there are not too many results, as this can slow performance considerably.

EXISTS operator

```
select c.* from customer c
 where not exists (select id from orders where orders.customer_id=c.id)
```

Here we are selecting the customers from the customer table where if one or more rows are returned then it will give you the value. If no values are returned then it omits it und does not show it. This means, these results will only return customers who have not placed any orders.

The `EXISTS` operator is almost always more helpful than the `IN` operator. The `EXISTS` operator searches if data sets meeting the conditions exist and when it finds results sends them back. The `IN` operator would initially fetch all data sets, i.e. fetch all orders, and then narrow down the result sets according to the conditions.

If you have a choice between `IN` and `EXISTS`, always use `EXISTS` as it's quicker.

INSERT and UPDATE with values

```
insert into category values (20, 'Cartoons')
```

`INSERT` - As no columns have been named here the values `20` and `Cartoons` are inserted from left to right in the `category` table columns. If the column names are not specified, data has to be inserted into all columns (the `category` table only has two columns). For larger tables it is wise to be more specific and always name the columns you wish to insert data into, as you may not wish to insert into all columns.

```
insert into category (id,txt) values (21, 'More cartoons')
```

Always take into consideration that `NOT NULL` fields have to be filled.

`UPDATE` applies to the whole table. It is simply a list of z variables or fields and their new values, with a condition.

Update product

```
  set title='FIREBIRD CONFERENCE DAY',
  Actor='FIREBIRD FOUNDATION'
  where id=1;
```

If you don't put a qualifying clause in there about what it's going to do, so if you don't have a `WHERE` clause, it will update everything! So always check thoroughly before committing!

Unlike `SELECT`, both these commands only interact with one table at a time.

You can also use `INSERT INTO` with `SELECT`ed data:

```
insert into customer_without_orders
select c.* from customer c
  where not exists (select id from orders where orders.customer_id=c.id)
```

This can be used to insert data into a table that's been supplied from another source (here the `select from customer`).

Whereas Firebird requires the table in which you want to insert data to already exist, the IBExpert SQL Editor however has a nice feature: it will create the table for you if it does not already exist! In the above example, if the `customers_without_orders` table does not already exist, IBExpert asks if it should create the table. If you agree, it creates a table according to the information supplied in the query and pushes the returns in to the new table `customer_without_orders`. This function is ideal if you wish to extract certain data for testing or for a temporary report.

DELETE

```
delete from orderlines
  where id<1000
```

This will delete all data sets with an `id` of less than 1000.

```
delete from orderlines
  where id between 1000 and 2000
```

This will delete all data sets with `id` between 1000 and 2000.

Be careful when defining your delete conditions. A mistake here and you will delete the wrong data sets or too many!

CREATE, ALTER and DROP

If you're just starting off, we would not recommend creating all database objects by writing SQL. Use IBExpert's DB Explorer to create and manipulate all your databases and database objects. Please refer to the IBExpert chapters: *DB Explorer* and *Database Objects*.

To understand how the database structure works, analyze the DDL code created by IBExpert as a result of your point and click actions. This can be found on the DDL page in all object editors.

See also:
`Select`
DDL-DataDefinitionLanguage
DML - Data Manipulation Language
Database Objects

Creating your first database

Developing a data model

A data model includes everything that is going to sit inside the database. If you are new to database development, it's worth taking a little time and effort to read up on the theory of database design. We recommend the database technology article; *Database design and database normalization* as a basic introduction to database model development.

Before you start you need to make a few rules and stick to them. For example, primary keys should always be a simple `BIGINT` internal generator ID, not influenced in any way by any actual data. Many developers use unique information fields as primary keys, such as a social security number or membership number. But what if the social security number system changes or the membership card is stolen and a new membership with the same member details needs to be created and the old made invalid? You are bound to encounter problems if you rely on such information for your primary key. And compound primary keys (primary keys consisting of more than one field) will almost always lead to problems at some stage as the sequence of the fields concerned must be identical in all referenced tables, and compound keys will always slow performance.

Another consideration is how to structure your data. This is where basic information about database normalization comes in. If you store your customer address data in your `customer` table and your supplier address data in your `supplier table`, you may end up with double entries (a supplier can also be a customer, a single customer may have more that one address). So create an `address` table with relationships to the `customer` and `supplier` tables. Using views the end user sees his customer, customer number and address or supplier, supplier number and his address.

Always start at the highest level, make sure you have got your entities correct. Construct your main tables and relationships. More information about the various kind of data relationships can be refered to below (*Relationships*). Don't get bogged down by the details at this intial stage; attributes can be added at a later stage. Scope it first - how big is it going to be? How's is it all going to fit together?

And when you do get down to the details, don't start using your fantasy or trying to look too far into the future. Only store information that is real and existent.

Naming conventions

You need to develop a naming convention that enables you and others to find and identify keys, table fields, procedures, triggers etc. simply and quickly, using a simple but effective combination of table names, field names, keys and relationships.

Please name things simply and logically: call a spade a spade, not an "manual excavation device" or "portable digging implement"! Another decision to be made is whether to name things in the singular or plural. If you have a team developing the same database, you are bound to have conflicts here and maybe even duplicates (e.g. `CUSTOMER` and `CUSTOMERS`), if you don't make a decision before you start! As the singular form is shorter than the plural in most languages, this is recommended, i.e. `CUSTOMER` instead of `CUSTOMERS`, `ORDERLINE` instead of `ORDERLINES` etc. Please note that in the `db1` database, `ORDER` had to be named `ORDERS`, because `ORDER` is a Firebird keyword. The table could still be named `ORDER` but would have to be defined in inverted commas, which could lead to other problems. So English-language developers need to be aware of Firebird keywords and avoid eventual conflicts.

Another tip is to avoid using `$` in your database object names, as `$` is always used in system object names. All Firebird and InterBase system objects begin with `RDB$` and IBExpert system objects begin with `IBE$`.

Primary keys are easily recognizable if the field name has the prefix `PK` (alternatively: `ID`) followed by a reference to the table name. Foreign keys should logically then contain the prefix `FK` followed by the table name which they reference.

Relationships

You need to be able to uniquely identify each row in each table, so each table requires a primary key. Other tables referencing this should be given a foreign key.

In our sample database, `db1`, each product is assigned to a category. The `category_id` links the `product` table to the `category` table, alternatively `FK_category` would also be a suitable name for the column referencing the relationship to the `category` table. In fact, if a relationship exists between two tables, put it in - make sure the database knows about it. It will help you in the long run, and in this way you can improve integrity, for example, you can enforce every product to be assigned to a category. Please refer to the *Keys* chapter for a comprehensive guide to Firebird/InterBase keys. Further information regarding constraints generally can be found in the *Constraints* chapter.

There are various kinds of relationships between data, which need to be taken into consideration when defining the constraints:

1:1

Within your application you have relationships which are 1:1. Many people say that if you have a 1:1 relationship between two tables, then it should be put together and become one table. However this is not alway the case, particularly when developing one application for different clients with different

requirements. There are often good reasons for maintaining a core `customer` table that is distributed to all customers, and then a `customer_x` table that includes information for a specific client. It prevents tables becoming too wide and confusing.

Another reason for 1:1 tables may be that in the case of wide tables with huge amounts of data, searching for specific information just takes too long. For example most journalists search in a press agency database using keywords for anything relevant to a particular subject (e.g. concerning 9/11) or for all recent articles (e.g. everything new in the last two days). They initially wish to see a full list of relevant articles including the title, creation date and short description. At this stage they do not need to view the whole article and accompanying photos for each article which meet their search conditions. This information can be returned later, after they have selected the article that particularly interests them. To improve performance, the table was split into four separate tables (each with a 1:1 relationship), the initial key information table (now containing the information most intensively searched for) being now only 2% the size of the original single table. The second table was used to store all other information, the third table the RTF articles themselves, and the fourth table the full-text search contents.

n:1

- n >= 0 Each category may contain one or more products, it may have no products.
- n > 0 Each category must contain at least one product.

As you can see n:1 relationships can be defined in accordance with your business logic and rules. The multiplicity is defined by yourself. You may need to define an n:1 relationship where n is > 0 but < 10. Maybe n can be `<null>`; when it is `<not null>` you are enforcing a relationship.

The demo database, `db1`, demonstrates a simple n:1 relationship whereby all products have one category, but one category can have many products or no products assigned to it.

n:m

A classic example can be seen in `db1`: one customer can purchase several products and a single product can be purchased by many customers. To make this happen you need to have some linking table in the middle. The `db1` example shows the link from `customer` to `orders`; `orders` is linked to `orderline` and `orderline` to `product`. All these relationships are built up using primary and foreign keys, thus forming an n:m relationship between customers and products. It is also possible to specify what should happen to these related data sets should one of them be updated or deleted. For example if you delete a customer in the `customer` table that has no orders (and therefore no order lines or products related to him) there is no problem. If however you attempt to delete a customer that has already placed orders, an error message will appear, due to a *violation of FOREIGN KEY constraint "FK_ORDERS_ID" on table "ORDERLINE".* This is necessary to maintain the database's integrity. Update and delete rules can be defined on the *Constraints* page in IBExpert's Table Editor. Please refer to *Constraints*, *Referential integrity* and *Table Editor/Constraints* for details.

To ascertain which relationships a table has with other database objects, and which dependencies other database objects have on a certain table, view the object editor's *Dependencies* page.

Data modeling using IBExpert's Database Designer

A simple method to initially design and visualize a new database is the IBExpert Database Designer. You can quickly and easily define what goes where, where are your key relationships, etc. It can also be used to graphically document an existing database, providing a logical view of the database structure and is an extremely quick and simple method to create views. Databases can be created or updated based on amendments made in the Designer by generating and running a script (please refer to *Generate Script*). They can be saved to file, exported and printed.

Create database

You can either use the command-line tool, `isql`, part of the Firebird package or the IBExpert SQL Editor to use DDL (Data Definition Language) to create your database manually. An easier option is to use the IBExpert Database menu item, Create Database.

Refer to the following subjects for further information:

- InterBase and Firebird command-line utilities - `isql`
  - DDL - `CREATE` statement
  - `CREATE DATABASE` statement
- Creating a database in IBExpert

Database objects

All database objects along with the how and when to use them are described in detail in the IBExpert documentation. Firebird/InterBase offer the following database objects:

- Domain
- Table
- View
- Stored procedure
- Trigger
- Generator
- Exception
- User-defined function UDF
- Role
- Index

The number of objects in a database is unlimited.

Understanding and using views

A view can be likened to a virtual table. It can be treated, in almost all respects, as if it were a table, using it as the basis for queries and even updates in some cases. It is possible to perform SELECT, PROJECT, JOIN and UNION operations on views as if they were tables. Only the view definition is stored in the database, it does not directly represent physically stored data.

Views simplify the visual display of of complex data. However when creating updateable views, a number of factors need to be taken into consideration.

Simple views displaying only one table can be updated as if they were a table. But complex views containing many tables can only update if the business logic has been well thought through and realized with triggers. This is necessary for the database to understand and know how it is to react in certain situations. For example, a user alters a category from cartoon to animation in a data set. Should the database a) allow the user to do this, b) alter the category just for this data set or c) alter the category for all films assigned to the cartoon category? Indeterminate views will damage your data integrity. Before creating a view, you need to decide whether to allow access to the view directly by the user, whether the user is only able to view data, or whether you wish to allow data updates using triggers or stored procedures.

You can simplify the relationships between data and tables for the user by flattening key information for them into a single view. We can add security by allowing users, for example, to update a film title but not allow them to alter a film category, by creating triggers on the view.

A further security option is to create views leaving fields with sensitive information (PIN numbers, passwords, confidential medical details and such like) blank. For example, in a product table with the fields: ID, FIRSTNAME, LASTNAME, ACCOUNT_NO, PIN,ADDRESS, ZIP and TOWN etc, a view of the table could be created as follows:

```
as
select
  id,
  firstname,
  lastname,
  account_no,
  '',
  address, etc.
```

Without suitable triggers and constraints, it is possible to add data to the "blank" column, it still cannot be seen in the view.

Another good reason for introducing views is for reasons of compatibility following data model improvements and the subsequent metadata alterations. For example, you need to split your product table up into two smaller tables, product_main and product_detail. All new triggers, procedures, exceptions etc. will be written based on these new table names and contents. However if you do not wish to update and alter all existing dependencies, you can simply create a view with the old table name and the old table structure. Universal triggers can be used to forward any data alterations made here onto the new tables.

Views can also be defined as stored SELECTS, for example:

```
CREATE VIEW Vw_Product_Short(TITLE,TXT)
AS
Select p.title,c.txt
from product p
join category c on c.id=p.category_id
```

Views can be created using SQL in IBExpert's SQL Editor and then saved as a view using the *Create View* icon. Alternatively they can be created in IBExpert's View Editor.

Once created, they can be treated in SQL SELECTs exactly as if they were tables:

```
select * from Vw_Product_Short
```

Further information can be found in the IBExpert documentation chapter, *Updatable views and read-only views*. For further information on IBExpert's View Editor, please refer to *View Editor*. To create a view in the SQL Editor, please refer to *Create view or procedure from SELECT*.

Comparing data models

IBExpert also offers you the possibility to compare the metadata of two different databases, and generate a script which alters the structure of the first database, making the structure the same as the second database.

A huge advantage of Firebird is that metadata can be manipulated and altered during runtime. Regardless of whether you are adding fields to tables or changing the basic structure, users can still work on the database data. Please note that there is a limitation of the number of metadata changes you may make to any single table, before having to perfom a backup and restore (please refer to *253 changes of table <table_name> left*).

Further reading (novice):

- Database Comparer

Futher reading (advanced):

- Automatic database structure comparison with recompilation of triggers and procedures
- Comparing databases using IBEBlock
- Comparing scripts with IBEBlock
- ibec_CompareMetadata

Programming the Firebird server

Many developers shy away from coding directly on the database server. IDEs (Integrated Development Environments) such as Delphi or C++ Builder may be easier to write and quicker and easier to debug. However, developing an efficient application with an intelligent database that offers the highest possible performance can only be achieved by a combination of the two, along with intelligent programming.

Reasons for server-side programming include:

Speed of execution: server-side programmíng does exactly what it says, the work is done on the server, and the results are sent out to the client (whether over a short internet line or worldwide). Client-side programming fetches all data and tables it might need, and then sorts and analyzes them on the client PC. So if you've got to perform computations on a large database or table, you've got to suck all the data back to the workstation to actually do the work. This can lead to time-consuming queries, traffic congestion and long wait times for the user.

It is possible to achieve up to 50,000 operations per second within a stored procedure. A Delphi or PHP application is considered efficient when it achieves just 3,000 operations a second. If you're skeptical, try migrating some of your code from your front-end to the server and test and compare the performance!

Consistency: database operations performed on the server are either completed successfully or rolled back (i.e. not executed at all). They are never partially completed. Another advantage of server-side programming is when you have different front-ends, e.g. Dephi and PHP, doing similar things, programming both to call a single procedure to perform a task is not just easier than programming the whole thing twice, it also ensures consistency. Both applications call the same procedure and are therefore guaranteed to provide the same result. Any alterations that may need to be made in the future only need to be made once, directly in the procedure.

Modularity: stored procedures can be written for singular tasks such as order taking, order processing and dispatch. They can then call each other. Modularity is clear/easy to comprehend, which also makes future adjustments easier. And in the example above (Delphi and PHP applications share the same database) modularity is achieved, as any alterations that may need to be made in the future only need to be made once, directly in the procedure.

Even though PSQL (Procedure SQL) is initially not so easy to write as IDEs as the programming language is not as rich and not as user-friendly, if you want to develop efficient high-performance database applications, it is vital you take the time and effort to get to grips with this.

See also:
Structured Query Language
PSQL
Stored Procedure
Writing stored procedures and triggers

Writing stored procedures and triggers

The stored procedure and trigger language is a language created to run in a database. For this reason its range is limited to database operations and necessary functions; PSQL is in itself however a full and powerful language, and offers more functionalities than you can use if you were just sat on the client. The full range of keywords and functions available for use in procedures and triggers can be found in the *Structured Query Language* chapter, *Stored Procedure and Trigger Language*. New features can be found in the *Firebird 2 Release Notes*.

InterBase/Firebird provides the same SQL extensions for use in both stored procedures and triggers. These include the following statements:

- DECLARE VARIABLE
- BEGIN … END
- SELECT … INTO : variable_list
- Variable = Expression
- /* comments */
- EXECUTE PROCEDURE
- FOR select DO …
- IF condition THEN … ELSE …
- WHILE condition DO …

Both stored procedure and trigger statements includes SQL statements that are conceptually nested inside the main statement. In order for InterBase/Firebird to correctly parse and interpret a procedure or trigger, the database software needs a way to terminate the CREATE PROCEDURE or CREATE TRIGGER that is different from the way the statements inside the CREATE PROCEDURE/TRIGGER are terminated. This can be done using the SET TERM statement.

Since IBExpert version 2005.03.12 there is added support for following Firebird 2 features:

- DECLARE <cursor_name> CURSOR FOR ...
- OPEN <cursor_name>
- FETCH <cursor_name> INTO ...
- CLOSE <cursor_name>
- LEAVE <label>
- NEXT VALUE FOR <generator>

Stored procedure

Firebird/InterBase uses stored procedures as the programming environment for integrating active processes in the database. Please refer to the IBExpert documentation chapter, *Stored Procedure* for the definition, description and variables of a stored procedure along with comprehensive instructions of how to use IBExpert's Stored Procedure Editor.

There are two types of stored procedure: executable and selectable. An executable procedure returns no more than one set of variables. A select procedure can, using the SUSPEND keyword, push back variables, one data set at a time. If an EXECUTE PROCEDURE statement contains a SUSPEND?, then SUSPEND has the same effect as EXIT. This usage is legal, but not recommended, and it is unfortunately an error that even experienced programmers often make.

The syntax for declaring both types of stored procedure is the same, but there are two ways of invoking or calling one: either a stored procedure can act like a functional procedure in another language, in so far as you execute it and it either gives you one answer or no answers:

```
execute procedure <procedure_name>
```

It just goes away and does something. The other is to make a stored procedure a little more like a table, in so far as you can

```
select * from <procedure_name>
```

and get data rows back as an answer.

Further reading:
Stored procedure
EXECUTE PROCEDURE
Stored procedure and trigger language
Stored procedure language

Simple procedures

An example of a very simple procedure that behaves like a table, using `SUSPEND` to provide the returns:

```
CREATE PROCEDURE DUMMY
RETURNS (TXT CARCHAR(10))
AS
BEGIN
  TXT='DOG';
  SUSPEND;
  TXT='CAT';
  SUSPEND;
  TXT='MOUSE';
  SUSPEND;
END
```

In this example, the return variable is `TXT`. The text `DOG` is entered, and by specifying `SUSPEND` the server pushes the result, `DOG` into the buffer onto a result set stack. When the next data set is written, it is pushed onto the result pile. Using `SUSPEND` in a procedure, allows data definition that is not possible in this form in an SQL. It is an extremely powerful aid, particularly for reporting.

FOR SELECT ... DO ...SUSPEND

```
CREATE PROCEDURE SEARCH_ACTOR(
    NAME VARCHAR(50))
RETURNS (
    TITLE VARCHAR(50),
    ACTOR VARCHAR(50),
    PRICE NUMERIC(18,2))
AS
BEGIN
  FOR
    select TITLE,ACTOR,PRICE from product
    where actor containing :name
    INTO :TITLE,:ACTOR,:PRICE
  DO
  BEGIN
    SUSPEND;
  END
END
```

This procedure is first given a name, `SEARCH_ACTOR`, then an input parameter is specified, so that the user can specify which name he wishes to search for. The columns to be returned are `TITLE`, `ACTOR` and `PRICE`. The procedure then searches in a `FOR ...SELECT` loop for the relevant information in the table and returns any data sets meeting the condition in the input parameter.

It is also possible to add conditions; below all films costing more that $30.00 are to be rounded down to $30.00:

```
CREATE PROCEDURE SEARCH_ACTOR(
    NAME VARCHAR(50))
RETURNS (
    TITLE VARCHAR(50),
    ACTOR VARCHAR(50),
    PRICE NUMERIC(18,2))
AS
BEGIN
  FOR
    SELECT TITLE,ACTOR,PRICE FROM PRODUCT
    WHERE ACTOR CONTAINING :NAME
    INTO :TITLE,:ACTOR,:PRICE
  DO
  BEGIN
    IF (PRICE<30)THEN PRICE=30
    SUSPEND;
  END
END
```

A good way of analyzing such procedures is to view them in the IBExpert Stored Procedure and Trigger Debugger.

To proceed further, the number of returns can be limited, for example, `FIRST 10`:

```
CREATE PROCEDURE SEARCH_ACTOR(
    NAME VARCHAR(50))
RETURNS (
    TITLE VARCHAR(50),
    ACTOR VARCHAR(50),
    PRICE NUMERIC(18,2))
AS
BEGIN
  FOR
    SELECT FIRST 10 TITLE,ACTOR,PRICE FROM PRODUCT
    WHERE ACTOR CONTAINING :NAME
    INTO :TITLE,:ACTOR,:PRICE
  DO
  BEGIN
    IF (PRICE<30)THEN PRICE=30
    SUSPEND;
```

```
  END
END
```

If you declare a variable for the `FIRST` statement, it needs to be put into brackets when referred to lower down in the procedure:

```
CREATE PROCEDURE SEARCH_ACTOR(
    NAME VARCHAR(50))
RETURNS (
    TITLE VARCHAR(50),
    ACTOR VARCHAR(50),
    PRICE NUMERIC(18,2))
AS
DECLARE VARIABLE i INTEGER;
BEGIN
  FOR
    SELECT FIRST (:i) TITLE,ACTOR,PRICE FROM PRODUCT
    WHERE ACTOR CONTAINING :NAME
    INTO :TITLE,:ACTOR,:PRICE
  DO
  BEGIN
    IF (PRICE<30)THEN PRICE=30
    SUSPEND;
  END
END
```

FOR EXECUTE ... DO ...

`EXECUTE STATEMENT` allows statements to be used in procedures, allowing dynamic SQLs to be executed contained in a string expression. Here, the above example has been adapted accordingly:

```
CREATE PROCEDURE SEARCH_ACTOR(
  NAME VARCHAR(50))
RETURNS (
  TITLE VARCHAR(50),
  ACTOR VARCHAR(50),
  PRICE NUMERIC(18,2))
AS
Declare variable i integer;
BEGIN
  i=10;
  FOR
    execute statement
    'select first '|| :I ||' TITLE,ACTOR,PRICE from product
    where actor containing '''||name||''''
    INTO :TITLE,:ACTOR,:PRICE
  DO
  BEGIN
    if (price>30) then price=30;
    SUSPEND;
  END
END
```

It is also possible to define the SQL as a variable:

```
CREATE PROCEDURE SEARCH_ACTOR(
  NAME VARCHAR(50))
RETURNS (
  TITLE VARCHAR(50),
  ACTOR VARCHAR(50),
  PRICE NUMERIC(18,2))
AS
Declare variable i integer;
Declare variable SQL varchar(1000);
BEGIN
  i=10;
  Sql =  'select first '|| :i ||' TITLE,ACTOR,PRICE from product
          where actor containing '''||name||''''
  FOR
    execute statement :sql
    INTO :TITLE,:ACTOR,:PRICE
  DO
  BEGIN
    if (price>30) then price=30;
    SUSPEND;
  END
END
```

Theoretically it is possible to store complete SQL statements in the database itself, and they can be called at any time. It allows an enormous flexibility and a high level of user customization. Using such dynamic procedures allows you to define your SQL at runtime, making on the fly alterations as the situation may demand.

Note that not all SQL statements are allowed. Statements that alter the state of the current transaction (such as `COMMIT` and `ROLLBACK`) are not allowed and will cause a runtime error.

The INTO clause is only meaningful if the SQL statement returns values, such as SELECT, INSERT ... RETURNING or UPDATE ... RETURNING. If the SQL statement is a SELECT statement, it must be a 'singleton' SELECT, i.e. it must return exactly one row. To work with SELECT statements that return multiple rows, use the FOR EXECUTE INTO statement.

It is not possible to use parameter markers (?) in the SQL statement, as there is no way to specify the input actuals. Rather than using parameter markers, dynamically construct the SQL statement, using the input actuals as part of the construction process.

WHILE ... DO

The WHILE ... DO statement also provides a looping capability. It repeats a statement as long as a condition holds true. The condition is tested at the start of each loop.

LEAVE and BREAK

LEAVE and BREAK are used to exit a loop. You may want to exit a loop because you've found the information you were looking for, or you only require, for example, the first 50 results.

By issuing a BREAK, if a specified condition isn't met, the procedure will break out of this loop and carry on executing past it, i.e. you go out of the layer you're in and proceed to the next one.

LEAVE is new to Firebird 2.0. The LEAVE statement also terminates the flow in a loop, and moves to the statement following the END statement that completes that loop. It is only available inside of WHILE, FOR SELECT and FOR EXECUTE statements, otherwise a syntax error is thrown.

The LEAVE <label> syntax allows PSQL loops to be marked with labels and terminated in Java style. They can be nested and exited back to a certain level using the <label> function. Using the BREAK statement this is possible using flags.

```
CNT = 100;
L1:
WHILE (CNT >= 0) DO
   BEGIN
   IF (CNT < 50) THEN
      LEAVE L1; -- exists WHILE loop
   CNT = CNT – l;
   END
```

The purpose is to stop execution of the current block and unwind back to the specified label. After that execution resumes at the statement following the terminated loop. Don't forget to specify the condition carefully, otherwise you could end up with an infinite loop! As soon as you insert your WHILE loop, specify whatever should cause the loop to finish.

Note that LEAVE without an explicit label means interrupting the current (most inner) loop:

```
FOR SELECT  ... INTO ......
DO
  BEGIN
  IF () THEN
     SUSPEND;
  ELSE
     LEAVE; -- exits current loop
  END
```

The Firebird 2.0 keyword LEAVE deprecates the existing BREAK, so in new code the use of LEAVE is preferred.

EXECUTE statement

To create a simple table statistic, we can create a new procedure, TBLSTATS:

```
CREATE PROCEDURE TBLSTATS
RETURNS (
  table_name VARCHAR(100),
  no_recordsInteger)
BEGIN
  FOR SELECT r.rdb$relation_name FROM rdb$relations r
     WHERE r.rdb$relation_name NOT CONTAINING '$'
  INTO :table_name
  DO
  BEGIN
     EXECUTE STATEMENT 'select count (*) from '||:table_name into :no_records;
  END
  SUSPEND;
END
```

This TBLSTATS fetches a table and a count, and goes through all tables, pushes the table names in and counts all data sets in the database, allowing you to see how large your tables are.

Recursions and modularity

If a procedure calls itself, it is recursive. Recursive procedures are useful for tasks that involve repetitive steps. Each invocation of a procedure is referred to as an instance, since each procedure call is a separate entity that performs as if called from an application, reserving memory and stack space as required to perform its tasks.

Stored procedures can be nested up to 1,000 levels deep. This limitation helps to prevent infinite loops that can occur when a recursive procedure provides no absolute terminating condition. Nested procedure calls may be restricted to fewer than 1,000 levels by memory and stack limitations of the server.

Recursive procedures are often built for tree structure. For example:

```
Create procedure spx
(inp integer)
returns
(outp integer)
as
declare variable vx integer;
declare variable vy integer;
begin
  ...
  execute procedure spx(:vx) returning values :vy;
  ...
end
```

The input integer is defined and the variables computed in some way. Then the procedure calls itself and the returning values are returned to another variable.

A good example of this is a typical employee table in a large hierarchical company, where the table has a column containing a pointer to the employees' boss. Every employee has a boss, and the bosses have bosses, who may also have bosses. If you wished to see a list of all bosses for one individual or the upstream management, then you could create a procedure selecting into and finish this with a suspend. Then it would go and call the same procedure again, this time with the resulting boss's ID. The procedure would carry on in this way until it reached the top level management, who answer to noone (the CEO).

Debugging

Up to Firebird version 2.1, Firebird offered no integrated debugging API at all. The only solution was to create log tables or external tables to record what the procedure was doing, and try to debug that way. However, as your triggers and procedures become more complex, an intelligent and sound debugging tool is vital.

Stored procedure and trigger debugger

IBExpert has an integrated Stored Procedure and Trigger Debugger which simulates running a procedure or trigger on the database server by interpreting the procedure and running the commands one at a time. It offers a number of useful functionalities, such as *breakpoints*, *step into*, *trace* or *run to cursor*,you can watch certain parameters, analyze the performance and indices used, and you can even change values on the fly. If you have Delphi experience you will easily find your way around the Debugger as key strokes etc. are the same.

Please refer to the IBExpert documentation chapter, *Debug procedure or trigger (IBExpert Debugger)* for details.

Optimizing procedures

Procedure operations are planned on *Prepare*, which means that the index plan is created upon the first prepare. When working with huge amounts of data, it is critical that you write it, rewrite it, look at each of the SQLs in it and break it down to ensure that it is optimally set up. A major contributing factor to the performance and efficiency of procedures are indices. The subject of indices is an extensive subject, which has been covered in detail in other areas of this documentation site:

- Index
- SQL Editor / Plan Analyzer
- SQL Editor / Performance Analysis
- IBExpert Table Editor / Indices
- Recompute selectivity of all indices
- Firebird for the database expert: Episode 1 - Indexes
- Enhancements to indexing in Firebird 2.0

Also take into consideration the use of operators such as `LIKE` and `CONTAINING`, as well as the use of strings such as , as none of these can use indices. For example, in the DemoDB, db1, compare:

```
select * from product where actor like 'UMA%'
```

The server returns all data sets beginning with the name `UMA`. If you examine the *Performance Analysis*, you will see that 56 indexed read operations were performed, and the Plan Analysis shows that the `IDX_PROD_ACTOR` index was used.

If however you need to view all records, where the name `UMA` appears somewhere in the `ACTOR` field:

```
select * from product where actor like ''
```

Now the server has had to perform 10,000 non-indexed reads, rather more than the 56 in the last example!

So if you can, use `STARTING WITH` instead of `LIKE` or `CONTAINING`. Check each procedure operation individually and remove bottlenecks, use the debugger, check the index plans, not forgetting to recompute the selectivity of your indices regularly. Use the Plan Analyzer and Performance Analysis to help you compare and improve your more complex procedures.

Another consideration with those extremely complex procedures is to postpone the SUSPEND. If you have a SUSPEND on every data row on a report that may be returning thousands of rows of calculated results, it will slow your system. If you wish to have an element of control over it, then put your SUSPEND every 100 or 1,000 rows. This way the database server fills a buffer and sends the results back in the specified quantity. It makes it more manageable, and you can stop it at any time should it congest your system too much.

Please also refer to *Optimizing SQL statements*.

Complex SELECTs or selectable stored procedures?

Selectable procedures can sometimes offer higher performance than complex selects. For example:

```
CREATE PROCEDURE SPPROD
RETURNS (TITLE VARCHAR(50),TXT VARCHAR(20))
AS
declare variable cid bigint;
BEGIN
  FOR                              --outer select
    Select p.title,p.category_id
    from product p
    INTO :TITLE,:cid
  DO
  BEGIN
    select c.txt from category c
    where c.id=:cid into :txt;      --inner select
    SUSPEND;
  END
END
```

This simple example is mimicking a join. You have a procedure here which is going to return a title and some text. First it goes through all the products, selecting the relevant titles. This outer select is however only providing one of the output fields. So another select is nested within the procedure, providing the information for the second output field, cid.

Although some developers feel there's no reason to construct procedures this way, ever so often you will find that the optimizer really has a problem with a certain join, because it takes too long for it to work out how to approach the query. Breaking things down like this can actually often provide a more immediate response.

Trigger

A trigger on the other hand is a special table- or database-bound procedure that is started automatically. After creating your database and constructing your table structure, you need to get your triggers sorted. Triggers are extremely powerful - the so-called police force of the database. They ensure database integrity because you just can't get round them. You, the developer, tell the system how to invoke them and whether they should react to an INSERT, UPDATE or DELETE. And once we're there in a table inserting, updating or deleting, it is impossible not to execute them. You can specify whether your trigger should fire on an INSERT or an UPDATE or a DELETE, or on all three actions (universal trigger).

Comprehensive details concerning triggers, how to create them, the different types and variables can be found in the IBExpert documentation chapter, *Trigger*.

Don't put all your logic into one trigger, build up layers of them, e.g. one for generating the primary key, one for logging or replication, one for passing on information of the data manipulation to another table etc. The order in which such a series of triggers is executed can be important. The before insert logging trigger needs to know the primary key, so the before insert primary key trigger needs to be fired first. The firing position is user-defined, beginning with 0. Please refer to *Trigger position* in the IBExpert documentation chapter, *Trigger*.

Using procedures to create and drop triggers

```
CREATE EXCEPTION ERRORTXT 'ERROR';
CREATE PROCEDURE createautoinc
AS
declare variable sql varchar(500);
declare variable tbl varchaR(30);
BEGIN
  FOR
    select rdb$relation_name from rdb$relations r
    where r.rdb$relation_name not containing '$'
    INTO :TBL
  DO
  BEGIN
     sql='CREATE trigger '||:tbl||'_bi0 for '||:tbl||' '||
         'active before insert position 0 AS '||
         'BEGIN '||
         '  if (new.id is null) then '||
         '  new.id = gen_id(id, 1); '||
         'END';
      execute statement :sql;
  END
  when any do exception errortxt :tbl;
END
```

This is a simple procedure which uses all table names (all tables are stored in `rdb$relations`) and creates a `BEFORE INSERT` trigger which adds an autoincrement ID. The following procedure then drops the trigger:

```
CREATE PROCEDURE dropautoinc
AS
declare variable sql varchar(500);
declare variable tbl varchaR(30);
BEGIN
  FOR
    select rdb$relation_name from rdb$relations r
    where r.rdb$relation_name not containing '$'
    INTO :TBL
  DO
  BEGIN
     sql='DROP trigger '||:tbl||'_bi0;';
     execute statement :sql;
  END
  when any do exception errortxt :tbl;
END
```

Firebird Administration using IBExpert

This documentation introduces DBAs to Firebird administration, with the emphasis on IBExpert as an aid to make your life easier. Even the more experienced Firebird DBAs will find a wealth of tips here.

Source: Firebird School at the Firebird Conference 2007 held in Hamburg, Germany

Firebird Administration using IBExpert

Administration tasks

The Firebird DBA really does have an easy job as their are no administration tasks which he *has* to do! And when the application is programmed well, absolutely no maintenance is necessary! However databases do occasionally encounter problems, usually due to poor programming. So here are a few things the Firebird DBA should be aware of.

Downloading and installing the various Firebird versions

Please refer to the previous chapter, *Download and Install Firebird*.

Automating the database backup and restore

It is not necessary for users to logout during a Firebird backup. A consistent backup is performed, regardless of whether users are working on the database at the time. A database backup can be performed using the IBExpert Services menu item, Backup Database, or the Firebird command-line tool, GBAK.

For obvious reasons, should you need to perform a database restore, it is vital that no users are working on the database during the restore. A database restore can be performed using the IBExpert Services menu item, Restore Database, or the Firebird command-line tool, GBAK. Please note that if you run the GBAK restore in verbose mode, it can take an awful long time.

```
C:\>gbak
gbak:legal switches are:
        -B(ACKUP_DATABASE)     backup database to file
        -BU(FFERS)             override page buffers default
        -C(REATE_DATABASE)     create database from backup file
        -CO(NVERT)             backup external files as tables
        -E(XPAND)              no data compression
        -FA(CTOR)              blocking factor
        -G(ARBAGE_COLLECT)     inhibit garbage collection
        -I(NACTIVE)            deactivate indexes during restore
        -IG(NORE)              ignore bad checksums
        -K(ILL)                restore without creating shadows
        -L(IMBO)               ignore transactions in limbo
        -M(ETA_DATA)           backup metadata only
        -MO(DE) <access>       "read_only" or "read_write" access
        -N(O_VALIDITY)         do not restore database validity conditions
        -NT                    Non-Transportable backup file format
        -O(NE_AT_A_TIME)       restore one table at a time
        -OL(D_DESCRIPTIONS)    save old style metadata descriptions
        -P(AGE_SIZE)           override default page size
        -PAS(SWORD)            Firebird password
        -R(ECREATE_DATABASE) [O(VERWRITE)] create (or replace if OVERWRITE used)

                                 database from backup file
        -REP(LACE_DATABASE)    replace database from backup file
        -RO(LE)                Firebird SQL role
        -SE(RVICE)             use services manager
        -T(RANSPORTABLE)       transportable backup -- data in XDR format
        -USE_(ALL_SPACE)       do not reserve space for record versions
        -USER                  Firebird user name
        -V(ERIFY)              report each action taken
        -Y  <path>             redirect/suppress status message output
        -Z                     print version number
gbak:switches can be abbreviated to the unparenthesized characters
```

When performing a backup only the index definitions are stored, then when the database is restored, data are restored into the tables, and right at the end the indices newly generated. Backup and restore also resets all transaction parameters, that can be viewed in the Database Statistics.

Alway backup onto another machine. Check that the file stamp is differenct and do a test restore regularly to confirm that are backup files are fine.

It is possible to automate the database backup in a batch file in the Windows *Scheduled Tasks*.

A great tool for automating your backups and restores is the IBExpertBackupRestore Scheduler. This enables you to automate backups and restores, and can send you an e-mail to inform you of any errors or confirming that there were no errors.

See also:
InterBase and Firebird command-line utilities
IBExpertBackupRestore
Backup

Garbage collection

Garbage collection is the ongoing cleaning of the database and is performed in the background around the clock. This constantly reaorganizes the memory space used by the database. If you don't clean up, database performance will slowly but surely degrade. Garbage collection works for both data pages and

index pages (if you have created 100,000 new data sets and deleted another 100,000 data sets, an index won't help much, if the 100,000 deleted pages are still there and being searched through.

The Firebird garbage collector does not require administrative commands or manuel maintenance as certain other database environments do. Whether the garbage collector works efficiently or not depends on how the application works.

For further information regarding garbage collection, please refer to the IBExpert Services menu item, Backup Database / Garbage collection.

See also:
Garbage
Garbage collectors

Setting up protocols

Your database is full of information. Sometimes it is helpful to log certain aspects of the information manipulation (selects, inserts, update, deletes), to gain an insight what is really happening in your database.

- **Manual:** Create a trigger on each table where you want to have a protocol
- **Almost automatically:** take a look at the script `db2.sql` found in the IBExpertDemoDB folder, which creates a fully functional transaction log just by executing the procedure `INITLOG`.
- **Automatically:** Open the table you wish to log in the IBExpert Table Editor and click on the *Logging* page. Confirm the generation of `IBE$` System tables if required, and then select *Prepare table for logging*.
- Other tools with advanced log functions can be found in the IBExpert Tools menu item, Log Manager.

See also:
Bidirectional replication for InterBase and Firebird
Log Manager

Setting up and testing the ODBC driver

If you need an ODBC driver, it can be downloaded from http://www.firebirdsql.org. Then use the Windows menu: *Settings / System Control / Administration / Data Source* and select `fbodbc`. This now allows you to access Firebird data from non-Firebird applications such as, for example, OpenOffice Base.

Should you wish to import data from other data sources, please refer to the IBExpert Tools menu item, ODBC Viewer.

Importing and exporting data

The Firebird core only offers import and export using external files, which requires a setting in and restarting the server.

The files can be defined by declaring a table:

```
create table external file
```

This function is extremely quick; 100,000 data sets can be imported or exported every second. It is however limited for certain datatypes, particularly those of a variable length, such as blobs. The best solution is to define the table using the above instruction, and defining as far as possible all fields as `CHAR`.

You can alternatively use the Firebird ODBC driver with any ODBC-capable tool, the IBExpert ODBC Viewer, or IBExpert's IBEBlock, ODBC support. You can even automate your import/export using IBEBlock.

Results of SQL queries can be exported from the IBExpert SQL Editor.

See also:
IBEBlock examples including data import and export

Detect and avoid database errors

Typical reasons for corrupt databases include:

- File system backup tools
- Anti-virus tools
- Hard drive defect
- Server crash with forced writes inactive.

Database errors can be detected from Firebird error messages and entries in the `firebird.log` file.

More about database corruption can be found the *Database Technology Articles* section. Damaged databases can be repaired using `GFIX` or IBExpert.

Database repair

Database repair using GFIX

SET ISC_USER=SYSDBA SET ISC_PASSWORD=masterkey

```
Copy employee.gdb database.gdb
```

Validate database:
```
gfix -v -full database.gdb
```

On error try mend:
```
gfix -mend -full -ignore database.gdb
```

Check again:
```
gfix -v -full database.gdb
```

On error try backup without garbage collection:
```
gbak -backup -v -ignore -garbage database.gdb database.gbk
```

Finally try restore:
```
gbak -create -v database.gbk database.gdb
```

See also:
GBAK
GFIX

Alternative database repair methods

Database corruption can occur at any time in any part of the database. The sudden panic that often accompanies such a serious problem can be mitigated by planning for the worst case scenario, before ich actually happens: who to call, what to do. Having a plan and executing it. Our proposal: always have a warm backup copy of the database as read-only. Most companies can function with a read-only database for at least a few hours without critically failing the business, giving you time to put your contengency plan into action. Always rely on two databases: the live and the replicated; so with the knowledge that you can switch in an emergency with minimal loss of data.

For more information about replication, please refer to *Bidirectional replication for InterBase and Firebird*.

Then you will need to begin to analyze your problem, locate it and, as far as possible, repair it.

Begin with GFIX. If that doesn't bring you any further, limit the damage to as few data sets as possible, and use IBExpert's Extract Metadata to extract all healthy data. Please refer to our article: *Database repair using Extract Metadata*.

See also:
Database corruption
Firebird for the database expert: Episode 3 - On Disk Consistency
Preventing data loss

Typical causes of server problems

Network problems

If you encounter network problems try to ping the server. Check the `firebird.log`, as this can indicate where the source lies.

Approximately half the problems with failure to reach the server are due to a Firewall. If you're using the default port 3050 make sure this is listed in your Firewall settings. Although Firebird normally only requires one port, this is not the case, if you use the Event Alerter. The Event Alerter is a mechanismus with which you can trigger a message, when a certain event occurs, to be sent to a client. These Event Alerters are a powerful feature. As soon as you register any events with the Firebird server it will open a separate port. You can specify which port in the `firebird.conf` file. Otherwise it selects a random port.

Hardware problems

One of the issues on Firebird server hardware is running out of disk space, often due to temp files. Many DBAs don't set their temp directory in `firebird.conf`, and often forget to check the temp directory when they notice the're running out of space. When the hard drive begins to become full, Windows storesdata pages anywhere it can find space. Which of course degrades performance when searching for and uploading the data on these pages. Please refer to *Temporary files* below for further information.

OS problems

When performance starts to degrade it's important not just to look at queries and programming, but also at the operating system itself.

1. **Windows system restore:** On Windows *My Computer / System Properties* the automatic *System restore* can be disabled. This also prevents Windows copying all manner of file into the `Win/System32/dllcache` directory (it not been unknown to discover files of 5GB and more in this directory!).

2. **Automatic Windows update:** the infamous automatic Windows update with it automatic rebooting is the cause of many Firebird server machines suddenly being shut down, because noone was sitting in front of the screen to stop it. This must be disabled! And it's not just Windows. There are many other services running that may deny you server access.

So prevent any updates running and rebooting your system automatically, even antivirus applications. Close everything up, leaving only those really vital ports free. Backups can be configured via ftp onto a backup server.

As far as possible, use a dedicated server for your Firebird applications.

Detect and avoid server problems

Check the Firebird logs from time to time. This provides an opportunity to notice things that users don't realize are going wrong. Check the Windows Event log as well. When the daily log starts to increase in size, look for the causes, e.g. that the server is often restarted. The cause of frequent Firebird server reboots is often due to UDFs. Writing robust UDFs is vital. Poorly written UDFs can lead to technical suicide, if you are not familiar with memory management. If 2 processes are using the same UDF simultaneously, it can well lead to server instability. Before you go ahead and write your own UDFs for everything, consider taking an existing one from a library such as FrreAdhocUDF, and complement it if necessary.

Recommendation:

- Use only robust UDF libraries, such as FreeAdhocUDF.
- Check every UDF you've written yourself not just once, but 10 times!

If you're using two difference Firebird/InterBase flavors concurrently, check that the correct `fbclient.dll`/`gds32.dll` version is installed on the server and all clients. You'd be amazed how often DBAs are surprized by this or that previously undiscovered `dll` suddenly turning up, because somewhere there is an old InterBase version installed (and maybe even still running). When you start your Firebird 2 database, it tries to work with the old `dll`. Ensure that at least the correct client library is available in your application directory for the application's database version.

Remove any old redundant InterBase versions.

Use the IBExpert Communication Diagnostics to test connect to your server. Analyze any error messages returned. Alternatively attempt a connection at TCPIP level and pinging the server. When the server can't be reached this way, it is obviously not a Firebird problem. Please refer to the IBExpert Services menu item, Communication Diagnostics for further information.

Understanding the log file

Go back through the last couple of months logs and search for patterns. Then the source of many problems often goes back that far (eg. page corruptions are not always immediately noticeable). There are a few typical unimportant entries, such as

```
INET/inet_error: connect errno = 10061
```

or the Guardian restarting and of course, a routine shutdown.

There are however, a few important entries which you should take note of, should they appear in your log.

- **Terminated abnormally:** an indication that someone has shut down your Firebird server by pulling the plug.
- **Modifying procedure xxx which is currently in use by active user requests:** this occurs fairly often with Firebird 2. It's not critical if you modify a procedure whilst others are using it. The problem arises due to the multi-generational architecture - when others are working with the procedure, you can only see the results of the old procedure.
- **Page xxx is an orphan:** if this message starts to occur regularly, perform a backup and restore.
- **Page xxx wrong type:** this one's pretty terminal, because it's a clear indication that your database is corrupt. It is important to determine which pages are affected, because they not be in use any more, or only store old record versions. In this case the problem will be solved by the next database sweep. On the other hand, if you're unlucky the next database sweep will turn it into a real problem!

Temporary files

Firebird temp files are created when something needs to be sorted or combined from multiple tables and no index is usable or there is not enough sort memory available.

Firebird temp files begin with `FB` and, by default, they are stored in the Windows `/temp` directory, when the Firebird server is installed as a service. The Firebird temp directory can be altered and specified in the `firebird.conf`.

Temp files can get very big very quickly. One of the reasons for this is that they include the full space for long `CHAR` or `VARCHAR` columns. If you need large character fields, use a blob field. The size of a blob field is dependent on the database page size, for example, in a database with a page size of 8 KB, the maximum blob size is 32 GB.

Memory configuration

Memory settings depend on the one hand on the database page size and on the other the default cache pages specified in `firebird.conf`. The default value is 2048 of the database pages are reserved for the cache. This value can be altered in the `firebird.conf`, the maximum value being 128,000. However, if the memory specified in the `firebird.conf` (number of pages multiplied by the page size) is larger than the actual available memomory, it will not be possible to open the database!

We therefore recommend leaving the default size in the `firebird.conf` as it is at 2048, and instead, define in the IBExpert Services menu item, Database Properties, that the database should use 20.000 pages for the cache. The KB size is calculated automatically, and this is the quantity of bytes which remains in the working memory, which of course speeds up the database performance. This cache buffers setting for the database overrides the default cache pages in `firebird.conf`.

Please note:

- **SuperServer:** cache memory per *database* = page size * buffers
- **Classic server:** cache memory per *connection* = page size * buffers

Therefore it is important to define the cache memory for the Classic server at a lower level than for the SuperServer.

See also:
Page size

Optimization

This section concentrates upon the performance optimization of your Firebird server. With any system there is always a limiting factor. If you remove that limiting factor, something else then in turn becomes the limiting factor. It is therefore vital to be aware of all these factors which contribute to your overall database server performance.

Operating systems

Certainly the popular operating system today is Microsoft, although Linux is constantly improving its strong foothold in the market. With regard to Windows it is fairly irrelevant which version you use. Windows 2000 does have the advantage however, that it does not carry as much overhead as Windows XP and co. Physically it can be roughly estimated, that a Firebird server installation on Windows working with VMware, the performance is approximately 30% less than native processor use. VMware offers a number of advantages, for example that you can back up the complete VMware, complete with database, configuration etc., enabling the database to be restarted immediately with the same IP address. And VMware files are pretty well impossible to corrupt.

Performance variations are minimal when using the same hardware and the same Firebird version. Slight discrepancies in different areas may be detected, these having different advantages and disadvantages, which need to be assessed individually for indivual application requirements.

The real advantage with Linux is quite simply the stability of the total system. With Windows it is possible to achieve a high level of stability, there are a number of parameters and settings that need to be accordingly configured. Linux is certainly better with regard to memory configuration, and the larger the application, the more advantages you will discover with Linux. And if you wish to run a web server alongside your Firebird server on the same machine, you should definitely consider Linux.

If however you have a classic medium-sized system with 10-20 users, you will not detect any significant differences in overall performance.

Optimal hard disk use

The optimal hard disk configuration for an efficient Firebird server is to have separate dedicated hard disks for the operating system, database and temp files. Partitions are of no advantage here, as the read/write head still has to scan the whole drive. The decisive factor with fixed disks is the read/write speed; and a large cache can also improve performance.

Raid systems are useful for large databases, and the larger the disk cache the better.

Small databases up to 2 GB can fit in the cache RAM – that can be the database cache RAM or just the Windows cache RAM.

Optimizing hardware configuration

Take into consideration the following factors when looking at optimizing your hardware:

- Multicore CPU are useful for the Firebird Classic server, at least two cores are advisable for the SuperServer - for the server itself, and another for events.
- Large cache server CPUs (Xeon, Opteron) are useful for all architectures - particularly with large databases ith a high number of users.
- Server main boards are optimized for I/O speed.
- High speed RAM DDR3/DDR2.

Optimizing OS configuration

Firstly, remove all unnecessary tasks and services from the database server. Scrutinize anything listed in the Task Manager, when you are unsure why it's there, stop it running, and if possible deinstall the application that started it in the first place. A Windows system can run with a minimum number of processes on dedicated database server.

High performance database servers should not be used for anything else, be it file servers, mail servers (every time they do a POP grab, you're bound to register a discernable drop in database performance), or print servers and the like. No antivirus software is at all necessary, no backup/restore software that handles open file backup, especially not for the database files but also for the temp files. Even when invoking a shadow, by backing up your database files, serious degradation can be noticed in the overall server performance, particulary if you have intensive user traffic at the time. Refer to *Automating the database backup and restore* to automate backups to be performed at a low traffic time period.

And please do not run a 3D OpenGL screen saver; fancy screen savers also contribute to performance degradation! And if you're using Linux, run the server without the GUI to save even more memory that can be better used by your database server.

Firebird benchmarks tests

The IBExpertDemoDB can be used for simple server benchmark tests. By running the `db1.sql` it is possible to quickly determine discrepancies in performance on different hardware and OS configurations. Please refer to *IBExpert Benchmarks* for details of benchmarking possibilites using IBExpert tools.

Important: when benchmark testing, take into consideration the potential database size and number of users in a year's time. Testing performance on double your current database size with double the number of users will offer you the comfort factor in the near future!

Optimizing the database

1. Split complex tables into several smaller ones (Database normalization).
    - For reasons of compatability with legacy databases, it might help to add an updatable view with the name of the old table and with the same structure.
    - Old source code can still use the old name for `SELECT`, `INSERT`, `UPDATE` or `DELETE`; new source code can work directly on the new smaller tables.

   This can provide a real improvement in speed, especially in the case of very complex tables. Typically it also improves the restore speed considerably.
2. Do not use GUID for primary key fields, as these use much more space and will be slower as an `INTEGER` or `BIGINT`.
3. Do not use very long `CHAR`/`VARCHAR` fields unless they are really necessary.
4. Seldom-used columns should be stored in different tables.
5. Use indices only where necessary.
6. Compound indices should only be used on large tables.

Parameters for optimal performance

1. Database model - if your database model is weak no amount of tweaking other parameters will make any significant difference. Read the *Database design and database normalization* article and use IBExpert's Database Designer to optimize your database model.
2. Test SQL statements (refer to *Optimizing SQL statements* for further information).
3. Analyze index plans - tons of information, examples and tips can be found here: *Index statistics*, *Index*, *Performance Analysis*.
4. Transaction control - monitor, analyze and improve.
5. Server-side programming - let the server do the work, rather than transferring masses of data pages to the client and performing your queries there.
6. Optimizing cache - refer to *Temporary files*, *Memory configuration* and *Optimizing hardware configuration* for further information.
7. Hardware
8. Operating System
9. Network

Index statistics

Imagine the following situation: you have a database of all the inhabitants of Great Britain. You require a list of all men living in Little Bigton. How should the server process the query? The population of Great Britain is currently around 60 million. Approximately half are men. Should the server first select all men (around 30,000,000) and then take these results and select all those who live in Little Bigton, or should it first select all residents of Little Bigton (which let's say has a population of around 5,000) and then select all men?

The best selectivity is of course to first select all residents of Little Bigton, and then discern the number of males. The problem is that when you send the query to the server, it needs further information to help it decide how to go about executing the query. For this it uses indices, and to decide which index is the best to use first, it relies on the index selectivity.

Refer to the following articles for further information regarding indices and index statistics:

- Index
- SQL Editor / Plan Analyzer
- SQL Editor / Performance Analysis
- IBExpert Table Editor / Indices
- Recompute selectivity of all indices
- Enhancements to indexing in Firebird 2.0
- Firebird for the database expert: Episode 1 - Indexes
- Recreating Indices 1
- Recreating Indices 2

Secure data transfer

Many applications may have external users, who need to connect to the database remotely and access or exchange database data, often over dialup, satellite or public wide area networks. There are two key issues here: firstly that by using public band widths there is a security risk. Secondly, even reasonable amounts of data can congest a poor band width without compression.

Compression reduced the file size, which increases speed. However the big issue for connection speed is latency, which can be measured for example by pinging the server. Latency is a more critical factor than the bandwidth.

Many people set up VCNs through to their service, which solves both issues. The VPN does the compression for you and provides you with a secure tunnel. Alternatively there is an excellent free tool on the market, Zebedee, offering a tunnel that can be used to compress and encrypt the TCP traffic between the Firebird server and the client, similar to SSH or SSL. Basically you have a small piece of software sitting on the server and on the client. You need to specify some port redirections and it listens on one port, decompresses the data and pushes it through to the correct port where the Firebird server (or Firebird client) can be reached. By return it compresses and encrypts data going out. It is even possible to specify client ID files so that the connection is only allowed when the respective client ID files are available both on the server and the client

The software can be downloaded from http://www.winton.org.uk/zebedee and is available for Windows, Linux and Unix. It is open source and completely free.

Optimizing SQL commands

Tips for optimizing SQL commands can be found in the SQL Editor chapter, *Optimizing SQL statements*.

If you are new to Firebird SQL please first read *Firebird Development using IBExpert* for a comprehensive introduction to Firebird SQL.

The following references provide full syntax and examples of Firebird SQL:

- *SQL Language Reference* (InterBase 6.0, Firebird 1.x)
- *Firebird 2 SQL Reference Guide* (Firebird 2.x)

Firebird SQL Server 2.x Administration Handbook

5th November 2007

Stefan Heymann Copyright © 2007 Consic Software Engineering
heymann@consic.de
translated into English by Debra J. Miles, Copyright © 2008 IBExpert KG
www.ibexpert.com

About this book

This handbook is a guide for Firebird database administrators. It is not a constituent of the official Firebird documentation. This is not a comprehensive guide; it includes those features and details necessary for the installation and operation of a typical database for small and medium-sized applications.

This handbook relates to Firebird 2.0 versions and upwards. An edition referring to Firebird 1.5 *[German language only]* can be obtained from Consic.

The current version of this handbook *[German language only]* can be downloaded from http://www.consic.de/firebird.

About Firebird

Firebird is, along with MySQL and PostgreSQL, the most successful open source database for professional applications. Firebird provides all important functionalities that the large databases such as Oracle, DB2, Sybase and MSSQL offer as a matter of course. Views, triggers, procedures, user-defined functions and a stable transaction model provide for a robust and powerful platform for database applications.

The Firebird server can look back at over 20 years development history. It is successdfully deployed by the German Press Agency (dpa) and German Telekom in key applications used by several hundred users. The license model allows it to be deployed - even in a commercial environment - totally free of charge. Firebird is available for Windows, Linux, Sun, Mac and other operating systems.

In 2007 Firebird was awarded the Sourceforge Choice Community Award in two categories:

- Best project for the enterprise
- Best user support

Installation

Preliminary considerations

Firebird is available for the following platforms:

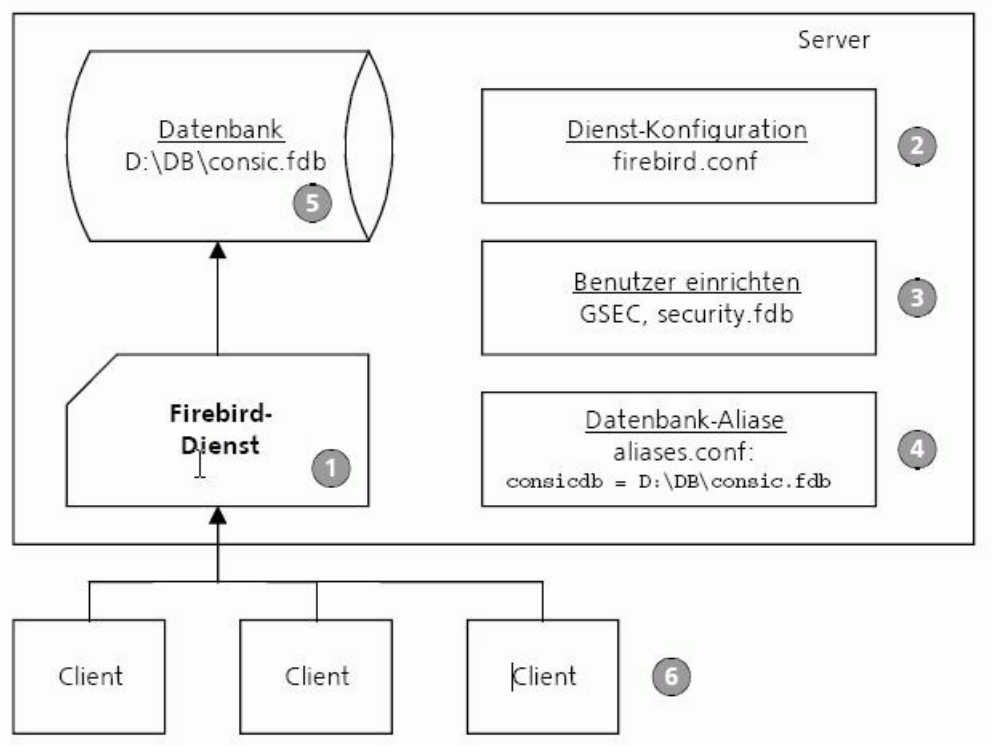
- Win32 (Windows 2003, XP, 2000, NT4, etc.)
- Linux (i386, AMD64)
- FreeBSD (Intel), HP/UX, Mac OS-X and Sun Solaris x86. These are not described in any further detail in this handbook.

Firebird is an extremely slim server, the full installation requires less than 20 MB hard drive space. Clients can also be installed on all supported operating systems. The clients are also slim and in their simplest form the DLL comprises but a few hundred kilobytes (`fbclient.dll`).

Terms and definitions

Server: The computer upon which the Firebird service is running.
Service: A Windows service or Linux demon.
Database: a file (or connected multiple files), that contain a related set of tables, indices, procedures, triggers etc.

Procedure



The procedure for installing the database server, database and clients roughly follows these steps:

1. Installation of the Firebird service on the server.
2. Verification or alteration of the services configuration (`firebird.conf`).
3. Specification of users and if necessary, alteration of the SYSDBA password.
4. Alias definition for each database (`aliases.conf`).

5. Setting up the database.
6. Client installation.

See also:
Download and install Firebird
Firebird 2 Quick Start Guide
Firebird 2.0.4 Release Notes

SuperServer, Classic server

The Firebird service can be installed as a "SuperServer" or "Classic server":

- **SuperServer:** a new thread is started for each incoming client connection. It is quick and requires less system resources.

  Recommended.

- **Classic server:** a new process is started for each incoming client connection. This can scale better on multi-processore machines.

Windows installation

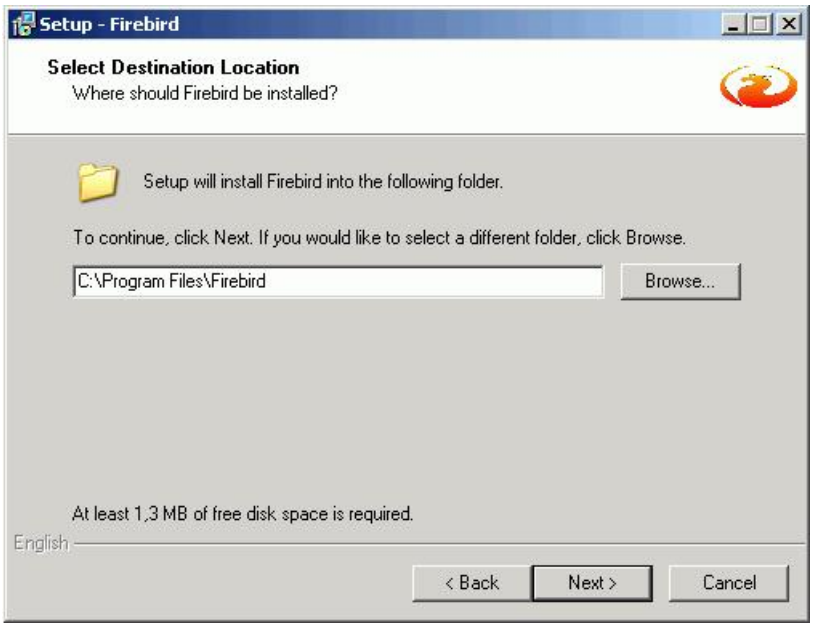
Start the installation program (e.g. `Firebird-2.0.3.12981-1-Win32.exe`).

See also:
Server versions and differences

Target directory

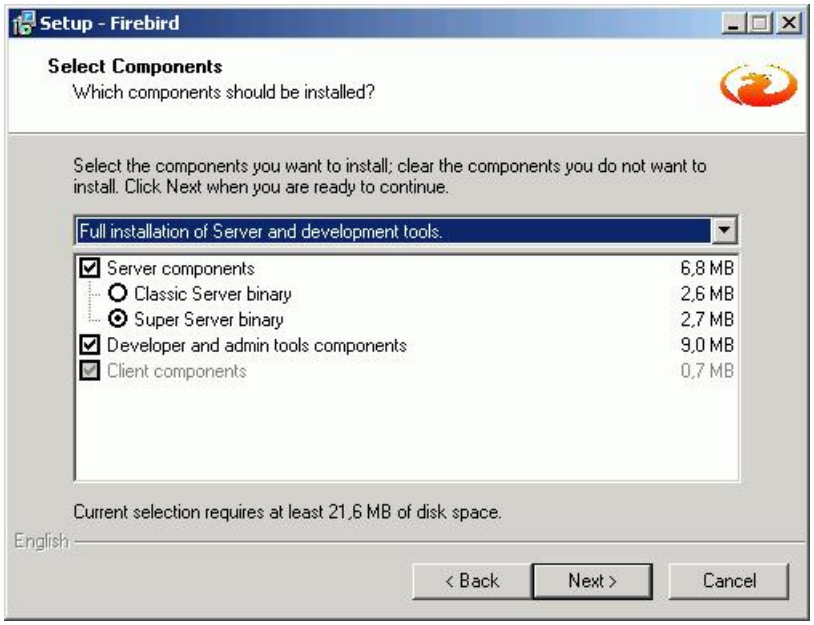
It is possible to install multiple Firebird services on a single system, although this will be seldom necessary for an operative installation.

We recommend abbreviating the directory path proposed by the install wizard and using the directory: `C:\Program Files\Firebird`:



Components

For productive environments we recommend installing the SuperServer as this consumes less resources. The *Server components* and *Developer and admin tools components* should also be installed:



Additional tasks and functions

- **Guardian:** The Firebird Guardian can be installed along with the Firebird service. This is a monitoring utility that does nothing other than check whether the Firebird server is running or not, restarting it if necessary (Watchdog).
  Recommendation: Use the Guardian.
- **Run the Firebird Server as an application or service:** Firebird should be run as a service on productive servers.
  Recommendation: Run as a service.
- **Start Firebird automatically every time you boot up** Of course!
  Recommendation: Yes.
- **Copy Firebird client library to <system> directory:** If this option is checked, the Firebird client DLL `fbclient.dll` is also copied in the Windows System directory, and can be found more easily by applications. This isn't necessary on pure database server machines. It can however be advantageous (although not essential) on servers that co-function as file servers or for administrative purposes.
  Recommendation: don't check this option.
- **Generate client library as GDS32.DLL for legacy app. support:** Check this option, if you still want to run Borland InterBase™ applications, which will expect the presence of a `GDS32.DLL`.
  Recommendation: don't check this option.

Following installation the Firebird service and, if selected, the Firebird Guardian service are immediately ready for use (*Automatic start*).

Services

The following services appear in the Windows *Services* panel:

- **Firebird Guardian:** `DefaultInstance` (if installed)
- **Firebird Server:** `DefaultInstance`

If the Guardian has been installed, it is sufficient just to start and end the Guardian. Otherwise the database services can be started manually or automatically in the usual Windows way.

Ports

Firebird listens by default to TCP Port 3050, the service is called `gds_db`. This can however lead to conflicts, if another Borland InterBase™ database is already running on the same machine. In this case, the port needs to be altered in the `firebird.conf` (found in the Firebird root directory) and, if necessary, an entry in the `etc/services` file. Apart from that, no further changes are necessary.

Example: changing the port to 3051:

Specify in `firebird.conf`:

```
RemoteServiceName = firebirdsql
RemoteServicePort = 3051
```

Specify in the `services` file:

```
firebirdsql       3051/tcp      # Firebird 1.5 Server
```

Databases

The database files need to be on the same local file system as the database service itself. Netwerk drives cannot be used (independent of whether these correspond via a UNC path or a drive letter).

Database administrator SYSDBA

When a service is first installed there is only one database user: the `SYSDBA`. This user has the password, `masterkey`.

To change the SYSDBA password, use the command-line GSEC tool, stored in the Firebird bin directory. Enter the following command (directly on the server wher the database service is running):

```
cd \Programme\Firebird\bin
gsec -user sysdba -password masterkey -modify sysdba -pw <new password>
```

When the password for example should be defined as master, enter the following:

```
gsec -user sysdba -password masterkey -modify sysdba -pw master
```

Only the first eight characters of the password are significant. If the new password is longer, the following warning appears:

```
Warning - maximum 8 significant bytes of password used
```

See also:
Download and install Firebird
Firebird Administration
Configuring Firebird

Installing on Linux

Install the rpm package with a suitable package tool. It sets up the Firebird demon and the Firebird Guardian demon. The Guardian is a watchdog demon, checking if the Firebird demon itself is still running and restarts it if necessary.

rpm Package Manager

Firebird can be installed and deinstalled using the command-line Package Manager:

Installation

```
rpm -ivh <rpm-Datei>
```

for example:

```
rpm -ivh FirebirdSS-1.5.1.4481-0.i686.rpm
```

Deinstallation

Determine the exact package names with:

```
rpm -qa Fire*
```

Deinstall using:

```
rpm -e <Package-Name>
```

For example:

```
rpm -e FirebirdSS-1.5.1.4481-0
```

Database administrator SYSDBA

When service is newly installed there is just one user: the SYSDBA. This has a password allocated by the installation, which can be found in the SYSDBA.password file, found in the Firebird root directory (usually /opt/firebird). To change this automatically generated password, which is difficult to memorize and type, use the bin/changeDBAPassword.sh script.

Write permission

Important: the user account firebird requires write permission on all directories, in which databases are to be stored!

Firebird Manager fbmgr

The Firebird server process can be started and stopped using the Firebird Manager fbmgr. fbmgr can be found in the bin subdirectory. It can only be started by an administrator.

Starting up

The service process can be started using the –start option:

```
./fbmgr -start
```

A further option may be specified to determine whether the Guardian should also run or not:

–once Starts Firebird without the Guardian. –forever Starts Firebird with the Guardian. This is the default parameter.

```
./fbmgr -start -forever
```

Shutting down

Using the option `-shut` all transactions are rolled back (ROLLBACK), all client connections disconnected and the service process shut down. The SYSDBA password has to be specified:

```
./fbmgr -shut -password masterkey
```

See also:
Using IBExpert and Delphi applications in a Linux environment: accessing Firebird

Windows client installation

The client installation can install either a minimal client, that can be used to start Firebird applications, or a client together with the administration tools.

Start the same setup program that was used to install the services. Under *Select components* select the option *Install client tools for developers and database administrators*.

You should carry out this installation on all computers, where administrative tasks are to be done. Tools such as GBAK, GFIX, GSEC etc. are installed here.

There are some applications that have their own client, in this case a separate installation is not necessarily required.

See also:
Download und Install Firebird
Firebird Administration

Service Configuration

The installation

The complete Firebird installation has a Firebird root directory with a number of subdirectories. The directory structure in Windows and Linux is identical.

Firebird root directory

Files (important files in bold):

aliases.conf	Configuration file for database aliases.
firebird.conf	Configuration file for the server.
firebird.log	Error protocol.
firebird.msg	Server messages.
<rechnername>.lck	Lock file.
readme.txt	Service readme file.
security.fbk	Data backup of the security database.
security.fdb	Security database: comprises user names and passwords.
IDPLicence.txt IPLicence.txt	License regulations for Firebird (Firebird is open source, the license allows free circulation and use, even for commercial purposes).

Windows bin subdirectory

fbclient.dll	Client access library.
fbguard.exe	The Firebird Guardian service.
fbserver.exe	The actual Firebird database service.
gbak.exe	GBAK tool for backup and restore.
gdef.exe	GDML tool (outdated, no longer used).
gfix.exe	GFIX tool: settings, repair, administration.
gpre.exe	GPRE-Tool: C preprocessor.
gsec.exe	GSEC tool: user administration.
gsplit.exe	GSPLIT tool.
gstat.exe	GSTAT tool: statistics.
ib_util.dll	Utilities.
icu*.dll	Different libraries for the support of international character sets.
instclient.exe	Client library installation as gds32.dll in Windows System directory (usually not necessary).
instreg.exe	Registration of an installation in the registry (only necessary when installing manually).
instsvc.exe	Tool for installing/deinstalling the service and for the start and shutdown pf the service (only necessary when installing by Hand).
isql.exe	ISQL tool: Interactive execution of DDL and DML commands, execution of SQL scripts.
msvcp71.dll	System DLL.
msvcr71.dll	System DLL.
qli.exe	Interactive GDML tool (out of date, no longer used).
nbackup.exe	Tool for incremental backups.

Linux bin subdirectory

SSchangeRunUser.sh	Shell script for alterating the SuperServer user.
SSrestoreRootRunUser.sh	Shell script for restoring the SuperServer user.
changeDBAPassword.sh	Shell script for altering the SYSDBA password.
createAliasDB.sh	Shell script for creating a new alias. Invoke: ./createAliasDB.sh <aliasname> <datenbankname>

fb_config	Shell script containing sundry information.
	Invoke: `./fb_config [options]`
	Options:
	`--cflags`
	`--libs`
	`--embedlibs`
	`--bindir`
	`--version`

fb_lock_print	Shell script for the output of locking information.
fbguard	Firebird Guardian demon.
fbmgr	Firebird Manager for starting and shutting down the Firebird demon.
fbmgr.bin	Firebird Manager.
fbserver	Firebird server demon.
gbak	GBAK tool: backup, restore.
gdef	GDML tool (out of date, no longer used).
gfix	GFIX tool: settings, repair, adminstration.
gpre	GPRE tool: C preprocessor.
gsec	GSEC tool: user administration.
gstat	GSTAT tool: statistics.
isql	ISQL tool: interactive execution of DDL and DML commands, execution of SQL scripts.
qli	Interactive GDML tool (out of date, no longer used).
nbackup	Tool for incremental backups.

Other subdirectories (both platforms)

doc	Documentation, release notes, readmes, etc.
examples	Sample programs and databases.
help	Online help (currently practically empty).
include	Include files for the development of C-based client applications and UDFs.
intl	International support.
lib	Library files for the development of C-based client applications and UDFs.
UDF	User-defined functions.

firebird.conf

The `firebird.conf` file, found in the Firebird root directory, can be edited in any Text Editor. Key parameters include:

`DefaultDbCachePages = 2048`	Number of cached database pages per database.
`RemoteServiceName = gds_db` `RemoteServicePort = 3050`	Name of the service in the `services` file and/or TCP port number for the service. This only needs to be altered if a Borland InterBase™ service is already running or potential confusion with InterBase is to be avoided.
`DatabaseAccess = Full`	Only accepts one of the following values: * `None`: only databases listed in `aliases.conf` may be used. * `Full` (Default): all databases may be used. * `Restrict`: only databases found in the specified paths may be used. These paths must be specified in a semicolon-separated list (on Windows e.g. `C:\DataBase;D:\Mirror`, on Unix e.g. `/db;/mnt/mirrordb`).

Recommendation: We strongly recommend this parameter be used to restrict backdoor access to the system. Uncontrolled access to all databases can seriously endanger your system security.

See also:
`firebird.conf`

Database System Administrator SYSDBA

The user, `SYSDBA` (*System Database Administrator*) has Database Administrator status. He has all permissions.

The standard password for `SYSDBA` is: `masterkey`

The `SYSDBA` password should be changed immediately following installation of a productive system.

Linux server

When installing on Linux systems a random password is generated. This can be found in the `SYSDBA.password` file in the Firebird root directory.

A new SYSDBA password can be assigned in the shell script `bin/changeDBAPassword.sh`.

Network integration TCP/IP

Following a standard installation, the Firebird service listens to port `3050/tcp`. This can be altered if wished in the `firebird.conf`. It is also usual procedure to add the following entry in the `services` file:

```
gds_db     3050/tcp
```

This specification also needs to be adjusted accordingly.

As the service name, `gsd_db`, is for InterBase databases, another service name needs to be defined if InterBase and Firebird installations are to run in parallel. This service name also needs to be specified correspondingly in the `firebird.conf` and the `services` file (our proposal: `firebirdsql`).

- Location of the `services` file in Windows: `\Windows\system32\drivers\etc\services`
- Location of the `services` file in Linux: `/etc/services`

The service or demon needs to be restarted following any alterations to `firebird.conf`.

Security

The `security.fdb` database, stored in the Firebird root directory, is responsible for user administration.

SYSDBA always has all permissions and rights. The user who created the database is the database owner and also has all permissions and rights for that database.

Users can be administrated using the GSEC tool (refer to *Administration tools*)

Administration tools

Firebird comes with a number of administration command-line tools:

isql	Command-line interactive tool for the execution of DDL and DML commands and scripts.
gbak	Backup, restore.
gfix	Various parameters, repair.
gsec	User administration.
gstat	Statistics.
fbmgr	Linux only: starts and shuts down the Firebird demon.
nbackup	Incremental backups.
instsvc	Service setup.
instreg	Registry parameters setup.

We recommend the comprehensive tool, IBExpert (http://www.ibexpert.com), which also offers a free Personal Edition, for working with Firebird. This tool is however only available for Windows.

You can also download the free "FbAdmin" from the Consic homepage. This is a simple, German-language administration program, that covers the most important administrative tasks: http://www.consic.de/firebird.

ISQL

The ISQL utility ("Interactive SQL") can be found in the Firebird installation's bin directory. When started it reports back with an SQL prompt:

```
SQL> _
```

Each command must end with a semicolon to be executed. Commands can also extend over several lines, from the second line onwards they must be preceded with CON> (*Continue*) as a prompt.

Create a database

Use the following command to create a new, empty database:

```
SQL> create database 'c:\test.fdb'
CON> user 'SYSDBA'
CON> password 'masterkey'
CON> page_size 4096
CON> default character set iso8859_1 ;
```

A page size of 4096 bytes is considered optimal for up-to-date server operating systems. The page size hast to be a multiple of 1024.

Following the database creation, you should convert to SQL Dialect 3. This can be done using the GFIX utility (detailed in a separate chapter):

```
gfix c:\test.fdb -user SYSDBA -password masterkey -sql_dialect 3
```

Connect to a database

```
SQL> connect 'c:\test.fdb' user 'SYSDBA' password 'masterkey';
```

Use this command to test to connect to a database.

Closing ISQL

ISQL can be closed using the commands, QUIT or EXIT. EXIT commits the current transaction first, QUIT rolls the current transaction back.

Executing an SQL script file

```
isql -i C:\DB\myscript.sql
```

The script file should include a CONNECT command for the database connection. Alternatively the database can be named, along with the user name and password, directly:

Starting ISQL with a direct database connection

```
isql c:\test.fdb -user SYSDBA -password masterkey
```

Determining the database SQL dialect

```
SQL> show sql dialect;
       Client SQL dialect is set to: 3 and database SQL dialect is: 3.
```

See also:
ISQL

GSEC: user administration

The users of all databases run by one service are stored in the security database, security.fdb. There is alway at least one user, the Database Administrator, SYSDBA.

Following the installation of a new service, the SYSDBA's password is set to masterkey. (Exception: Firebird for Linux, see *Installing on Linux*).

Only the first 8 characters of a Firebird password are significant. A password may not contain any spaces.

Starting GSEC

GSEC can only be started by the SYSDBA.

To start GSEC on the local server, enter:

```
gsec -user sysdba -password <password> [options]
```

To start GSEC for a server in the network, enter:

```
gsec -user sysdba -password <password> -database <databasename>
```

where <databasename> is the name of the security.fdb database on the server.

GSEC can be used as an interactive command-line tool. Alternatively the commands can also be input directly on a command line.

Commands

di[splay]	Displays all users.
di[splay] <username>	Displays all information for the specified user (excepting the password).
a[dd] <username> -pw <password> [options]	Insert a new user.
mo[dify] <username> [options]	Alters the user.
de[lete] <username>	Deletes the user.
h[elp] oder ?	Displays the help.
q[uit]	Ends the interactive mode.
z	Displays the GSEC version number.

If you do not wish to start the interactive mode, all commands may be entered directly in the command line. Each command then need to be preceded by a hyphen ("-").

Options

-pa[ssword] <password>	The password of the user carrying out the alterations.
-user <username>	The user name of the user carrying out the alterations.
-pw <password>	Password of the user being altered or new password.
-fname <first name>	First name of the user being altered.
-mname <middle name>	Middle name of the user being altered.
-lname <last name>	Last name of the user being altered.

Examples

Add the user Elvis Presley as user name, ELVIS, the password is Aaron:

```
gsec -user SYSDBA -password masterkey
GSEC> add elvis -pw Aaron -fname Elvis -lname Presley
GSEC> quit
```

Change user `ELVIS`'s password to `chuck`:

```
gsec -user SYSDBA -password masterkey
GSEC> modify elvis -pw chuck
GSEC> quit
```

On Linux, change the `SYSDBA` password from `harry` to `hamburg`:

```
gsec -user SYSDBA -password masterkey -database
-> harry:/opt/firebird/security.fdb -modify sysdba -pw hamburg
```

On Windows, change `SYSDBA`'s password from `Sally` to `hannover`:

```
gsec -user SYSDBA -password masterkey -database
-> sally:"C:\Program Files\Firebird\security.fdb"
-> -modify sysdba -pw hannover
```

Change `SYSDBA`'s password on server, `jake`, on TCP port `3051` to `london`:

```
gsec -user SYSDBA -password masterkey -database
-> jake/3051:/opt/firebird/security.fdb" -modify sysdba -pw london
```

Delete user `JOE` on the local server:

```
gsec -user SYSDBA -password masterkey -delete joe
```

See also:
`ISQL`
Security in Firebird 2
IBExpert Grant Manager
IBExpert User Manager

Databases

A database consists of a file (distribution across several files is possible). This file contains all tables, indices, user rights (Grants), foreign keys, stored procedures, triggers, etc.

Usual suffix: `.fdb`

This file must be stored on the same computer as the Firebird service itself. Access to a file server is technically impossible (regardless of whether via UNC names or a hard drive letter).

A database file will always get bigger, never smaller. The only possibility to reduce the size of a database file, is to performa a backup and restore.

Database string

In order to connect to a certain Firebird database, the client must enter the database string. This is composed of the following:

```
<servername> [/<port>] ":" <datenbank>
```

`servername`	Name of the database server in the TCP/IP network.
`port`	Port number or IP service name, if the standard port 3050 is not to be used (see also `firebird.conf`).
`datenbank`	Either the file name of the database. *Important:* This name must always be entered from the viewpoint of the database server's local file system (no clearance directory names or similar). The directory in which the database is stored must not require clearance in order to use it. *or* The name of the database alias, as defined in `aliases.conf`.

The `DatabaseAccess` parameter in `firebird.conf` determines whether file names, aliases or both may be used.

The rules regarding case sensitivity conform to the server operating system. On a Linux server case sensitivity needs to be taken in consideration, on Windows it doesn't.

Example Windows server

The database server name is `dbserver`. The default port is used. The database file is stored on `C:\DB\pmm.fdb`:

```
dbserver:C:\DB\pmm.fdb
```

Example Linux server

The database server name is `dbserver`. The default port is used. The database file is stored on `/db/pmm.fdb`:

```
dbserver:/db/pmm.fdb
```

Example port number 3051

The database server name is `dbserver`. Port 3051 is to be used. The database file is stored on `C:\DB\pmm.fdb`:

```
dbserver/3051:C:\DB\pmm.fdb
```

If the port number is to a service name in the `services` file:

```
firebirdsql 3051/tcp
```

then the service name can be used instead of the port number:

```
dbserver/firebirdsql:C:\DB\pmm.fdb
```

See also:
Configuring Firebird

Alias names

Entering the full database connection string with directory and file name is cumbersome and a potential security risk. For this reasons alias names can be defined on the server.

These can be defined in the `aliases.conf` file.

Here you can find alias specifications:

```
<aliasname> = <pfad- und dateiname>
```

Example

The database server name is `dbserver`. The default port is used. The database file is stored on `/db/pmm.fdb`, an alias name `pmm` is to be specified for the database.

`aliases.conf` definition:

```
pmm = /db/pmm.fdb
```

The database connection string is now:

```
dbserver:pmm
```

A combination with the syntax for port number or service name specification is also possible:

```
dbserver/3051:pmm
```

See also:
Configuring Firebird

Owner, permissions

The database "owner" is the user that created the database (i.e. executed the `CREATE DATABASE` command). He kann grant permissions (read, write, execute) to other users (`GRANT`). If he does not `GRANT` any other users permissions, only the owner can perform DDL und DML operations.

In addition the `SYSDBA` user always has all permissions on all databases.

Only the `SYSDBA` or database owner can perform a backup or replace an existing database by a restore.

Database configuration

Editing mode

Editing operations on the database file can be buffered. Buffering is quicker, but can be unreliable in the case of a crash. This should therefore be disabled on productive systems.

Buffering is specified in `GFIX` or using an administration tool such an IBExpert. The user must be SYSDBA or the database owner.

GFIX: general syntax

```
gfix <datenbank> -user <benutzername> -password <passwort>
   -write {sync|async}
```

Enable forced writes (no buffering)

```
gfix c:\mydb.fdb -user SYSDBA -password masterkey -write sync
```

Disable forced writes (Buffering)

```
gfix c:\mydb.fdb -user SYSDBA -password masterkey -write async
```

Database sweeps

Firebird performs a garbage collection ("sweep") at irregular intervals, cleaning up open transactions. This is necessary due to Firebird and InterBase's multi-generational architecture. This stores certain data sets in a series of generations, to allow all open transactions a consistent data view.

An automatic sweep is executed when a certain number of incomplete transactions has been reached. This number in the "sweep interval". The sweep interval can be specified at any wished number.

A sweep interval of 0 (zero) switches off automatic sweeping.

A sweep can also be executed at a specified time (e.g. at night).

Only the SYSDBA or the database owner may specify the sweep interval.

Specifying the sweep interval

Specification of the sweep interval at 20,000 transactions:

```
gfix c:\test.fdb -user SYSDBA -password masterkey -housekeeping 20000
```

Deactivating the automatic sweep

```
gfix c:\test.fdb -user SYSDBA -password masterkey -housekeeping 0
```

Forcing a sweep

```
gfix c:\test.fdb -user SYSDBA -password masterkey -sweep
```

See also:
Firebird for the database expert: Episode 4 - OAT, OIT and Sweep

SQL dialect

Firebird emanates from Borland InterBase. For legacy reasons two SQL dialects, with marginal differences, are supported (`Dialect 1`, `Dialect 3`).

Dialect 3 is the preferred choice for new databases. This offers separate datatypes for DATE, TIME and TIMESTAMP (only TIMESTAMP is a combination comprising date and time).

Databases created by the CREATE DATABASE statement however have a default dialect 1. They need to be subsequently altered to dialect 3:

```
gfix c:\test.fdb -user SYSDBA -password masterkey -sql_dialect 3
```

The current specified SQL dialect for a database can be determined using ISQL:

```
isql
SQL> connect mydb.fdb user SYSDBA password masterkey;
SQL> show sql dialect;
        Client SQL dialect is set to: 3 and database SQL dialect is: 3
```

Multi-file databases

A database can be split across multiple files. However it is not possible to specify which parts of the database are stored in which file. As the old 4 GB limit (up to and including version InterBase 6.0), we do not consider a distribution across multiple files recommendable. Therefore this is not documented here any further.

Please refer to the *InterBase 6.0 Operations Guide* or the relevant Firebird documentation.

See also:
Multi-file database

Database shutdown

A database can be in a variety of states:

- **NORMAL:** The database is active and online: the normal state, allowing you to work with the database.
- **MULTI:** Only connections from the SYSDBA and the database owner are allowed.
- **SINGLE:** Only one single connection by the SYSDBA is allowed.
- **FULL:** Exclusive shutdown: the database is completely offline, no connections are allowed. In this state the database file (.fdb file) can be accessed (e.g. copied).

GFIX can be used to start or shutdown a database to these levels. So that connected users are not simply "thrown out" for the shutdown, there are various options to specify a certain shutdown time.

Shutdown

To shut down to the next level use the GFIX option -shut, followed by the name of the level.

Using the option -force the number of seconds can be specified, that the service should wait, until all other users have disconnected. If any connections still exist following this period, they are automatically disconnected. Open transactions are rolled back.

Alternatively the options -attach can be used to specify a certain number of seconds that should be waited until all users have disconnected. Following this period if there are any users that have still not disconnected, the shutdown is aborted and an error message published.

Shutdown from NORMAL to SINGLE

```
gfix-user sysdba -password masterkey localhost:mydb -shut single -force 0
```

The -force 0 option ensures here that all users except the SYSDBA are disconnected immediately (0 seconds waiting period).

Restart

To boot up use the -online option instead of -shut. The level name needs to be specified here as well. The options -force or -attach cannot be used here, as the restart begins immediately.

```
gfix -user sysdba -password masterkey localhost:mydb -online normal
```

Backup

Firebird database backups should be performed for the following reasons:

- The database file (.fdb file) should not be backed up directly as a file, as it is not compatible with other platforms and InterBase/Firebird versions.
- Moreover the .fdb file is in an instable condition if one or more users are connected to the database (open edit access etc.). This is also an argument against backing up the database file at file level.
- No empty page areas or indices need to be stored in the backup. The backup file is therefore (usually much) smaller.
- Databases can also be repaired or reduced in size by performing a backup and restore.
- If a database needs to be ported to another platform (e.g. from a Windows server to a Linux server), it is not the database file that is ported but the backup. This is then imported to the destination server by performing a restore of the backup file.

A backup generates a backup file. This has its own file format and contains a consistent data view, because the backup extract the data as an independent transaction.

A backup can be carried out during runtime. During this time database performance may degrade, particularly if the backup runs for some time.

The usual suffix for backup files is: .fbk

Backup in productive environments

Productive systems should be backed up regularly. The .fdb backup file can be backed up using conventional file backup methods.

If the server runs through the night, the backup can be started by a scheduler (Windows: AT service, Linux: cron).

Windows

The AT command can be used to issue tasks to a Winedows NT server (NT4, 2000, XP, 2003), which should be performed at a certain specified time.

Example: A database should be backed up nightly at 4 am. Enter the following command in the Windows prompt:

```
at 04:00 /every:mo,di,mi,do,fr,sa,so /interactive

-> c:\Programme\Firebird\bin\gbak -t -user SYSDBA -password masterkey
-> harry:c:\DB\pmm.fdb k:\Backups\pmm.fbk
```

Tip: Do not run such tasks nightly between 02:00 and 03:00. When clocks are put forward to summer time in the Spring this hour does not exist at all, when changing back in the Fall, this hour occurs twice.

See also:
Backup Database
Firebird Administration

GBAK utility

Backup and restore are executed by Firebird using the GBAK utility. The GBAK utility may be installed on any computer, even on the database server itself. It can be found in the Firebird bin directory.

GBAK is a command-line tool, which means it can be easily called from batch files, shell scripts or scheduler services.

General syntax

```
GBAK <optionen> -user <benutzer> -password <passwort> <quelle> <ziel>
```

The most important general options:

–b	Backup (default; does not need to be specified explicitly).
–c	Restore (Create).
–r	Replace: an existing database is overwritten by the restore.
–user <benutzername>	Specification of the user name.
–password <passwort>	Specification of the password.

-v	Verbose: detailed log of the action currently being conducted.
-y <dateiname>	Exports all log messages into the specified file. The file may not already exist at the time GBAK starts!
-y suppress_output	No log output.
-z	Display the GBAK version number.

See also:
GBAK

Backup

The database must be named as source and the backup file named as the target. The target must be a file name in the computer file system which is executing GBAK. If no directory is explicitly named, the current directory is used.

A backup may only be performed by the SYSDBA or the database owner.

Options

-t	Transportable Backup: A backup is generated, which can be read by all InterBase/Firebird database, independent of version and platform. Recommended for all backups.
-g	Prevents garbage collection being performed during the backup.
-ignore	Checksum errors are ignored during the backup.
-m	Metadata only: Only the metadata are backed up, not the table contents.
-nt	Non-transportable format: The opposite of -t. Not recommended.
-se <hostname>:service_mgr	Uses the Service Manager. Backup: the backup file is created on the database server. Restore: the restore is made from a file which is on the database server. This option must be specified if the security2.fdb is to be backed up.

Typical backup example

```
gbak -v -t -user SYSDBA -password masterkey dbserver:pmm c:\Backup\pmm.fbk
```

-v	Verbose output.
-t	Transportable format.
-user SYSDBA	User name.
-password masterkey	Password (the password can be entered in quotes if it contains empty spaces).
dbserver:pmm	Database name (pmm is obviously an alias registered on dbserver).

Another example:

```
gbak -v -t -user SYSDBA -password masterkey joe:/db/pmm.fdb c:\backup.fbk
```

Metadata backup

```
gbak -v -t -m -user SYSDBA -password masterkey dbserver:pmm c:\backup.fbk
```

See also:
Backup Database

Restore

A restore converts a backup file into a database. The source is the backup file (.fbk file) and the target is the database name. It is possible to overwrite an existing database.

Options

-c	Restore in a new database. I.e. the database file of the new database *MUST NOT* exist, otherwise the restore is aborted and an error message appears. Mutually exclusive with -rep.
-rep	Replaces an existing database. This database may not be in use at the time of the restore! It can only be performed by the SYSDBA or the database owner. Mutually exclusive with -c.
-i	Sets all indices to inactive when restoring. The restore is quicker and indices can be activated singly or together, and recomputed by the activation.
-n	Removes all validity constraints from the metadata. This enables data to be restored which violates these constraints and otherwise could not be restored.
-o	Restores one table at a time. This can be uses to partially restore databases with corrupt table data.

-p <bytes>	Sets a different page size for the new database. The page size must be a multiple of 1024. Values > 16984 cannot be used, values < 4096 are not recommended (and not allowed in Firebird 2.1).
-use_all_space	Fills all database pages to 100% instead of the usual 80%.

Typical restore example

```
gbak -c -v -user SYSDBA -password masterkey c:\backup\pmm.fbk dbserver:pmm
```

Restore to an existing database

```
gbak -rep -v -user SYSDBA -password masterkey c:\backups\pmm.fbk
   dbserver:/db/pmm2.fdb
```

See also:
Restore Database

User database security2.fdb

All Firebird service users are stored in the user database, security2.fdb in the Firebird root directory. For a complete data backup a backup of this database should also be made. GBAK can be used for this.

The security database can however not be backed up remotely. The Service Manager has to be used. The backup file is generated physically on the database server. If it is created in a released directory, it can then be moved to another location.

For security reasons the security database and any backups of it should not be accessible to non-administrators.

A direct backup of the security2.fdb is however possible, as the Firebird service always has it open. So should you ever need to recover the security2.fdb you will need to follow the following procedure:

- You need a functional user database, so that the service can run. If necessary carry out a new installation. (Here the SYSDBA user is already set up with a password that is known.)
- Perform a restore using GBAK, however not directly overwriting the existing security2.fdb in the Firebird root directory, but somewhere else.
- Shut down the Firebird service. In Windows using the Services Manager, in Linux with the fbmgr utility.
- Replace the security2.fdb in the Firebird root directory with the file just created by the restore.
- Restart the Firebird service.

See also:
Security in Firebird 2

Links, Literature

http://www.firebirdsql.org	Home page of the Firebird project. Containing news and links to the downloads.
http://www.ibphoenix.com	Home page of a team, that is involved in the Firebird development, and provides additional information.
http://www.destructor.de/firebird	Firebird information and documentation.
http://www.ibexpert.com	IBExpert information and downloads.
http://www.consic.de/firebird	This handbook and further Firebird information and downloads.
The Firebird Book	The Firebird Book, *A Reference for Database Developers: An essential guide for developers and administrators working with the Firebird open source relational database management system*. Helen Borrie, 2004, 1092 Seiten, ISBN 1590592794

Firebird 2 Cheat Sheet

Author: Lorenzo Alberton, http://www.alberton.info

Firebird SQL Cheat Sheet - Details

The cheat sheet is organized in 5 sections. The first section contains a list of the available datatypes, their description and the range of values that each of them supports.

The second section contains a list of the internal functions. The ones listed here are the Firebird 2 built-in functions; they're grouped by field of interest (aggregate, conditional, string functions).

The third section contains a list of the Default UDF functions. Firebird bundles an UDF library with some useful functions not included in the core. These functions are listed here, grouped by field of interest (mathematical and string functions).

The fourth section contains some useful queries, like the most useful queries to manage `TRANSACTIONS`, `SAVEPOINTS`, `SEQUENCES`, a sample query with a `LIMIT / OFFSET` clause, and some queries against the System Tables to retrieve a list of the tables, fields, indices and constraints.

The last section holds a list of the PHP `ibase_*` functions. PHP has a Firebird/Interbase module and this is used by PHP developers to connect to, and query, a Firebird database. This section lists the functions available in PHP for connecting to and managing a Firebird database.

FireBird Data Types

BLOB	Variable[1]
BIGINT	Integer, 64 bits (-2×10^{63} to $2 \times 10^{63}-1$)
CHAR(n)	String, n characters (1 to 32,767 bytes[2])
DATE	Integer, 32 bits 01-01-100 to 31-12-9999
DECIMAL (precision [, scale])	Decimal (precision: 1-18, scale: 1-18) DECIMAL(8,3)=ppppp.sss
DOUBLE PRECISION	Floating point, 64 bits 2.225×10^{-308} to 1.797×10^{308}
FLOAT	Floating point, 32 bits 1.175×10^{-38} to 3.402×10^{38}
INTEGER	Integer, 32 bits, signed -2,147,483,648 to 2,147,483,647
NUMERIC (precision [, scale])	similar to DECIMAL(precision [, scale])
SMALLINT	Integer, 16 bits (-32,768 to 32,767)
TIME	Integer, 32 bits 0:00:00 to 23:59:59.9999
TIMESTAMP	Integer, 64 bits
VARCHAR(n)	String, up to n characters (0 to 32,765 bytes[2])

[1] Dynamically sizable dataype for storing large data such as graphics, text, and digitized voice. Blob subtype describes Blob contents.
[2] Charset character size determines the maximum number of characters that can fit in 32K.

Firebird®

Available on Windows, MacOS, Linux, BSD, Solaris, Unix, HP-UX

Many ways to access your database: native API, dbExpress drivers, ODBC, OLEDB, JDBC, .Net, Python, PHP, Perl...

FB internal functions

Group
COUNT, AVG, MIN, MAX, SUM

Conditional
CASE WHEN *condition* THEN *result* [WHEN ...] [ELSE *result*] END
CASE *condition* WHEN *val* THEN *result* [WHEN ...] [ELSE *result*] END
COALESCE (*value* [, ...])
IIF (*condition*, *val1*, *val2*)
NULLIF (*val1*, *value2*)

String
BIT_LENGTH (*value*)
CHAR_LENGTH (*value*)
LOWER (*value*)
OCTET_LENGTH (*value*)
SUBSTRING (*str* FROM *start* [FOR *count*])
TRIM ([LEADING | TRAILING | BOTH] [*chars* FROM] *value*)
UPPER (*value*)

Other
CAST (*value* AS *datatype*)
EXTRACT (*part* FROM *ts*)

System
CURRENT_CONNECTION
CURRENT_DATE
CURRENT_ROLE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TRANSACTION
CURRENT_USER
USER

Default UDF Functions

Mathematical
ABS (*value*)
ACOS (*value*)
ASIN (*value*)
ATAN (*value*)
ATAN2 (*val1*, *val2*)
BIN_AND (*val1*, *val2*)
BIN_OR (*val1*, *val2*)
BIN_XOR (*val1*, *val2*)
CEILING (*value*)
COS (*value*)
COSH (*value*)
COT (*value*)
DIV (*val1*, *val2*)
FLOOR (*value*)
LN (*value*)
LOG (*base*, *value*)
LOG10 (*value*)
MOD (*val1*, *val2*)
PI
RAND ()
SIGN (*value*)
SIN (*value*)
SINH (*value*)
SQRT (*value*)
TAN (*value*)
TANH (*value*)

String
ASCII_CHAR (*int*)
ASCII_VAL (*char*)
LTRIM (*value*)
STRLEN (*value*)
SUBSTR (*str*, *start*, *end*)

You can write your own UDF functions in any programming language that is compiled into a shared library

Useful queries
-- Limit query
SELECT FIRST *limit* SKIP *offset* * FROM *table_name* ...
-- List tables
SELECT RDB$RELATION_NAME FROM RDB$RELATIONS WHERE RDB$SYSTEM_FLAG=0 AND RDB$VIEW_BLR IS NULL
-- List table fields
SELECT RDB$FIELD_NAME FROM RDB$RELATION_FIELDS WHERE RDB$RELATION_NAME='*table_name*'
-- List table constraints
SELECT RDB$INDEX_NAME FROM RDB$INDICES WHERE RDB$RELATION_NAME='*table_name*' AND (RDB$UNIQUE_FLAG IS NOT NULL OR RDB$FOREIGN_KEY IS NOT NULL)
-- List table indices
SELECT RDB$INDEX_NAME FROM RDB$INDICES WHERE RDB$RELATION_NAME='*table_name*' AND RDB$UNIQUE_FLAG IS NULL AND RDB$FOREIGN_KEY IS NULL
-- Handle sequences (NB: they're called GENERATORs in FB 1.x)
• CREATE|DROP SEQUENCE *name*; • SET SEQUENCE *name* TO *value*; • SELECT GEN_ID(*name*, *increment*) FROM RDB$DATABASE; • SELECT NEXT VALUE FOR *name* FROM RDB$DATABASE;
-- Transactions
• SET TRANSACTION [READ WRITE | READ ONLY] [WAIT | NO WAIT] [ISOLATION LEVEL {SNAPSHOT [TABLE STABILITY] | READ COMMITTED [[NO] RECORD_VERSION]}] • SAVEPOINT *name*; • ROLLBACK [WORK] [TO [SAVEPOINT] *name*]; • COMMIT;

FireBird functions in PHP5	
ibase_add_user	ibase_field_info
ibase_affected_rows	ibase_free_event_handler
ibase_backup	ibase_free_query
ibase_blob_add	ibase_free_result
ibase_blob_cancel	ibase_gen_id
ibase_blob_close	ibase_maintain_db
ibase_blob_create	ibase_modify_user
ibase_blob_echo	ibase_name_result
ibase_blob_get	ibase_num_fields
ibase_blob_import	ibase_num_params
ibase_blob_info	ibase_param_info
ibase_blob_open	ibase_pconnect
ibase_close	ibase_prepare
ibase_commit_ret	ibase_query
ibase_commit	ibase_restore
ibase_connect	ibase_rollback_ret
ibase_db_info	ibase_rollback
ibase_delete_user	ibase_server_info
ibase_drop_db	ibase_service_attach
ibase_errcode	ibase_service_detach
ibase_errmsg	ibase_set_event_handler
ibase_execute	ibase_timefmt
ibase_fetch_assoc	ibase_trans
ibase_fetch_object	ibase_wait_event
ibase_fetch_row	

Available for free from **www.alberton.info**

You can download the cheat sheet here or view at Lorenzo's website: http://www.alberton.info/firebird_cheat_sheet.html

Firebird 2 SQL Reference Guide (Preview)

The complete reference of all SQL keywords and commands supported by Firebird
Members of the Firebird Documentation project
December 2007

Firebird 2 SQL Reference Guide

Introduction

The Firebird SQL Reference Guide contains an alphabetical index of all keywords and built-in functions available in a Firebird database.

Note that not all terms are available everywhere. At the start of every entry there is an item *Availability* that tells in what context(s) a keyword or function can be used. The terms used there are described in the following.

DSQL

Dynamic SQL is the context of a SQL client (application) sending SQL commands to the server.

ESQL

Embedded SQL is the context of a SQL command embedded in an application. This is in essence the same as DSQL, except that every ESQL statement must be preceeded with the `EXEC` SQL keyword.

ISQL

ISQL (or Interactive SQL) is a command line tool that is included in the Firebird distribution. It allows access to (almost) the full feature set available in Firebird, and is the recommended tool to narrow down the source of a potential problem with a SQL command should you find one. Unlike most other connectivity components and tools, ISQL shows also warning messages that may not be shown.

PSQL

PSQL (or Procedural SQL) is the SQL context used in Stored Procedures and Triggers. There are some special commands and keywords only available in PSQL, like the `NEW` and `OLD` context variables in triggers. But there are also some limitations against D/E/ISQL: as a rule of thumb, PSQL is limited to DML (Data Manipulation Language), while the other flavours also allow DDL (Data Definition Language) statements.

Firebird 2 SQL Reference Guide

Introduction

The Firebird SQL Reference Guide contains an alphabetical index of all keywords and built-in functions available in a Firebird database.

Note that not all terms are available everywhere. At the start of every entry there is an item *Availability* that tells in what context(s) a keyword or function can be used. The terms used there are described in the following.

DSQL

Dynamic SQL is the context of a SQL client (application) sending SQL commands to the server.

ESQL

Embedded SQL is the context of a SQL command embedded in an application. This is in essence the same as DSQL, except that every ESQL statement must be preceeded with the EXEC SQL keyword.

ISQL

ISQL (or Interactive SQL) is a command line tool that is included in the Firebird distribution. It allows access to (almost) the full feature set available in Firebird, and is the recommended tool to narrow down the source of a potential problem with a SQL command should you find one. Unlike most other connectivity components and tools, ISQL shows also warning messages that may not be shown.

PSQL

PSQL (or Procedural SQL) is the SQL context used in Stored Procedures and Triggers. There are some special commands and keywords only available in PSQL, like the NEW and OLD context variables in triggers. But there are also some limitations against D/E/ISQL: as a rule of thumb, PSQL is limited to DML (Data Manipulation Language), while the other flavours also allow DDL (Data Definition Language) statements.

Firebird 2 SQL Reference Guide

Introduction

The Firebird SQL Reference Guide contains an alphabetical index of all keywords and built-in functions available in a Firebird database.

Note that not all terms are available everywhere. At the start of every entry there is an item *Availability* that tells in what context(s) a keyword or function can be used. The terms used there are described in the following.

DSQL

Dynamic SQL is the context of a SQL client (application) sending SQL commands to the server.

ESQL

Embedded SQL is the context of a SQL command embedded in an application. This is in essence the same as DSQL, except that every ESQL statement must be preceeded with the EXEC SQL keyword.

ISQL

ISQL (or Interactive SQL) is a command line tool that is included in the Firebird distribution. It allows access to (almost) the full feature set available in Firebird, and is the recommended tool to narrow down the source of a potential problem with a SQL command should you find one. Unlike most other connectivity components and tools, ISQL shows also warning messages that may not be shown.

PSQL

PSQL (or Procedural SQL) is the SQL context used in Stored Procedures and Triggers. There are some special commands and keywords only available in PSQL, like the NEW and OLD context variables in triggers. But there are also some limitations against D/E/ISQL: as a rule of thumb, PSQL is limited to DML (Data Manipulation Language), while the other flavours also allow DDL (Data Definition Language) statements.

Firebird 2 SQL Reference Guide

Introduction

The Firebird SQL Reference Guide contains an alphabetical index of all keywords and built-in functions available in a Firebird database.

Note that not all terms are available everywhere. At the start of every entry there is an item *Availability* that tells in what context(s) a keyword or function can be used. The terms used there are described in the following.

DSQL

Dynamic SQL is the context of a SQL client (application) sending SQL commands to the server.

ESQL

Embedded SQL is the context of a SQL command embedded in an application. This is in essence the same as DSQL, except that every ESQL statement must be preceeded with the EXEC SQL keyword.

ISQL

ISQL (or Interactive SQL) is a command line tool that is included in the Firebird distribution. It allows access to (almost) the full feature set available in Firebird, and is the recommended tool to narrow down the source of a potential problem with a SQL command should you find one. Unlike most other connectivity components and tools, ISQL shows also warning messages that may not be shown.

PSQL

PSQL (or Procedural SQL) is the SQL context used in Stored Procedures and Triggers. There are some special commands and keywords only available in PSQL, like the NEW and OLD context variables in triggers. But there are also some limitations against D/E/ISQL: as a rule of thumb, PSQL is limited to DML (Data Manipulation Language), while the other flavours also allow DDL (Data Definition Language) statements.

Firebird 2 SQL Reference Guide

Introduction

The Firebird SQL Reference Guide contains an alphabetical index of all keywords and built-in functions available in a Firebird database.

Note that not all terms are available everywhere. At the start of every entry there is an item *Availability* that tells in what context(s) a keyword or function can be used. The terms used there are described in the following.

DSQL

Dynamic SQL is the context of a SQL client (application) sending SQL commands to the server.

ESQL

Embedded SQL is the context of a SQL command embedded in an application. This is in essence the same as DSQL, except that every ESQL statement must be preceeded with the EXEC SQL keyword.

ISQL

ISQL (or Interactive SQL) is a command line tool that is included in the Firebird distribution. It allows access to (almost) the full feature set available in Firebird, and is the recommended tool to narrow down the source of a potential problem with a SQL command should you find one. Unlike most other connectivity components and tools, ISQL shows also warning messages that may not be shown.

PSQL

PSQL (or Procedural SQL) is the SQL context used in Stored Procedures and Triggers. There are some special commands and keywords only available in PSQL, like the NEW and OLD context variables in triggers. But there are also some limitations against D/E/ISQL: as a rule of thumb, PSQL is limited to DML (Data Manipulation Language), while the other flavours also allow DDL (Data Definition Language) statements.

ABS( ) [2.1]

Returns the absolute value of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ABS(<numeric expression>)
```

Argument	Description
`<number expression>`	The numeric expression whose absolute value is returned.

Description

Returns the absolute value of a number. The result is always `>= 0`.

Examples

```
select abs(amount) from transactions
select abs(4-7) from rdb$database
(returns 3)
select abs(NULL) from rdb$database
(returns NULL)
```

See also:
`SIGN()`

ABS( ) [2.1]

Returns the absolute value of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ABS(<numeric expression>)
```

Argument	Description
`<number expression>`	The numeric expression whose absolute value is returned.

Description

Returns the absolute value of a number. The result is always `>= 0`.

Examples

```
select abs(amount) from transactions
select abs(4-7) from rdb$database
(returns 3)
select abs(NULL) from rdb$database
(returns NULL)
```

See also:
`SIGN()`

ACOS() [2.1]

Returns the arc cosine of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ACOS(<numeric expression>)
```

Important: The argument to ACOS must be in the range -1 to 1.

Argument	Description
`<number expression>`	The numeric expression whose arc cosine is returned.

Description

Returns the arc cosine of a number. Argument to ACOS must be in the range -1 to 1. Returns a value in the range 0 to PI.

Examples

```
select acos(x) from y
```

See also:
COS()
SIN()

ALTER DATABASE

Adds secondary files to the current database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also: the *Data Definition Guide* for more information about multifile databases and the *Operations Guide* for more information about exclusive database access.

See also:
CREATE DATABASE
DROP DATABASE

ALTER DATABASE BEGIN/END BACKUP [2.0]

(no contents yet)

ALTER DOMAIN

Changes a domain definition.

(Syntax currently not included because of possible copyright issues.)

For a complete discussion of creating domains, and using them to create column definitions, refer to Firebird domains in *Using Firebird-Domains and Generators* (ch. 15 p. 285).

See also:
CREATE DOMAIN
CREATE TABLE
DROP DOMAIN

ALTER EXCEPTION

Changes the message associated with an existing exception.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ALTER EXCEPTION name 'message'
```

Argument	Description
`Description name`	Name of an existing exception message.
`'message'`	Quoted string containing ASCII values.

For more information on creating, raising, and handling exceptions, refer to *Using Firebird- Error trapping and handling*. (ch. 25 p. 549).

See also:
ALTER PROCEDURE
ALTER TRIGGER
CREATE EXCEPTION
CREATE PROCEDURE
CREATE TRIGGER
DROP EXCEPTION

ALTER EXTERNAL FUNCTION [2.0]

(no contents yet)

ALTER INDEX

Activates or deactivates an index.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
ALTER TABLE
CREATE INDEX
DROP INDEX
SET STATISTICS

ALTER PROCEDURE

Changes the definition of an existing stored procedure.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also: CREATE PROCEDURE for a complete description.

Terminator

Argument	Description
terminator	Terminator - defined by the ISQL SET TERM command to signify the end of the procedure body, required by ISQL.

Syntax

```
SET TERM <new terminator> <old terminator>
```

The <old terminator> is not part of the command, but the command terminator. Because SET TERM is exclusively an ISQL command, the command terminator is always required. A procedure can be altered by its creator, the SYSDBA user and, on Linux/UNIX, the root user and any user with root privileges.

Procedures in use are not altered until they are no longer in use.

ALTER PROCEDURE changes take effect when they are committed. Changes are then reflected in all applications that use the procedure without recompiling or relinking.

For more information on creating and using procedures, see *Using Firebird- Programming on Firebird Server* (ch. 25 p. 494). For a complete description of the statements in procedure and trigger language, refer to *PSQL-Firebird Procedural Language*.

See also:
CREATE PROCEDURE
DROP PROCEDURE
EXECUTE PROCEDURE

ALTER SEQUENCE .. RESTART WITH [2.0]

Sets the current value of a sequence / generator.

Availability: +DSQL +ESQL +ISQL -PSQL

Syntax

```
ALTER SEQUENCE <name> RESTART WITH <start_value>
```

Important: ALTER SEQUENCE, like SET GENERATOR, is a good way to screw up the generation of key values! It is important to know that sequences and generators are outside of any transaction control.

Argument	Description
<name>	Name of the sequence / generator to be set.
<start_value>	New starting value for the sequence / generator.

Description

This is the SQL-99-compliant (and therefor recommended) syntax for the SET GENERATOR command. It directly sets a sequence / generator to the given value.

The command is not available in -PSQL since it is a DDL and not a DML statement (this can, however, be surpassed by the use of EXECUTE STATEMENT).

This command is useful to reset e.g. an ID-generating sequence after a DELETE FROM <table>, but in almost all other circumstances it is a dangerous thing to do.

Read the *Generator Guide* which is available as part of the Firebird documentation set for an in-depth discussion of the use of sequences / generators, and esp. why it is dangerous and not recommended to use this statement in live databases.

Examples

```
ALTER SEQUENCE SEQ_ID_EMPLOYEE RESTART WITH 1;
```

(equivalent to SET GENERATOR SEQ_ID_EMPLOYEE TO 1)

See also:
SET GENERATOR
CREATE SEQUENCE
DROP SEQUENCE
NEXT VALUE FOR

ALTER TABLE

Changes a table by adding, dropping, or modifying columns or integrity constraints.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about altering tables, see *Using Firebird- Altering tables* (ch. 17 p. 340).

See also:
ALTER DOMAIN
CREATE DOMAIN
CREATE TABLE

ALTER TRIGGER

Changes an existing trigger.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For a complete description of the statements in procedure and trigger language, *PSQL-Firebird Procedural Language*. For more information, see *Using Firebird- Triggers* (ch. 25 p. 532).

See also:
CREATE TRIGGER
DROP TRIGGER

ASCII_CHAR() [2.1]

Returns the ASCII character with the specified code.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ASCII_CHAR(<numeric expression>)
```

Important: The argument to `ASCII_CHAR` must be in the range `0` to `255`.

Argument	Description
`<numeric expression>`	The code for the ASCII character to be returned.

Description

Returns the ASCII character with the specified code. The argument to `ASCII_CHAR` must be in the range `0` to `255`. The result is returned in character set `NONE`.

Examples

1. DSQL

```
select ascii_char(65) from rdb$database
```

(returns 'A')

2. PSQL

```
mystr = mystr || ascii_char(13) || ascii_char(10);
```

(adds a Carriage Return + Line Feed to `mystr`)

3. PSQL

The following selectable procedure returns the alphabet in upper and lower case:

```
CREATE PROCEDURE ALPHABET
returns (ALPHA_UPPER char(26), ALPHA_LOWER char(26))
AS
declare variable i integer;
begin
ALPHA_UPPER = '';
ALPHA_LOWER = ''; i = 0;

while (i < 26) do
begin
ALPHA_UPPER = TRIM(ALPHA_UPPER) || ASCII_CHAR(i + 65);
ALPHA_LOWER = TRIM(ALPHA_LOWER) || ASCII_CHAR(i + 65 + (ASCII_VAL('a')-
ASCII_VAL('A')));

i = i + 1;
end

suspend;
end
```

See also:
`ASCII_VAL()`

ASCII_VAL() [2.1]

Returns the ASCII code of the first character of the specified string.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ASCII_VAL(<val>)
```

Important: if `<val>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<val>`	A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype.

Description

Returns the ASCII code of the first character of the specified string.

Rules

1. Returns `0` if the string is empty.
2. Throws an error if the first character is multi-byte.
3. Returns `NULL` if `<val>` is (or evaluates to) `NULL`.

Examples

```
select ascii_val(x) from y

select ascii_val('A') from rdb$database (returns 65)
```

See also:
`ASCII_CHAR()`

ASIN() [2.1]

Returns the arc sine of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ASIN(<number>)
```

Important: The argument to `ASIN` must be in the range `-1` to `1`.

Argument	Description
`<number>`	The number or numeric expression whose arc sine is returned.

Description

Returns the arc sine of a number. Argument to `ASIN` must be in the range `-1` to `1`. Returns a value in the range `-PI/2` to `PI/2`.

Examples

```
select asin(-1) from rdb$database
(returns 1,5707963267949 = -PI/2)

select asin(0) from rdb$database
(returns 0)

select asin(1) from rdb$database
(returns 1,5707963267949 = PI/2)
```

See also:
`COS()`
`SIN()`

ATAN( ) [2.1]

Returns the arc tangent of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ATAN(<number>)
```

Important: The argument to `ATAN` must be in the range `-1` to `1`.

Argument	Description
`<number>`	The number or numeric expression whose arc tangent is returned.

Description

Returns the arc sine of a number. Argument to `ATAN` must be in the range `-1` to `1`. Returns a value in the range `-PI/2` to `PI/2`.

Examples

```
select atan(-1) from rdb$database
(returns -0,7853981633974 = -PI/4)

select atan(0) from rdb$database
(returns 0)

select atan(1) from rdb$database
(returns 0,7853981633974 = PI/4)
```

See also:
`COS()`
`SIN()`

ATAN2() [2.1]

Returns the arc tangent of the first number / the second number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ATAN2(<number1>,<number2>)
```

Important: The arguments to `ATAN2` must be in the range `-1` to `1`.

Argument	Description
`<number1>`	The first numeric expression whose arc tangent is returned.
`<number2>`	The second numeric expression whose arc tangent is returned.

Description

Returns the arc tangent of the first number / the second number. Returns a value in the range `-PI` to `PI`.

Examples

```
select atan2(1,1) from rdb$database
(returns 0,7853981633974 = PI/4)

select atan2(0,0) from rdb$database
(returns 0)
```

See also:
`COS()`
`SIN()`

AVG( )

Calculates the average of numeric values in a specified column or expression.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`COUNT( )`
`MAX( )`
`MIN( )`
`SUM( )`

BASED ON

Declares a host-language variable based on a column.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`BEGIN DECLARE SECTION`
`CREATE TABLE`
`END DECLARE SECTION`

BEGIN DECLARE SECTION

Identifies the start of a host-language variable declaration section.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`BASED ON`
`END DECLARE SECTION`

BIN_AND() [2.1]

Returns the result of a binary *and* operation performed on all arguments.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
BIN_AND( <number>[, <number> ...] )
```

Argument	Description
`<number>`	The numbers that the binary `AND` operation is executed on.

Examples

```
SELECT bin_and(1,3,7) from rdb$database
```
(returns `1`)

```
SELECT bin_and(2,6,10) from rdb$database
```
(returns `2`)

See also:
`BIN_OR()`
`BIN_XOR()`

BIN_OR() [2.1]

Returns the result of a binary *or* operation performed on all arguments.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
BIN_OR( <number>[, <number> ...] )
```

Argument	Description
`<number>`	The numbers that the binary `OR` operation is executed on.

Description

Examples

```
SELECT bin_and(1,3,7) from rdb$database
```
(returns `7`)

```
SELECT bin_or(2,6,10) from rdb$database
```
(returns `14`)

See also:
`BIN_AND()`
`BIN_XOR()`

BIN_SHL() [2.1]

Returns the result of a binary shift left operation performed on the arguments (`first << second`).

Availability: DSQL ESQL ISQL PSQL

Syntax

```
BIN_SHL( <number1>,<number2> )
```

Important: `<number2>` must be `>= 0`.

Argument	Description
`<number1>`	The number that gets binary shifted left.
`<number2>`	How many bits to shift `<number1>` left.

Examples

```
SELECT bin_shl(16,1) from rdb$database
```
(returns `32`)

```
SELECT bin_shl(16,4) from rdb$database
```
(returns `256`)

See also:
`BIN_SHR()`

BIN_SHR() [2.1]

Returns the result of a binary shift right operation performed on the arguments (`first >> second`).

Availability: DSQL ESQL ISQL PSQL

Syntax

```
BIN_SHL( <number1>,<number2> )
```

Important: `<number2>` must be `>= 0`.

Argument	Description
`<number1>`	The number that gets binary shifted right.
`<number2>`	How many bits to shift `<number1>` right.

Description

Examples

```
SELECT bin_shr(16,1) from rdb$database
```
(returns `8`)

```
SELECT bin_shr(16,4) from rdb$database
```
(returns `1`)

```
SELECT bin_shr(16,8) from rdb$database
```
(returns `0`)

See also:
`BIN_SHL()`

BIN_XOR() [2.1]

Returns the result of a binary XOR operation performed on all arguments.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
BIN_OR( <number>[, <number> ...] )
```

Argument	Description
`<number>`	The numbers that the binary `XOR` operation is executed on.

Examples

```
SELECT bin_xor(1,3,7) from rdb$database
(returns 5)
SELECT bin_xor(2,6,10) from rdb$database
(returns 14)
```

See also:
`BIN_AND()`
`BIN_OR()`

BIT_LENGTH / CHAR_LENGTH / CHARACTER_LENGTH / OCTET_LENGTH [2.0]

These functions will return information about the size of strings.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
BIT_LENGTH(<val>)
CHAR_LENGTH(<val>)
CHARACTER_LENGTH(<val>)
OCTET_LENGTH(<val>)
```

Important

If no `TRIM()` is applied to `<val>`, trailing blanks in `<val>` will add to the result (see example).

Argument	Description
`<val>`	A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype.

Description

These three new functions will return information about the size of strings:

1. `BIT_LENGTH` returns the length of a string in bits.
2. `CHAR_LENGTH/CHARACTER_LENGTH` returns the length of a string in characters.
3. `OCTET_LENGTH` returns the length of a string in bytes.

Examples

```
select
rdb$relation_name,
char_length(rdb$relation_name),
bit_length(trim(rdb$relation_name)),
char_length(trim(rdb$relation_name))
octet_length(trim(rdb$relation_name))
from rdb$relations;
```

CASE [1.5]

Allows the result of a column to be determined by the outcome of a group of exclusive conditions.

Availability: DSQL ESQL ISQL PSQL

Syntax

simple CASE:

```
CASE <search expression>
WHEN <value expression> THEN <result expression>
{ WHEN <value expression> THEN <result expression> }
[ ELSE <result expression> ]
```

searched CASE:

```
CASE
WHEN <search condition> THEN <result expression>
{ WHEN <search condition> THEN <result expression> }
[ ELSE <result expression> ]
```

Argument	Description
`<search expression>`	The expression to be examined by the CASE construct.
`<value expression>`	a constant for this CASE branch.
`<search condition>`	an expression that, if it evaluates to TRUE, gives the result in this WHEN branch.
`<result expression>`	the result returned when this WHEN or ELSE branch matches.

Description

Allow the result of a column to be determined by the outcome of a group of exclusive conditions. There are two variations of the CASE construct: simple and searched.

In the simple CASE, an expression following the keyword CASE is evaluated and compared against the various values in the simple WHEN clauses. The result given after THEN in the first matching WHEN argument is returned.

In the searched CASE, every WHEN clause holds an expression that gets evaluated. The result will be the argument following the WHEN clause for the first WHEN clause that evaluates to true.

There are three more variations to CASE:

- NULLIF is equivalent to CASE WHEN V1 = V2 THEN NULL ELSE V1 END.
- COALESCE is equivalent to CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END.
- DECODE is an inline version of CASE implemented as a function call.

Examples

Simple example:

```
SELECT
o.ID,
o.Description,
CASE o.Status
WHEN 1 THEN 'confirmed'
WHEN 2 THEN 'in production'
WHEN 3 THEN 'ready'
WHEN 4 THEN 'shipped'
ELSE 'unknown status  || o.Status || '
END
FROM Orders o;
Searched example:
SELECT
o.ID,
o.Description,
CASE
WHEN (o.Status IS NULL) THEN 'new'
WHEN (o.Status = 1) THEN 'confirmed'
WHEN (o.Status = 3) THEN 'in production'
WHEN (o.Status = 4) THEN 'ready'
WHEN (o.Status = 5) THEN 'shipped'
ELSE 'unknown status  || o.Status || '
END
FROM Orders o;
```

See also:
COALESCE()
NULLIF()
DECODE()
IF()

CAST( )

Converts a column from one datatype to another.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
UPPER( )
Firebird 2.0.4. Release Notes: CAST() behaviour improved
ibec_Cast

CEIL() / CEILING() [2.1]

Returns a value representing the smallest integer that is greater than or equal to the input argument.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
{ CEIL | CEILING }( <number> )
```

Argument	Description
`<number>`	The number whose next-greater integer value is returned.

Description

Returns a value representing the smallest integer that is greater than or equal to the input argument.

Examples

```
select ceil(1.0) from rdb$database
```
(returns 1)

```
select ceil(1.1) from rdb$database
```
(returns 2)

```
select ceil(-1.1) from rdb$database
```
(returns -1)

See also:
FLOOR()
ROUND()

CLOSE

Closes an open cursor.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CLOSE (BLOB)
COMMIT
DECLARE CURSOR
FETCH
OPEN
ROLLBACK

CLOSE (BLOB)

Terminates a specified blob cursor and releases associated system resources.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
DECLARE CURSOR (BLOB)
FETCH (BLOB)
INSERT CURSOR (BLOB)
OPEN (BLOB)

COALESCE [1.5]

a shortcut for a CASE construct returning the first non-NULL value.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
COALESCE ( <value expression> { , <value expression> } )
```

Argument	Description
<value expression>	an expression to be evaluated.

Description

Allows a column value to be calculated by a number of expressions, from which the first expression to return a non-NULL value is returned as the output value.

- COALESCE (V1, V2) is equivalent to the following case specification: CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END
- COALESCE (V1, V2,..., Vn), for n >= 3, is equivalent to the following case specification: CASE WHEN V1 IS NOT NULL THEN V1 ELSE COALESCE (V2,...,Vn) END

Examples

```
SELECT
PROJ_NAME AS Projectname,
COALESCE(e.FULL_NAME,'[< not assigned >]') AS Employeename
FROM
PROJECT p
LEFT JOIN EMPLOYEE e
ON (e.EMP_NO = p.TEAM_LEADER);
SELECT
COALESCE(Phone,MobilePhone,'Unknown') AS "Phonenumber"
FROM
Relations;
```

See also:
CASE
NULLIF()
DECODE()
IIF()

COLLATE (BLOB) [2.0]

(no contents yet)

COLLATE [PSQL] [2.1]

(no contents yet)

COMMENT [2.0]

Allows specification of comments on database metadata.

Availability: +DSQL +ESQL +ISQL -PSQL

Syntax

```
COMMENT ON DATABASE IS ( <comment> | NULL )
COMMENT ON COLUMN <tblviewname>.<fieldname> IS ( <comment> | NULL )
COMMENT ON PARAMETER <procname>.<paramname> IS ( <comment> | NULL )
COMMENT ON <basic_type> <name> IS ( <comment> | NULL )
```

Important

An empty literal string `''` will act as `NULL`.

Argument	Description
`<comment>`	the comment: a literal string constant (not an expression!).
`<tblviewname>`	name of a table or view.
`<fieldname>`	name of a column? in a table or view.
`<procname>`	name of a stored procedure.
`<paramname>`	name of a parameter of a stored procedure.
`<basic_type>`	can be `DOMAIN`, `TABLE`, `VIEW`, `PROCEDURE`, `TRIGGER`, `EXTERNAL FUNCTION`, `FILTER`, `EXCEPTION`, `GENERATOR`, `SEQUENCE`, `INDEX`, `ROLE`, `CHARACTER SET` or `COLLATION`.
`<name>`	name of a metadata object of type `<basic_type>`.

Description

This command provides a way to set the `RDB$DESCRIPTION` field in all of the `RDB$` system tables using a SQL command - that is, without the need to directly update the `RDB$` tables (which is not recommended). It allows you to comment or document any metadata object in a database.

Examples

```
COMMENT ON DATABASE IS 'This is a Firebird database';
SELECT RDB$DESCRIPTION FROM RDB$DATABASE;

COMMENT ON SEQUENCE SEQ_ID_LOG IS 'generates new IDs for the LOG table';
SELECT RDB$DESCRIPTION FROM RDB$GENERATORS
WHERE RDB$GENERATOR_NAME='SEQ_ID_LOG';

COMMENT ON COLUMN LOG.ID IS 'primary key of the LOG table';
SELECT RDB$DESCRIPTION FROM RDB$RELATION_FIELDS
WHERE RDB$RELATION_NAME='LOG' AND RDB$FIELD_NAME='ID';
```

See also:
RDB$ system tables

COMMIT

Makes a transaction's changes to the database permanent, and ends the transaction.

Availability: DSQL ESQL ISQL PSQL

(Syntax is currently not included because of possible copyright issues.)

For more information about handling transactions, see *Using Firebird - Transactions in Firebird* (ch. 8 p. 90).

See also:
Data transaction `COMMIT`
`DISCONNECT`
`ROLLBACK`

CONNECT

Attaches to one or more databases.

Availability: DSQL ESQL ISQL* PSQL
. *A subset of `CONNECT` options is available in ISQL.

(Syntax currently not included because of possible copyright issues.)

Also refer to *Using Firebird - Configuring the database cache* (ch. 5 p. 67) for more information about cache buffers and *Managing Security* in ch. 22 of the same volume for more information about database security.

See also:
DISCONNECT
SET DATABASE
SET NAMES

COS( ) [2.1]

Returns the cosine of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
COS(<number>)
```

Important

If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number or numeric expression whose cosine is returned.

Description

Returns the cosine of a number. The angle is specified in radians and returns a value in the range `-1` to `1`.

Examples

```
select cos(0) from rdb$database
(returns 1)

select cos(-1) from rdb$database
(returns 0,5403023058681)

select cos(1) from rdb$database
(returns 0,5403023058681)
```

See also:
`SIN()`

COSH() [2.1]

Returns the hyperbolic cosine of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
COSH(<number>)
```

Important

If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number or numeric expression whose hyperbolic cosine is returned.

Description

Returns the hyperbolic cosine of a number. The angle is specified in radians and returns a value in the range `-1` to `1`.

Examples

```
select cosh(0) from rdb$database
(returns 1)

select cosh(-1) from rdb$database
(returns 1,5430806348152)

select cosh(1) from rdb$database
(returns 1,5430806348152)
```

See also:
`SIN()`
`COS()`

COT() [2.1]

Returns Returns 1 / tan(argument).

Availability: DSQL ESQL ISQL PSQL

Syntax

```
COT(<number>)
```

Important

If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number or numeric expression whose cotangent is returned.

Description

Returns the cotangent of a number. The angle is specified in radians and returns a value in the range `-1` to `1`.

Examples

```
select cot (0) from rdb$database
(returns INF)

select cot(-1) from rdb$database
(returns -0,6420926159343)

select cot(1) from rdb$database
(returns 0,6420926159343)
```

See also:
SIN()

COUNT( )

Calculates the number of rows that satisfy a query's search condition.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
AVG( )
MAX( )
MIN( )
SUM( )

CREATE COLLATION [2.1]

(no contents yet)

CREATE DATABASE

Creates a new database.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
CREATE {DATABASE | SCHEMA} 'filespec'
    [USER 'username' [PASSWORD 'password']]
    [PAGE_SIZE [=] int]
    [LENGTH [=] int [PAGE[S]]]
    [DEFAULT CHARACTER SET charset]
    [<secondary_file>];
<secondary_file> = FILE 'filespec' [<fileinfo>] [< secondary_file>]
<fileinfo> = {[LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int }
    [<fileinfo>]
```

Important

In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in ISQL, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	Description
`'filespec'`	A new database file specification; file naming conventions are platformspecific. See *Creating a database* for details about database file specification.
`USER 'username'`	Checks the username against valid user name and password combinations in the security database on the server where the database will reside. * Windows client applications must provide a user name on attachment to a server. * Any client application attaching to a database on NT or NetWare must provide a user name on attachment.
`PASSWORD 'password'`	Checks the password against valid user name and password combinations in the security database on the server where the database will reside; can be up to 8 characters. * Windows client applications must provide a user name and password on attachment to a server. * Any client application attaching to a database on NT or NetWare must provide a password on attachment.
`PAGE_SIZE [=] int`	Size, in bytes, for database pages. int can be 1024, 2048, 4096 (default), 8192 or 16384. From Firebird 2.1 onward, 1024 and 2048 can not be used any more.
`DEFAULT CHARACTER SET charset`	Sets the default character set for a database charset is the name of a character set; if omitted, character set defaults to `NONE`.
`FILE 'filespec'`	Names one or more secondary files to hold database pages after the primary file is filled. For databases created on remote servers, secondary file specifications cannot include a node name.
`STARTING [AT [PAGE]] int`	Specifies the starting page number for a secondary file.
`LENGTH [=] int [PAGE [S]]`	Specifies the length of a primary or secondary database file. Use for primary file only if defining a secondary file in the same statement.

Description

`CREATE DATABASE` creates a new, empty database and establishes the following characteristics for it:

- The name of the primary file that identifies the database for users. By default, databases are contained in single files.
- The name of any in which the database is stored. A database can reside in more than one disk file if additional file names are specified as secondary files. If a database is created on a remote server, secondary file specifications cannot include a node name.
- The size of database pages. Increasing page size can improve performance for the following reasons:
  - Indexes work faster because the depth of the index is kept to a minimum.
  - Keeping large rows on a single page is more efficient.
  - Blob data is stored and retrieved more efficiently when it fits on a single page.

If most transactions involve only a few rows of data, a smaller page size might be appropriate, since less data needs to be passed back and forth and less memory is used by the disk cache.

- The number of pages in each database file.
- The dialect of the database.The initial dialect of the database is the dialect of the client that creates it. For example, if you are using ISQL, either start it with the `-sql_dialect n` switch or issue the `SET SQL DIALECT n` command before issuing the `CREATE DATABASE` command. Typically, you would create all databases in dialect 3. Dialect 1 exists to ease the migration of legacy databases.

Note: To change the dialect of a database, use the gfix tool.

- The character set used by the database.

For a list of the character sets recognized by Firebird, see *Character sets and collations available in Firebird*. Choice of `DEFAULT CHARACTER SET` limits possible collation orders to a subset of all available collation orders. Given a specific character set, a specific collation order can be specified when data is selected, inserted, or updated in a column.

If you do not specify a default character set, the character set defaults to `NONE`. Using character set `NONE` means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it.

You can load any character set into a column defined with `NONE`, but you cannot load that same data into another column that has been defined with a different character set. In that case, no transliteration is performed between the source and destination character sets, and transliteration errors may occur during assignment.

- System tables that describe the structure of the database. After creating the database, you define its tables, views, indexes, and system views as well as any triggers, generators, stored procedures, and UDFs that you need.

Important

In DSQL, you must execute `CREATE DATABASE EXECUTE IMMEDIATE`. The database handle and transaction name, if present, must be initialized to zero prior to use.

Read-only databases: Databases are always created in read-write mode. You can change a database to read-only mode in either of two ways: You can specify mode `-read_only` when you restore a backup or you can use gfix `-mode read_only` to change the mode of a read-write database to read-only.

About file sizes: Firebird dynamically expands the last file in a database as needed until it reaches the filesystem limit for shared access files. This applies to single-file database as well as to the last file of multifile databases. It is important to be aware of the maximum size allowed for shared access files in the filesystem environment where your databases live. Firebird database files are limited to 2GB in many environments. The total file size is the product of the number of database pages times the page size. The default page size is 4KB and the maximum page size is 16KB. However, Firebird files are small at creation time and increase in size as needed. The product of number of pages times page size represents a potential maximum size, not the size at creation.

Examples

The following ISQL statement creates a database in the default directory using ISQL:

```
CREATE DATABASE 'employee.gdb';
```

The next ESQL statement creates a database with a page size of 2048 bytes rather than the default of 4096:

```
EXEC SQL
CREATE DATABASE 'employee.gdb' PAGE_SIZE 2048;
```

The following ESQL statement creates a database stored in two files and specifies its default character set:

```
EXEC SQL
CREATE DATABASE 'employee.gdb'
DEFAULT CHARACTER SET ISO8859_1
FILE 'employee2.gdb' STARTING AT PAGE 10001;
```

See also:
`ALTER DATABASE`
`DROP DATABASE`

CREATE DOMAIN

Creates a column definition that is global to the database.

Availability: DSQL ESQL ISQL PSQL

(The syntax is currently not included because of possible copyright issues.)

Note 1: Be careful not to create a domain with contradictory constraints, such as declaring a domain NOT NULL and assigning it a DEFAULT value of NULL. The datatype specification for a CHAR or VARCHAR text domain definition can include a CHARACTER SET clause to specify a character set for the domain. Otherwise, the domain uses the default database character set.

For a complete list of character sets recognized by Firebird, see chapter 4, *Character Sets and Collation Orders* (p. 249). If you do not specify a default character set, the character set defaults to NONE. Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it. You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a different character set. In these cases, no transliteration is performed between the source and destination character sets, so errors can occur during assignment.

The COLLATE clause enables specification of a particular collation order for CHAR, VARCHAR, and NCHAR text datatypes. Choice of collation order is restricted to those supported for the domain's given character set, which is either the default character set for the entire database, or a different set defined in the CHARACTER SET clause as part of the datatype definition. For a complete list of collation orders recognized by Firebird, see chapter 4, *Character Sets and Collation Orders* (p. 249).

Columns based on a domain definition inherit all characteristics of the domain. The domain default, collation clause, and NOT NULL setting can be overridden when defining a column based on a domain. A column based on a domain can add additional CHECK constraints to the domain CHECK constraint.

See also:
ALTER DOMAIN
ALTER TABLE
CREATE TABLE
DROP DOMAIN

CREATE EXCEPTION

Creates a used-defined error and associated message for use in stored procedures and triggers.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information on creating, raising, and handling exceptions, see the *Using Firebird- Error trapping and handling* (ch. 25 p. 549).

See also:
ALTER EXCEPTION
ALTER PROCEDURE
ALTER TRIGGER
CREATE PROCEDURE
CREATE TRIGGER
DROP EXCEPTION

CREATE GENERATOR

Declares a generator to the database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
GEN_ID( )
SET GENERATOR
DROP GENERATOR

CREATE GLOBAL TEMPORARY TABLE [2.1]

(no contents yet)

CREATE INDEX

Creates an index on one or more columns in a table.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
ALTER INDEX
DROP INDEX
SELECT
SET STATISTICS

CREATE INDEX COMPUTED BY [2.0]

(no contents yet)

CREATE OR ALTER EXCEPTION [2.0]

(no contents yet)

CREATE OR ALTER {TRIGGER | PROCEDURE } [1.5]

(no contents yet)

CREATE PROCEDURE

Creates a stored procedure, its input and output parameters, and its actions.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information on creating and using procedures, see *Using Firebird- Programming on Firebird Server* (ch. 25 p. 494). For a complete description of the statements in procedure and trigger language, see chapter 3, *PSQL-Firebird Procedural Language* (p. 222).

See also:
ALTER EXCEPTION
ALTER PROCEDURE
CREATE EXCEPTION
DROP EXCEPTION
DROP PROCEDURE
EXECUTE PROCEDURE
SELECT

CREATE ROLE

Creates a role.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
GRANT
REVOKE
DROP ROLE

CREATE SEQUENCE [2.0]

Creates an integer number generator using SQL-99-compliant syntax.

Availability: +DSQL -ESQL +ISQL -PSQL

Syntax

```
CREATE ( SEQUENCE | GENERATOR ) <name>
```

Argument	Description
<name>	The name for the new generator / sequence.

Description

SEQUENCE is the SQL-99-compliant synonym for GENERATOR. SEQUENCE is a syntax term described in the SQL specification, whereas GENERATOR is a legacy InterBase syntax term.

It is recommended to use the standard SEQUENCE syntax.

A sequence generator is a mechanism for generating successive exact numeric values, one at a time. A sequence generator is a named schema object. In dialect 3 it is a BIGINT, in dialect 1 it is an INTEGER. It is often used to implement guaranteed unique IDs for records, to construct columns that behave like AUTOINC fields found in other RDBMSs.

Examples

```
CREATE SEQUENCE SEQ_ID_EMPLOYEE;
```

For a complete discussion on the concept and useage of sequences / generators, see the *Generator Guide* that is available as part of the Firebird documentation set.

See also:
CREATE GENERATOR
NEXT VALUE FOR
DROP SEQUENCE
ALTER SEQUENCE
CREATE TRIGGER

CREATE SHADOW

Creates one or more duplicate, in-sync copies of a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

Please also refer to *Using Firebird- Database shadows* (ch. 20 p. 379).

See also:
DROP SHADOW

CREATE TABLE

Creates a new table in an existing database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

Note 1: Constraints are not enforced on expressions.

Please also refer to *Using Firebird Tables* (ch. 17 p. 313) and *Managing Security* in ch. 22 of the same volume.

See also:
`CREATE DOMAIN`
`DECLARE TABLE`
`GRANT`
`REVOKE`

CREATE TRIGGER

Creates a trigger, including when it fires, and what actions it performs.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For a complete description of each statement, see chapter 3, *PSQL-Firebird Procedural Language* (p. 222). For discussion of programming triggers, see *Triggers, Coding the body of the code module* and *Implementing stored procedures and triggers* in *Using Firebird- Programming on Firebird Server* (ch. 25 p. 494).

See also:
ALTER EXCEPTION
ALTER TRIGGER
CREATE EXCEPTION
CREATE PROCEDURE
DROP EXCEPTION
DROP TRIGGER
EXECUTE PROCEDURE

CREATE TRIGGER ON CONNECT [2.1]

(no contents yet)

CREATE TRIGGER ON DISCONNECT [2.1]

(no contents yet)

CREATE TRIGGER ON TRANSACTION COMMIT [2.1]

(no contents yet)

CREATE TRIGGER ON TRANSACTION ROLLBACK [2.1]

(no contents yet)

CREATE TRIGGER ON TRANSACTION START [2.1]

(no contents yet)

CREATE VIEW

Creates a new view of data from one or more tables.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

Note 1: Although it is possible to create a view based on the output of a selectable stored procedure, it adds an unnecessary layer of dependency to do so. Using the output set of a stored procedure joined to a table, another view or another stored procedure is also theoretically possible but, in practice, it causes more trouble than it saves. With such complex requirements, it is almost invariably best to define the entire output within a selectable stored procedure.

A view is updatable if:

- It is a subset of a single table or another updatable view.
- All base table columns excluded from the view definition allow `NULL` values.
- The view's `SELECT` statement does not contain subqueries, a `DISTINCT` predicate, a `HAVING` clause, aggregate functions, joined tables, user-defined functions, or stored procedures.

If the view definition does not meet these conditions, it is considered read-only.

Note 2: Read-only views can be updated by using a combination of user-defined referential constraints, triggers, and unique indexes.

For a complete discussion, see *Using Firebird- Views* (ch. 19 p. 363).

See also:
`CREATE TABLE`
`DROP VIEW`
`GRANT`
`INSERT`
`REVOKE`
`SELECT`
`UPDATE`

CREATE VIEW [with column alias] [2.1]

(no contents yet)

CROSS JOIN [2.0]

(no contents yet)

See also:
`CROSS JOIN`

CURRENT_CONNECTION [1.5]

Context variable that holds the system ID of the current connection.

Availability: +DSQL +ESQL +ISQL +PSQL

Syntax

```
CURRENT_TRANSACTION
```

Important: Because the counter for this value is stored on the database header page, it will be reset after a database restore.

Argument	Description
CURRENT_CONNECTION	Returns the system identifier of the current connection.

Description

This context variable holds the current connection's system ID (data type INTEGER). It can be used for e.g. logging purposes. Every new connection that is made will receive a new, unique connection ID. In the monitoring tables (V2.1 and up), the value of CURRENT_CONNECTION corresponds to the field MON$ATTACHMENT_ID in MON$ATTACHMENTS, MON$TRANSACTIONS and MON$STATEMENTS.

Note: An active connection with a specific CURRENT_CONNECTION number will always correspond with one record in the MON$ATTACHMENTS table (but can have several associated transaction records in MON$TRANSACTIONS).

Examples

Obtain the current connection ID in a trigger:

```
NEW.CON_ID = CURRENT_CONNECTION;
```

List all transactions that are bound to the current connection (V2.1 and up):

```
SELECT * FROM MON$TRANSACTIONS WHERE MON$ATTACHMENT_ID=CURRENT_CONNECTION
```

List all statements that are executed within the current connection context, even if they use different transactions (V2.1 and up):

```
SELECT * FROM MON$STATEMENTS WHERE MON$ATTACHMENT_ID=CURRENT_CONNECTION
```

See also:
CURRENT_TRANSACTION
CURRENT_USER
CURRENT_ROLE

CURRENT_ROLE [1.5]

Context variable returning the current SQL user's role.

Availability: +DSQL +ESQL +ISQL +PSQL

Syntax

```
CURRENT_ROLE
```

Argument	Description
CURRENT_ROLE	Returns the name of the role of the current SQL user (if any).

Description

Returns the name of the role the current user logged in with (see also CURRENT_USER). If no role was specified, it returns "NONE".

1. If you insist on using an InterBase v.4.x or 5.1 database with Firebird, ROLE is not supported, so current_role will be NONE (as mandated by the SQL standard in absence of an explicit role) even if the user passed a role name.
2. If you use InterBase 5.5, IB 6 or Firebird, the ROLE passed is verified. If the role does not exist, it is reset to NONE without returning an error.

This means that in Firebird you can never get an invalid ROLE returned by CURRENT_ROLE, because it will be reset to NONE. This is in contrast with InterBase, where the bogus value is carried internally, although it is not visible to SQL.

Examples

```
SELECT CURRENT_ROLE FROM RDB$DATABASE
INSERT INTO RoleLog (ID, USERNAME)
VALUES (NEXT VALUE FOR SEQ_ID_ROLELOG, CURRENT_ROLE)
```

See also:
CURRENT_USER
CURRENT_TRANSACTION
CURRENT_CONNECTION

CURRENT_TRANSACTION [1.5]

Context variable that holds the system ID of the current transaction.

Availability: +DSQL +ESQL +ISQL +PSQL

Syntax

```
CURRENT_TRANSACTION
```

Important: Because the counter for this value is stored on the database header page, it will be reset after a database restore.

Argument	Description
CURRENT_TRANSACTION	Returns the system identifier of the current transaction.

Description

This context variable holds the current transaction's system ID (data type INTEGER). It can be used for e.g. logging purposes.

Every new transaction that is started will receive a new, unique transaction ID.

In the monitoring tables (V2.1 and up), the value of CURRENT_TRANSACTION corresponds to the fields MON$TRANSACTIONS.MON$TRANSACTION_ID and MON$STATEMENTS.MON$TRANSACTION_ID.

Examples

Obtain the current transaction ID in a trigger:

```
NEW.TXN_ID = CURRENT_TRANSACTION;
```

List all statements that are executed within the current transaction (V2.1 and up):

```
SELECT * FROM MON$STATEMENTS WHERE MON$TRANSACTION_ID=CURRENT_TRANSACTION
```

See also:
CURRENT_CONNECTION
CURRENT_USER
CURRENT_ROLE

CURRENT_USER [1.5]

Context variable returning the SQL user name.

Availability: +DSQL +ESQL +ISQL +PSQL

Syntax

```
CURRENT_USER
```

Argument	Description
CURRENT_USER	Returns the name of the current SQL user.

Description

CURRENT_USER is a DSQL synonym for USER that appears in the SQL standard. They are identical. There is no advantage of CURRENT_USER over USER.

Examples

```
SELECT CURRENT_USER FROM RDB$DATABASE
INSERT INTO UserLog (ID, USERNAME)
VALUES (NEXT VALUE FOR SEQ_ID_USERLOG, CURRENT_USER)
```

See also:
CURRENT_ROLE
CURRENT_TRANSACTION
CURRENT_CONNECTION

CURSOR FOR [2.0]

(no contents yet)

DATEADD( ) [2.1]

Returns a date/time/timestamp value increased (or decreased, when negative) by the specified amount of time.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
DATEADD( <number> <timestamp_part> FOR <date_time> )
DATEADD( <timestamp_part>, <number>, <date_time> )
timestamp_part ::= { YEAR | MONTH | DAY | WEEKDAY | HOUR | MINUTE | SECOND}
```

Important: If any of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<date_time>	The starting date, time or timestamp for the calculation.
<number>	The offset to be added to <date_time>.
<timestamp_part>	The unit for <number>.

Description

Returns a date/time/timestamp value increased (or decreased, when negative) by the specified amount of time.

Examples

```
select dateadd(1 day for current_date) from rdb$database
```
(returns tomorrow's date)

```
select dateadd(-1 day for current_date) from rdb$database
```
(returns yesterday's date)

```
select dateadd(weekday,1,current_date) from rdb$database
```
(returns the date of today's weekday in the next week)

```
select dateadd(weekday,1,current_timestamp) from rdb$database
```
(returns the timestamp of today's weekday in the next week with the current time)

See also:
DATEDIFF( )

DATEDIFF() [2.1]

Returns the interval between two dates/times/timestamps.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
DATEDIFF( <timestamp_part> FROM <date_time1> FOR <date_time2> )
DATEDIFF( <timestamp_part>, <date_time1>, <date_time2> )
timestamp_part ::= { YEAR | MONTH | DAY | WEEKDAY | HOUR | MINUTE | SECOND}
```

Important: If any of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<date_time1>	The first date, time or timestamp for the calculation.
<date_time2>	The second date, time or timestamp for the calculation.
<timestamp_part>	The unit for <number>.

Description

Returns an exact numeric value representing the interval of time from the first date/time/timestamp value to the second one.

Rules:

1. Returns a positive value if the second value is greater than the first one, negative when the first one is greater, or zero when they are equal.
2. Comparison of date with time values is invalid.
3. YEAR, MONTH, DAY and WEEKDAY cannot be used with time values.
4. HOUR, MINUTE and SECOND cannot be used with date values.
5. All timestamp_part values can be used with timestamp values.

Examples

```
select datediff(SECOND,cast(current_date as timestamp),current_timestamp)
from rdb$database
```

(returns the number of seconds elapsed since midnight. The CAST is necessary because of Rule 2)

```
 select datediff(DAY,dateadd(1 weekday for current_date),current_date) from
rdb$database
```

(returns -7)

```
select datediff(SECOND,current_time,current_time) from rdb$database
```

(returns 0)

```
select datediff(SECOND,current_date,current_date) from rdb$database
```

(throws an error because of Rule 5, returns NULL)

See also:
DATEADD()

DECLARE CURSOR

Defines a cursor for a table by associating a name with the set of rows specified in a SELECT statement.

Availability: +DSQL +ESQL +ISQL +PSQL

Blob form: See DECLARE CURSOR (BLOB)

(Syntax currently not included because of possible copyright issues.)

See also:
CLOSE
DECLARE CURSOR (BLOB)
FETCH
OPEN
PREPARE
SELECT

DECLARE CURSOR (BLOB)

Declares a blob cursor for read or insert.

Availability: +DSQL +ESQL +ISQL +PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CLOSE (BLOB)
FETCH (BLOB)
INSERT CURSOR (BLOB)
OPEN (BLOB)

DECLARE EXTERNAL FUNCTION

Declares an existing user-defined function (UDF) to a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

Note: that beginning with Firebird 1, you must list the path in the Firebird configuration file if it is other than `ib_install_dir/UDF`. A path name is no longer useful in the `DECLARE EXTERNAL FUNCTION` statement. The Firebird configuration file is called `ibconfig` on Windows machines, `isc_config` on Linux/UNIX machines.

For more information about writing and using UDFs, see *Using Firebird Working with UDFs and Blob Filters* (ch. 26 p. 572). For declarations of the UDFs in the `ib_udf` and `fbudf` libraries, see *User-defined Functions* on page 257 in chapter 6.

See also:
DROP EXTERNAL FUNCTION

DECLARE FILTER

Declares an existing blob filter to a database.

Availability: DSQL ESQL ISQL PSQL

Syntax

(Syntax currently not included because of possible copyright issues.)

For more information about Blob subtypes and instructions on writing blob filters, see *Using Firebird - BLOB filters* (ch. 26 p. 596) and associated topics in that section.

See also:
DROP FILTER

DECLARE STATEMENT

Identifies dynamic SQL statements before they are prepared and executed in an embedded program.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
EXECUTE
EXECUTE IMMEDIATE
PREPARE

DECLARE TABLE

Describes the structure of a table to the preprocessor, gpre, before it is created with CREATE TABLE.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CREATE DOMAIN
CREATE TABLE

DECODE() [2.1]

A shortcut for a `CASE ... WHEN ... ELSE` expression.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
DECODE( <expression>, <search>, <result>[ , <search>, <result> ... ] [, <default> ] )
```

Argument	Description
`<expression>`	The expression to decode.
`<search>`	A possible match for `<expression>`.
`<result>`	The value returned when `<expression>` matches the preceeding `<search>` value.
`<default>`	The value returned when none of the `<search>` values matched `<expression>`.

Description

`DECODE` is an inline version of a `CASE ... WHEN ... ELSE` construct.

Examples

```
select decode(state, 0, 'deleted', 1, 'active', 'unknown') from x
```
(Returns `'deleted'` when state equals `0`, `'active'` when state equals `1` and otherwise returns `'unknown'`)

```
select decode(rdb$system_flag,1,'SYSTEM',0,'USER','unknown') from rdb$ triggers
```
(Returns `'SYSTEM'` for system triggers and `'USER'` for user-defined ones.)

Note: the output column's name is `'CASE'`.

See also:
`CASE`

DELETE

Removes rows in a table or in the active set of a cursor.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about using cursors, see the *Embedded SQL Guide* (`EmbedSQL.pdf`) of the InterBase® 6 documentation set, obtainable from Borland.

See also:
`DECLARE CURSOR`
`FETCH`
`GRANT`
`OPEN`
`REVOKE`
`SELECT`

DESCRIBE

Provides information about columns that are retrieved by a dynamic SQL (DSQL) statement, or information about dynamic parameters that statement passes.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about ESQL programming and the XSQLDA descriptor, see the *Embedded SQL Guide* of the InterBase® 6 documentation set, available from Borland.

See also:
`EXECUTE`
`EXECUTE IMMEDIATE`
`PREPARE,`

DISCONNECT

Detaches an application from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`COMMIT`
`CONNECT`
`ROLLBACK`
`SET DATABASE`

DROP DATABASE

Deletes the currently attached database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`ALTER DATABASE`
`CREATE DATABASE`

DROP DEFAULT [2.0]

(no contents yet)

DROP DOMAIN

Deletes a domain from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
ALTER DOMAIN
ALTER TABLE
CREATE DOMAIN

DROP EXCEPTION

Deletes an exception from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
ALTER EXCEPTION
ALTER PROCEDURE
ALTER TRIGGER
CREATE EXCEPTION
CREATE PROCEDURE
CREATE TRIGGER

DROP EXTERNAL FUNCTION

Removes a user-defined function (UDF) declaration from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
DECLARE EXTERNAL FUNCTION

DROP FILTER

Removes a blob filter declaration from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
DECLARE FILTER

DROP GENERATOR

Removes a generator from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CREATE GENERATOR

DROP GENERATOR revisited [1.5]

(no contents yet)

DROP INDEX

Removes an index from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about integrity constraints and system-defined indexes, see *Using Firebird - Tables* (ch. 17 p. 313). For a discussion of indexing and related issues, see *Indexes* in ch. 18 of the same volume.

See also:
`ALTER INDEX`
`CREATE INDEX`

DROP PROCEDURE

Deletes an existing stored procedure from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`ALTER PROCEDURE`
`CREATE PROCEDURE`
`EXECUTE PROCEDURE`

DROP ROLE

Deletes a role from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`CREATE ROLE`
`GRANT`
`REVOKE`

DROP SEQUENCE [2.0]

Removes a sequence or generator from a database.

Availability: +DSQL +ESQL +ISQL -PSQL

Syntax

```
DROP SEQUENCE <name>
```

Important: It is not possible to drop a sequence when it is used by e.g. a trigger. You can query the RDB$DEPENCIES table, column RDB$DEPENDED_ON_NAME, to find out what triggers and/or stored procedures use a sequence.

Argument	Description
<name>	Name of the sequence / generator to be dropped.

Description

To remove a sequence from a database, use DROP SEQUENCE.

This command is equivalent to DROP GENERATOR, but uses the SQL-99-compliant SEQUENCE syntax. It is therefor recommended to use this syntax instead of DROP GENERATOR.

Examples

```
DROP SEQUENCE SEQ_ID_EMPLOYEE;
```

See also:
DROP GENERATOR
CREATE SEQUENCE
ALTER SEQUENCE
NEXT VALUE FOR

DROP SHADOW

Deletes a shadow from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CREATE SHADOW

DROP TABLE

Removes a table from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
ALTER TABLE
CREATE TABLE

DROP TRIGGER

Deletes an existing user-defined trigger from a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
ALTER TRIGGER
CREATE TRIGGER

DROP VIEW

Removes a view definition from the database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CREATE VIEW

END DECLARE SECTION

Identifies the end of a host-language variable declaration section.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
BASED ON
BEGIN DECLARE SECTION

EVENT INIT

Registers interest in one or more events with the Firebird event manager.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about events, see *Howevents work*, *Handling events on a client and related topics* in *Using Firebird- Programming on Firebird Server* (ch. 25 p. 494).

See also:
CREATE PROCEDURE
CREATE TRIGGER
EVENT WAIT
SET DATABASE

EVENT WAIT

Causes an application to wait until notified of an event's occurrence.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about events, see *Howevents work*, *Handling events on a client* and related topics in *Using Firebird - Programming on Firebird Server* (ch. 25 p. 494).

See also:
EVENT INIT

EXECUTE

Executes a previously prepared dynamic SQL (DSQL) statement.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about ESQL programming and the XSQLDA, see the *Embedded SQL Guide* (`EmbedSQL.pdf`) available from Borland.

See also:
DESCRIBE
EXECUTE IMMEDIATE
PREPARE

EXECUTE BLOCK [2.0]

(no contents yet)

EXECUTE IMMEDIATE

Prepares a dynamic SQL (DSQL) statement, executes it once, and discards it.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about ESQL programming and the XSQLDA, see the *Embedded SQL Guide*.

See also:
`DESCRIBE`
`PREPARE`

EXECUTE PROCEDURE

Calls a stored procedure.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about indicator variables, see the *Embedded SQL Guide* (`EmbedSQL.pdf`) from the InterBase® 6 documentation set, available from Borland.

See also:
`ALTER PROCEDURE`
`CREATE PROCEDURE`
`DROP PROCEDURE`

EXECUTE STATEMENT [1.5]

(no contents yet)

EXP( ) [2.1]

Returns the exponential e to the argument.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
EXP(<number>)
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number.

Description

Returns the exponential e to the argument.

Examples

```
select EXP(0) from rdb$database
```
(returns `1`)

```
select EXP(1) from rdb$database
```
(returns `2,718281828459` or e)

```
select EXP(2) from rdb$database
```
(returns `7,3890560989307` or e^2)

See also:
`POWER()`

EXTRACT( )

Extracts date and time information from `DATE`, `TIME`, and `TIMESTAMP` values.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

FETCH

Retrieves the next available row from the active set of an opened cursor.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about cursors and XSQLDA, see the *Embedded SQL Guide* (`EmbedSQL.pdf`) from the InterBase® 6 documentation set, available from Borland.

See also:
`CLOSE`
`DECLARE CURSOR`
`DELETE`
`FETCH (BLOB)`
`OPEN`

FETCH (BLOB)

Retrieves the next available segment of a blob column and places it in the specified local buffer.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`BASED ON`
`CLOSE (BLOB)`
`DECLARE CURSOR (BLOB)`
`INSERT CURSOR (BLOB)`
`OPEN (BLOB)`

FIRST(m) SKIP(n)

(Text currently not included because of possible copyright issues.)

FLOOR( ) [2.1]

Returns a value representing the greatest integer that is lesser than or equal to the input argument.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
FLOOR( <number> )
```

Argument	Description
<number>	The number whose next-greater integer value is returned.

Description

Returns a value representing the greatest integer that is lesser than or equal to the input argument.

Examples

```
select floor(1.0) from rdb$database
```
(returns 1)

```
select floor(1.9) from rdb$database
```
(returns 1)

```
select floor(-1.1) from rdb$database
```
(returns -2)

See also:
CEIL( )
ROUND( )

FOR UPDATE [WITH LOCK] [1.5]

(no contents yet)

GDSCODE [1.5]

(no contents yet)

GEN_ID( )

Produces a system-generated integer value.

Availability: DSQL ESQL ISQL PSQL

(This text is currently not included because of possible copyright issues.)

See also:
CREATE GENERATOR
SET GENERATOR

GEN_UUID() [2.1]

Returns a universal unique number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
GEN_UUID()
```

Description

Returns a universal unique number.

Example

```
insert into records (id) value (gen_uuid());
```

See also:
GEN_ID()

GRANT

Assigns privileges to users for specified database objects.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about privileges, see *Using Firebird- Database-level security* (ch. 22 p. 429).

See also:
REVOKE

HASH( ) [2.1]

Returns a HASH of a value.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
HASH(<string>)
```

Important: If <string> is (or evaluates to) NULL, the result is NULL.

Argument	Description
<string>	The string the hash is calculated from.

Description

Returns a HASH of a value.

Examples

```
select HASH('') from rdb$database
```
(returns 0)

```
select HASH('Firebird') from rdb$database
```
(returns 20678676612)

```
select HASH('Firebird'||NULL) from rdb$database
```
(returns NULL)

IIF [2.0]

Shortcut function for a two-branch CASE construct:

Availability: DSQL ESQL ISQL PSQL

Syntax

```
IIF (<search_condition>, <value1>, <value2>)
```

Argument	Description
<search_condition>	The condition to be evaluated.
<value1>	The result returned if the <search_condition> evaluates to TRUE.
<value2>	The result returned if the <search_condition> evaluates to FALSE.

Description

IIF() returns the value of the first sub-expression if the given search condition evaluates to TRUE, otherwise it returns a value of the second sub-expression. It is implemented as a shortcut function for the following CASE construct:

```
CASE
WHEN <search_condition> THEN <value1>
ELSE <value2>
END
```

Examples

```
SELECT IIF(VAL > 0, VAL, -VAL) FROM OPERATION
```

See also:
CASE
COALESCE()
NULLIF()
DECODE()
ibec_IIF
Firebird 2.0.4. Release Notes: IIF expression syntax added

INSERT

Adds one or more new rows to a specified table.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

Argument	Description
`<TRANSACTION>`	Transaction name of the transaction that controls the execution of the `INSERT INTO` object `Name` of an existing table or view into which to insert.
`data col`	Name of an existing column in a table or view into which to insert values.
`VALUES (val [, val ...])`	Lists values to insert into the table or view; values must be listed in the same order as the target columns `select_expr` Query that returns row values to insert into target columns.

See also:
`GRANT`
`REVOKE`
`SET TRANSACTION`
`UPDATE`

INSERT CURSOR (BLOB)

Inserts data into a blob cursor in units of a blob segment-length or less in size.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`CLOSE (BLOB)`
`DECLARE CURSOR (BLOB)`
`FETCH (BLOB)`
`OPEN (BLOB)`

INSERT INTO ... DEFAULT VALUES [2.1]

Inserts a record without supplying field values.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
INSERT INTO <table> DEFAULT VALUES [RETURNING <values>]
```

Argument	Description
`<table>`	The table to insert a record into.
`<values>`	Optional return parameters (see `RETURNING`).

Description

Allows `INSERT` without supplying values, if `Before Insert` triggers and/or declared defaults are available for every column and none is dependent on the presence of any supplied `'NEW'` value.

Examples

```
INSERT INTO TableWithDefaults DEFAULT VALUES;
```

See also:
`INSERT`
`RETURNING`
`UPDATE OR INSERT`
Firebird 2.0.4 Release Notes: `RETURNING` clause for insert statements
`SELECT`
`SELECT` statement

INSERTING, UPDATING, DELETING [1.5]

(no contents yet)

LEAVE / BREAK [1.5]

(no contents yet)

LEAVE [<label_name>] [2.0]

(no contents yet)

LEFT( ) [2.1]

Returns the substring of a specified length.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
LEFT( <string expression>, <numeric expression> )
```

Important: if either of the arguments is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<string>`	The string expression (e.g. a field) where the output gets copied from.
`<numeric expression>`	Denotes how many chars the output will contain.

Description

Returns the substring of a specified length that appears at the start of a left-to-right string.

Examples

```
select left('Firebird', 4) from rdb$database
```
returns 'Fire'

```
select left('', 10) from rdb$database
```
returns ''

See also:
`RIGHT()`

LIKE ... ESCAPE?? [1.5]

(no contents yet)

LIST() [2.1]

Returns a string with concatenated matches.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
LIST '(' [ {ALL | DISTINCT} ] <value expression> [',' <delimiter
value>] ')'
<delimiter value> ::= { <string literal> | <parameter> |
<variable> }
```

Argument	Description
`<value expression>`	The expression to be concatenated.
`<delimiter value>`	The separator inserted between any matches.

Description

This function returns a string result with the concatenated non-`NULL` values from a group. It returns `NULL` if there are no non-`NULL` values.

Rules:

1. If neither `ALL` nor `DISTINCT` is specified, `ALL` is implied.
2. If `<delimiter value>` is omitted, a comma is used to separate the concatenated values.

Other Notes:

1. Numeric and date/time values are implicitly converted to strings during evaluation.
2. The result value is a BLOB with `SUB_TYPE TEXT` for all cases except list of BLOB with different subtype.
3. Ordering of values within a group is implementation-defined.

Examples

```
/* A */
SELECT LIST(ID, ':')
FROM MY_TABLE
/* B */
SELECT TAG_TYPE, LIST(TAG_VALUE)
FROM TAGS
GROUP BY TAG_TYPE
```

LN( ) [2.1]

Returns the natural logarithm of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
LN(<number>)
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number whose natural logarithm is returned.

Description

Returns the natural logarithm of a number.

Examples

```
select ln(0) from rdb$database
```
(throws the error `'expression evaluation not supported'` and returns `NULL`)

```
select ln(1) from rdb$database
```
(returns `0`)

```
select ln(10) from rdb$database
```
(returns `2,302585092994`)

```
select ln(exp(1)) from rdb$database
```
(returns `1`)

See also:
`EXP( )`

LOG( ) [2.1]

Returns the logarithm base x of y.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
LOG( <number1>, <number2> )
```

Important: If either of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<number1>	The logarithm base.
<number2>	The number whose logarithm base <number1> is calculated.

Description

Returns the logarithm base x of y.

Examples

```
select log(1,10) from rdb$database
```
(returns INF)

```
select log(0,0) from rdb$database
```
(returns NAN)

```
select log(exp(1),10) from rdb$database
```
(returns 2,302585092994)

```
select log(10,10000) from rdb$database
```
(returns 4)

See also:
LOG10( )

LOG10() [2.1]

Returns the logarithm base ten of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
LOG10(<number>)
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number whose logarith base 10 is calculated.

Description

Returns the logarithm base ten of a number. The function is equivalent to `LOG(10,<number>)`.

Examples

```
select log10(0) from rdb$database
```
(returns `-INF`)

```
select log10(1) from rdb$database
```
(returns `0`)

```
select log10(10) from rdb$database
```
(returns `1`)

```
select log10(10000) from rdb$database
```
(returns `4`)

See also:
`LOG()`

LOWER() [2.0]

Converts a string to all lower case.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
LOWER (<val>)
```

Argument	Description
val	A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype.

Description

LOWER() converts a specified string to all lower case characters. If applied to character sets that have no case differentiation, LOWER() has no effect.

Examples

The following ISQL statement changes the name, BMatthews, to bmatthews:

```
UPDATE EMPLOYEE
SET EMP_NAME = LOWER ('BMatthews')
WHERE EMP_NAME = 'BMatthews';
```

The next ISQL statement creates a domain called PROJNO with a CHECK constraint that requires the value of the column to be all lower case:

```
CREATE DOMAIN PROJNO
AS CHAR(5)
CHECK (VALUE = LOWER (VALUE));
```

See also:
CAST()
UPPER()
Firebird 2.0.4 Release Notes: New features for text data

LPAD() [2.1]

Prepends `string2` to the beginning of `string1`.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
LPAD( <string1>, <number> [, <string2> ] )
```

Important: If either of the arguments is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<string1>`	The string expression to be padded.
`<number>`	The length of the output string.
`<string2>`	The string to be prepended (default is a blank or space).

Description

`LPAD(string1, length, string2)` prepends `string2` to the beginning of `string1` until the length of the result string becomes equal to length.

Rules:

1. If the second string is omitted the default value is one space.
2. If the result string would exceed the length, the second string is truncated.

Examples

```
select LPAD('TEST',10) from rdb$database
```
(returns `' TEST'`, see Rule 1)

```
select LPAD('TEST',10,'x') from rdb$database
```
(returns `'xxxxxxTEST'`)

```
select LPAD('TEST',10,'1234') from rdb$database
```
(returns `'123412TEST'`, see Rule 2)

```
select LPAD('1234567890',5,'x') from rdb$database
```
(returns `'12345'`, that is: the output string is limited in length to `<number>`)

MAX()

Retrieves the maximum value in a column.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
AVG()
COUNT()
CREATE DATABASE
CREATE TABLE
MIN()
SUM()

MAXVALUE() [2.1]

Returns the maximum value of a list of values.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
MAXVALUE( <number> [,<number>] )
```

Important: If any of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<number>	A number or numeric expression).

Description

Returns the maximum value of a list of values.

Examples

```
select MAXVALUE(1,5,3) from rdb$database
```
(returns 5)

```
select MAXVALUE(1,5,NULL) from rdb$database
```
(returns NULL)

See also:
MAX()
MIN()
MINVALUE()

MIN()

Retrieves the minimum value in a column.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
AVG()
COUNT()
CREATE DATABASE
CREATE TABLE
MAX()
SUM()

MINVALUE() [2.1]

Returns the minimum value of a list of values.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
MINVALUE( <number> [,<number>] )
```

Important: If any of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<number>	A number or numeric expression.

Description

Returns the minimum value of a list of values.

Examples

```
select MINVALUE(1,5,3) from rdb$database
```
(returns 1)

```
select MINVALUE(1,5,NULL) from rdb$database
```
(returns NULL)

See also:
MAX()
MIN()
MAXVALUE()

MOD() [2.1]

Returns the remainder part of the division of X by Y.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
MOD( <number1>, <number2> )
```

Argument	Description
<number1>	The number or numeric expression the modulo is calculated from.
<number2>	The number or numeric expression that <number1> is divided by to calculate the modulo.

Description

Modulo: MOD(X, Y) returns the remainder part of the division of X by Y.

Examples

```
select MOD(10,3) from rdb$database
```
(returns 1)

```
select MOD(10,5) from rdb$database
```
(returns 0)

```
select MOD(-10,3) from rdb$database
```
(returns -1)

See also:
TRUNC()

MON$ Tables [2.1]

(no contents yet)

NATURAL JOIN [2.1]

(no contents yet)

NEXT VALUE FOR [2.0]

Generates the next value for a sequence / generator.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
NEXT VALUE FOR <name>
```

Argument	Description
<name>	Name of the sequence / generator whose next value is returned.

Description

Generates and returns the next value for a sequence.

The `NEXT VALUE FOR <name>` expression is a synonym for `GEN_ID(<name>, 1)`, using the SQL-99-compliant `SEQUENCE` syntax.

While the `GEN_ID()` function allows an optional step or increment value to be supplied in the function call, the increment is implicitly set to `1` when using `NEXT VALUE FOR`.

Examples

This example generates a new value for the `ID` column using a sequence, and returns that new value to the caller:

```
INSERT INTO EMPLOYEE (ID, NAME)
VALUES (NEXT VALUE FOR SEQ_ID_EMPLOYEE, 'John Smith')
RETURNING ID;
```

For more information about the use of sequences, refer to the *Generator Guide* that is available as part of the Firebird documentation set.

See also:
`GEN_ID()`
`CREATE SEQUENCE`
`ALTER SEQUENCE`
`DROP SEQUENCE`

NULLIF [1.5]

Returns NULL for a subexpression if it has a specific value, otherwise returns the value of the subexpression.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
NULLIF ( <value expression1> , <value expression2> )
```

Argument	Description
<value expression1>	The value returned when it is not NULL.
<value expression2>	The value returned if <value expression1> evaluates to NULL.

Description

Returns NULL for a subexpression if it has a specific value, otherwise returns the value of the subexpression.

NULLIF (V1, V2) is equivalent to the following case specification: CASE WHEN V1 = V2 THEN NULL ELSE V1 END.

Examples

```
UPDATE PRODUCTS SET STOCK = NULLIF(STOCK,0)
```

See also:
CASE
COALESCE()
DECODE()
IIF()

OPEN

Retrieve specified rows from a cursor declaration.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CLOSE
DECLARE CURSOR
FETCH

OPEN (BLOB)

Opens a previously declared blob cursor and prepares it for read or insert.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
OPEN [TRANSACTION name] cursor
{INTO | USING} :blob_id;
```

Argument	Description
`TRANSACTION name`	Specifies the transaction under which the cursor is opened .
`Default`	The default transaction.
`cursor`	Name of the blob cursor.
`INTO | USING`	Depending on the blob cursor type, use one of these: `INTO`: For `INSERT BLOB` `USING`: For `READ BLOB`.
`blob_id`	Identifier for the blob column.

See also:
`CLOSE (BLOB)`
`DECLARE CURSOR (BLOB)`
`FETCH (BLOB)`
`INSERT CURSOR (BLOB)`

OVERLAY( ) [2.1]

Returns `string1` replacing the substring from `<start>` for `<length>` by `string2`.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
OVERLAY( <string1> PLACING <string2> FROM <start> [ FOR <length> ] )
```

Important:

If either of the arguments is (or evaluates to) `NULL`, the result is `NULL`. Use the `FOR <length>` clause with care - see the examples below!

Description

Returns `string1` replacing the substring from `<start>` for `<length>` by `string2`.

The `OVERLAY` function is equivalent to:

```
SUBSTRING(<string1>, 1 FOR <start> - 1) || <string2> || SUBSTRING(<string1>, <start> + <length>)
```

If `<length>` is not specified, `CHAR_LENGTH( <string2> )` is implied. If `<length>` is specified, then the `<length>` characters of `<string1>` starting with character `<start>` will be replaced with the entire `<string2>`, that is `<string2>` will not be clipped or padded to adjust it to `<length>`.

Examples

```
select OVERLAY('1234567890' PLACING 'ABCD' FROM 3) from rdb$database
```
(returns `'12ABCD7890'`)

```
select OVERLAY('1234567890' PLACING 'ABCD' FROM 9) from rdb$database
```
(returns `'12345678ABCD'` - note the output is longer than `string1`!)

```
select OVERLAY('1234567890' PLACING 'ABCD' FROM 3 FOR 2 ) from rdb$database
```
(returns `'12ABCD567890'`)

```
select OVERLAY('1234567890' PLACING 'ABCD' FROM 3 FOR 4 ) from rdb$database
```
(returns `'12ABCD7890'`)

```
select OVERLAY('1234567890' PLACING 'ABCD' FROM 3 FOR 6 ) from rdb$database
```
(returns `'12ABCD90'`)

See also:
`SUBSTRING( )`

PI() [2.1]

Returns the number `PI`.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
PI()
```

Description

Returns the number `PI` with a precision of 13 decimals.

Examples

```
select PI() from rdb$database
```
(returns `3,1415926535898`)

See also:
`SIN()`
`COS()`

POSITION() [2.1]

returns the position of the substring X in the string Y.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
POSITION(<string1> IN <String2>)
```

Important: If either of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<string1>	The string whose position is to be found in <string2>.
<string2>	The string where <string1> is searched in.

Description

Returns the position of the substring X in the string Y. Returns 0 if X is not found within Y. The character matching is case sensitive.

Examples

```
select POSITION('bird' IN 'Firebird') from rdb$database
```
(returns 5)

```
select POSITION('Bird' IN 'Firebird') from rdb$database
```
(returns 0 - search is case sensitive!)

See also:
SUBSTRING()

POWER() [2.1]

Returns X to the power of Y.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
POWER( <number1>, <number2> )
```

Important: If either of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<number1>	The number that is put to the power of <number2>.

Description

Returns X to the power of Y. The function is equivalent to <number1>^<number2>.

Examples

```
select power(2,16) from rdb$database
```
(returns 65536)

```
select power(10,6) from rdb$database
```
(returns 1000000)

```
select power(10,1.5) from rdb$database
```
(returns 31,6227766016838)

```
select power(10,-1) from rdb$database
```
(returns 0.1)

See also:
EXP()

PREPARE

Prepares a statement for execution in embedded SQL.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

Note: This statement could also be prepared and described in the following manner:
```
EXEC SQL PREPARE Q FROM :buf; EXEC SQL DESCRIBE Q INTO SQL DESCRIPTOR xsqlda;
```

See also:
DESCRIBE
EXECUTE
EXECUTE IMMEDIATE

RAND() [2.1]

Returns a random value in the range between `0` and `1`.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
RAND()
```

Description

Returns a random value in the range between `0` and `1`.

Examples

```
select rand() from rdb$database
```
(returns a random double precision value with up to 13 decimals)

RDB$GET_CONTEXT [2.0]

(no contents yet)

RDB$SET_CONTEXT [2.0]

(no contents yet)

RECREATE EXCEPTION [2.0]

(no contents yet)

RECREATE PROCEDURE

`RECREATE PROCEDURE` redefines an existing stored procedure to a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`DROP PROCEDURE`
`CREATE PROCEDURE`
`ALTER PROCEDURE`

RECREATE TABLE

`RECREATE TABLE` redefines an existing table to a database.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`DROP TABLE`
`CREATE TABLE`
`ALTER TABLE`

RECREATE TRIGGER [2.0]

(no contents yet)

RECREATE VIEW

(Syntax currently not included because of possible copyright issues.)

RELEASE SAVEPOINT [1.5]

(no contents yet)

REPLACE( ) [2.1]

Replaces all occurrences of `<findstring>` in `<stringtosearch>` with `<replstring>`.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
REPLACE( <stringtosearch>, <findstring>, <replstring>)
```

Important: If either of the arguments is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<stringtosearch>`	The string to be searched and replaced in `<findstring>`.
`<findstring>`	The string where `<stringtosearch>` is searched in.
`<replstring>`	The string to replace `<findstring>`.

Description

Replaces all occurrences of `<findstring>` in `<stringtosearch>` with `<replstring>`. Search is *NOT* case sensitive.

Examples

```
select REPLACE('Firebird','i','1') from rdb$database
```
(returns `'F1reb1rd'`)

```
select REPLACE('Firefox','f','b') from rdb$database
```
(returns `'Firebox'` - search is not case sensitive)

```
select REPLACE('123123','2','two') from rdb$database
```
(returns `'1two31two3'`)

```
select REPLACE('ABCDE','B','BCB') from rdb$database
```
(returns `'ABCBCDE'` - replacement is not recursive)

See also:
`POSITION( )`
`SUBSTRING( )`

RETURNING [2.1]

Returns columns from an INSERT, UPDATE or DELETE operation to the caller.

Availability: +DSQL +ESQL +ISQL +PSQL

Syntax

```
INSERT INTO ... VALUES (...)
[RETURNING <column_list> [INTO <variable_list>]]
INSERT INTO ... SELECT ...
[RETURNING <column_list> [INTO <variable_list>]]
UPDATE OR INSERT INTO ... VALUES (...) ...
[RETURNING <column_list> [INTO <variable_list>]]
UPDATE ...
[RETURNING <column_list> [INTO <variable_list>]]
DELETE FROM ...
[RETURNING <column_list> [INTO <variable_list>]]
```

Important: In DSQL, the statement always returns the set, even if the operation had no effect on any record. Hence, at this stage of implementation, the potential exists to return an "empty" set. (This may be changed in a future version.)

Argument	Description
<column_list>	The list of columns to be returned as a result of the respective operation.
<variable_list>	Optional list of result variables to take the returned values (PSQL only).

Description

The purpose of the RETURNING clause is to enable the column values stored into a table as a result of the INSERT, UPDATE OR INSERT, UPDATE and DELETE statements to be returned to the client. The most likely usage is for retrieving the value generated for a primary key inside a BEFORE-trigger.

The RETURNING clause is optional and is available in both DSQL and PSQL, although the rules differ slightly. In DSQL, the execution of the operation itself and the return of the set occur in a single protocol round trip.

Because the RETURNING clause is designed to return a singleton set in response to completing an operation on a single record, it is not valid to specify the clause in a statement that inserts, updates or deletes multiple records.

Rules for using a RETURNING clause:

1. The INTO part (i.e. the variable list) is allowed in PSQL only, for assigning the output set to local variables. It is rejected in DSQL. 2. The presence of the RETURNING clause causes an INSERT statement to be described by the API as isc_info_sql_stmt_exec_procedure rather than isc_info_sql_stmt_insert. Existing connectivity drivers should already be capable of supporting this feature without special alterations. 3. The RETURNING clause ignores any explicit record change (update or delete) that occurs as a result of the execution of an AFTER trigger. 4. OLD and NEW context variables can be used in the RETURNING clause of UPDATE and UPDATE OR INSERT statements. 5. In UPDATE and UPDATE OR INSERT statements, field references that are unqualified or qualified by table name or relation alias are resolved to the value of the corresponding NEW context variable.

Examples

1.
```
INSERT INTO T1 (F1, F2)
VALUES (:F1, :F2)
RETURNING F1, F2 INTO :V1, :V2;
```

2.
```
INSERT INTO T2 (F1, F2)
VALUES (1, 2)
RETURNING ID INTO :PK;
```

3.
```
DELETE FROM T1
WHERE F1 = 1
RETURNING F2;
```

4.
```
UPDATE T1
SET F2 = F2 * 10
RETURNING OLD.F2, NEW.F2;
```

See also:
INSERT
UPDATE
DELETE
UPDATE OR INSERT
Firebird 2.0.4 Release Notes: RETURNING clause for insert statements
INSERT INTO ... DEFAULT VALUES
SELECT
SELECT statement

REVERSE ( ) [2.1]

Returns a string in reverse order.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
REVERSE(<string expression>)
```

Important: if `<string expression>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<string expression>`	The string to be returned in reverse order.

Description

Returns a string in reverse order. Useful function for creating an expression index that indexes strings from right to left.

Examples

```
create index people_email on people
computed by (reverse(email));

select * from people
where reverse(email) starting with reverse('.br');

select reverse('Firebird') from rdb$database;
```
(returns 'driberiF')

```
select reverse('reliefpfeiler') from rdb$database;
```
(returns 'reliefpfeiler', which is an existing German word!)

REVOKE

Withdraws privileges from users for specified database objects.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`GRANT`

REVOKE ADMIN OPTION FROM [2.0]

(no contents yet)

RIGHT() [2.1]

Returns the rightmost part of a string.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
RIGHT( <string>, <numeric expression> )
```

Important: If either of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<string>	The string expression (e.g. a field) where the output gets copied from.
<numeric expression>	Denotes how many chars the output will contain.

Description

Returns a substring, of the specified length, from the right-hand end of a string.

Examples

```
select right('Firebird',4) from rdb$database
```
(returns 'bird')

```
select right('Firebird',10) from rdb$database
```
(returns 'Firebird', that is the output is not padded if <string> is shorter than 10)

See also:
LEFT()
SUBSTRING()

ROLLBACK

Restores the database to its state prior to the start of the current transaction.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about controlling transactions, see *Using Firebird- Transactions in Firebird* (ch. 8 p. 90).

See also:
COMMIT
DISCONNECT

ROLLBACK RETAIN [2.0]

(no contents yet)

ROLLBACK [WORK] TO [SAVEPOINT] [1.5]

(no contents yet)

ROUND() [2.1]

Returns a number rounded to the specified scale.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
ROUND(<number1>,<number2>)
```

Important: If any of the arguments is (or evaluates to) NULL, the result is NULL.

Argument	Description
<number1>	The number or numeric expression to be rounded.
<number2>	The scale (number of decimal places) <number1> is rounded to.

Description

Returns a number rounded to the specified scale. If the scale (<number2>) is negative, the integer part of the value is rounded.

Examples

```
select round(0.123456789,6) from rdb$database
```
(returns 0.123457)

```
select round(0.123456789,3) from rdb$database
```
(returns 0.123)

```
select round(12345.6789,0) from rdb$database
```
(returns 12346.0)

```
select round(12345.6789,-3) from rdb$database
```
(returns 12000.0)

See also:
TRUNC()

ROWS [2.0]

(no contents yet)

ROW_COUNT [1.5]

(no contents yet)

RPAD() [2.1]

Appends `string2` to the end of `string1`.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
RPAD( <string1>, <number> [, <string2> ] )
```

Important: If either of the arguments is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<string1>`	The string expression to be padded.
`<number>`	The length of the output string.
`<string2>`	The string to be appended (default is a blank or space).

Description

`RPAD(string1, length, string2)` appends `string2` to the end of `string1` until the length of the result string becomes equal to length.

Rules:

1. If the second string is omitted the default value is one space.
2. If the result string would exceed the length, the second string is truncated.

Examples

```
select RPAD('TEST',10) from rdb$database
```
(returns `'TEST  '`, see Rule 1)

```
select RPAD('TEST',10,'x') from rdb$database
```
(returns `'TESTxxxxxx'`)

```
select RPAD('TEST',10,'1234') from rdb$database
```
(returns `'TEST123412'`, see Rule 2)

```
select RPAD('1234567890',5,'x') from rdb$database
```
(returns `'12345'`, that is: the output string is limited in length to `<number>`)

See also:
`LPAD()`

SAVEPOINT [1.5]

(no contents yet)

SELECT

Retrieves data from one or more tables.

Availability: DSQL ESQL ISQL PSQL*
*In PSQL, a variant syntax for SELECT is available. For details, refer to notes on *SELECT* and *FOR SELECT...INTO...DO* in the chapter *PSQL-Firebird Procedural Language*.

(Syntax currently not included because of possible copyright issues.)

Argument	Description
TRANSACTION Transaction	Name of the transaction under control of which the statement is executed; ESQL only.
SELECT [DISTINCT | ALL]	Specifies data to retrieve.
DISTINCT	Prevents duplicate values from being returned. ALL, the default, retrieves every value.
SELECT {[FIRST m] | [SKIP n]} ... ORDER BY ...	FIRST m returns an output set consisting of m rows, optionally SKIPping n rows and returning a set beginning (n+1) rows from the "top" of the set specified by the rest of the SELECT specification. If SKIP n is used and the [FIRST m] parameter is omitted, the output set returns all rows in the SELECT specification except the "top" n rows. These parameters generally make sense only if applied to a sorted set.
{*|val [, val ...]}	The asterisk (*) retrieves all columns for the specified tables.
val [, val ...]	Retrieves a list of specified columns, values and expressions.
INTO :var [, var ...]	Singleton select in ESQL only; specifies a list of host-language variables into which to retrieve values.
FROM tableref [, tableref ...]	List of tables, views, and stored procedures from which to retrieve data; list can include joins and joins can be nested.
table	Name of an existing table in a database.
view	Name of an existing view in a database.
procedure	Name of an existing stored procedure that functions like a SELECT statement.
alias	Brief, alternate name for a table, view, or column; after declaration in tableref, alias can stand in for subsequent references to a table or view.
joined_table	A table reference consisting of a JOIN.
join_type	Type of join to perform.
Default: INNER WHERE search_condition	Specifies a condition that limits rows retrieved to a subset of all available rows.
GROUP BY col [, col ...]	Partitions the results of a query, assembling the output into groups formed on the basis of common values in all of the output columns named in the grouping list. Precedence of grouping columns is left=high. Aggregations apply to the grouping column having the lowest precedence.
COLLATE collation	Specifies the collation order for the data retrieved by the query HAVING.
search_condition	Used with GROUP BY; specifies a condition that limits grouped rows returned UNION.
[ALL]	Combines two or more tables that are fully or partially identical in structure; the ALL option keeps identical rows separate instead of folding them together into one.
PLAN plan_expr	Specifies the access plan for the Firebird optimizer to use during retrieval.
plan_item	Specifies a table and index method for a plan ORDER BY.
order_list	Specifies columns to order, either by column name or ordinal number in the query, and the order (ASC or DESC) in which rows to return the rows.

For discussions of topics related to query specifications and SQL, see *Using Firebird- Firebird SQL & Queries* (ch. 9 p. 110)., For a full discussion of data retrieval in embedded programming using DECLARE CURSOR and SELECT, see the *Embedded SQL Guide (EmbedSQL)* of the InterBase® 6 documentation set, available from Borland.

See also:
DECLARE CURSOR
DELETE
INSERT
UPDATE
UPDATE OR INSERT
Firebird 2.0.4 Release Notes: RETURNING clause for insert statements
INSERT INTO ... DEFAULT VALUES
SELECT statement
RETURNING

SET DATABASE

Declares a database handle for database access.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information on the security database, see *Using Firebird- Managing Security* (ch. 22 p. 414).

See also:
COMMIT
CONNECT
ROLLBACK
SELECT

SET DEFAULT [2.0]

(no contents yet)

SET GENERATOR

Sets a new value for an existing generator.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CREATE GENERATOR
CREATE PROCEDURE
CREATE TRIGGER
GEN_ID()

SET HEAD[ing] toggle [2.0]

(no contents yet)

SET NAMES

Specifies an active character set to use for subsequent database attachments.

Availability: DSQL ESQL ISQL PSQL

(This text is currently not included because of possible copyright issues.)

For more information about character sets and collation orders, see *Using Firebird- Character Sets and Collation Orders* (ch. 16 p. 301).

See also:
CONNECT
SET DATABASE

SET SQL DIALECT

Declares the SQL Dialect for database access.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
SHOW SQL DIALECT

SET SQLDA_DISPLAY ON/OFF [2.0]

(no contents yet)

SET STATISTICS

Recomputes the selectivity of a specified index.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
ALTER INDEX
CREATE INDEX
DROP INDEX

SET TRANSACTION

Starts a transaction and optionally specifies its behavior.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

For more information about transactions, see *Using Firebird- Transactions in Firebird* (ch. 8 p. 90).

See also:
COMMIT
ROLLBACK
SET NAMES

SHOW SQL DIALECT

Returns the current client SQL dialect setting and the database SQL dialect value.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
SET SQL DIALECT

SIGN() [2.1]

Returns the sign of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
SIGN( <number> )
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number or numeric expression whose sign is returned.

Description

Returns `1`, `0`, or `-1` depending on whether the input value is positive, zero or negative, respectively.

Examples

```
select SIGN(-99) from rdb$database
```
(returns `-1`)

```
select SIGN(0) from rdb$database
```
(returns `0`)

```
select SIGN(99) from rdb$database
```
(returns `1`)

See also:
`ABS()`

SIN() [2.1]

Returns the sine of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
SIN(<number>)
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number or numeric expression whose sine is returned.

Description

Returns the sine of a number. The angle is specified in radians and returns a value in the range `-1` to `1`.

Examples

```
select sin(0) from rdb$database
```
(returns `0`)

```
select sin(-1) from rdb$database
```
(returns `-0,8414709848079`)

```
select sin(1) from rdb$database
```
(returns `0,8414709848079`)

```
select sin(PI()) from rdb$database
```
(returns `0`)

```
select sin(PI()/2) from rdb$database
```
(returns `1`)

See also:
`COS()`
`SINH()`

SINH() [2.1]

Returns the hyperbolic sine of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
SINH(<number>)
```

Important:

If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number or numeric expression whose sine is returned.

Description

Returns the hyperbolic sine of a number. The angle is specified in radians and returns a value in the range `-1` to `1`.

Examples

```
select sinh(0) from rdb$database
```
(returns `0`)

```
select sinh(-1) from rdb$database
```
(returns `-1,1752011936438`)

```
select sinh(1) from rdb$database
```
(returns `1,1752011936438`)

See also:
`COS()`
`SIN()`

SQL Commands

(no contents yet)

SQLCODE [1.5]

(no contents yet)

SQRT() [2.1]

Returns the square root of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
SQRT(<number>)
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number or numeric expression whose square root is returned.

Description

Returns the square root of a number.

Examples

```
select sqrt(0) from rdb$database
```
(returns `0`)

```
select sqrt(9) from rdb$database
```
(returns `3`)

```
select sqrt(-1) from rdb$database
```
(throws the error `'expression evaluation not supported'`, returns `NULL`)

See also:
`POWER()`

SUBSTRING()

Returns a string of specified length from within an input string, starting from a specified position in the input string.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also: The user-defined (external) functions `substr` and `substrlen`.

See also:
Firebird 2.0.4 Release Notes: Built-in function `SUBSTRING()` enhanced

SUM()

Totals the numeric values in a specified column.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
`AVG()`
`COUNT()`
`MAX()`
`MIN()`

TAN() [2.1]

Returns the tangent of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
TAN(<number>)
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number whose tangent is returned.

Description

Returns the tangent of an input number that is expressed in radians.

Examples

```
select tan(0) from rdb$database
```
(returns `0`)

```
select tan(-1) from rdb$database
```
(returns `-1,5574077246549`)

```
select tan(1) from rdb$database
```
(returns `1,5574077246549`)

See also:
`COT()`
`TANH()`

TANH( ) [2.1]

Returns the hyperbolic tangent of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
TANH(<number>)
```

Important: If `<number>` is (or evaluates to) `NULL`, the result is `NULL`.

Argument	Description
`<number>`	The number whose hyperbolic tangent is returned.

Description

Returns the hyperbolic tangent of an input number that is expressed in radians.

Examples

```
select tanh(0) from rdb$database
```
(returns `0`)

```
select tan(-1) from rdb$database
```
(returns `-0,7615941559558`)

```
select tanh(1) from rdb$database
```
(returns `0,7615941559558`)

See also:
`COT()` `TAN()`

TRIM() [2.0]

Trims characters (default: blanks) from the left and/or right of a string.

Availability: DSQL ESQL ISQL PSQL

Syntax

Simple:

```
TRIM (<val>)
```

Complete:

```
TRIM <left paren> [ [ <trim specification> ] [ <trim character> ]
FROM ] <value expression> <right paren>
<trim specification> ::= LEADING | TRAILING | BOTH
<trim character> ::= <value expression>
```

Argument	Description
`val`	A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype.

Description

`TRIM()` trims characters (default: blanks) from the left and/or right of a string.

Rules:

1. If `<trim specification>` is not specified, `BOTH` is assumed.
2. If `<trim character>` is not specified, `' '` is assumed.
3. If `<trim specification>` and/or `<trim character>` is specified, `FROM` should be specified.
4. If `<trim specification>` and `<trim character>` is not specified, `FROM` should not be specified.

See also:
RPAD()
LPAD()
Firebird 2.0.4 Release Notes: New features for text data

TRUNC() [2.1]

Returns the integral part of a number.

Availability: DSQL ESQL ISQL PSQL

Syntax

```
TRUNC( <number> )
```

Important: If <number> is (or evaluates to) NULL, the result is NULL.

Argument	Description
<number>	The number or numeric expression whose integral part is returned.

Description

Returns the integral part of a number. The function is equal to FLOOR() for positive numbers.

Examples

```
select trunc(1.1) from rdb$database
```
(returns 1)

```
select trunc(-1.1) from rdb$database
```
(returns -1, note FLOOR() would return -2 here.)

See also:
FLOOR()
CEIL()

TYPE OF [domains in PSQL] [2.1]

(no contents yet)

UNION DISTINCT [2.0]

(no contents yet)

UPDATE

Changes the data in all or part of an existing row in a table, view, or active set of a cursor.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
DELETE
GRANT
INSERT
REVOKE
SELECT

UPDATE OR INSERT [2.1]

Updates or inserts a record depending on whether it is already present.

Availability: +DSQL +ESQL +ISQL +PSQL

Syntax

```
UPDATE OR INSERT INTO <table or view> [(<column_list1>)]
VALUES (<value_list>)
[MATCHING <column_list2>]
[RETURNING <column_list3> [INTO <variable_list>]]
```

Important: INSERT and UPDATE permissions are needed on <table or view>. A "multiple rows in singleton select" error will be raised if the RETURNING clause is present and more than one record matches the search condition.

Argument	Description
<table or view>	The table or view where the update or insert takes place.
<column_list1>	Optional list of fields to update or insert.
<value_list>	List of field values to update or insert.
<column_list2>	List of fields that determine whether or not the record already exists.
<column_list3>	Optional list of returned values (see RETURNING).
<variable_list>	Optional list of variables where the RETURNING values are returned into.

Description

This syntax has been introduced to enable a record to be either updated or inserted, according to whether or not it already exists (checked with IS NOT DISTINCT).

When MATCHING is omitted, the existence of a primary key is required.

If the RETURNING clause is present, then the statement is described as isc_info_sql_stmt_exec_procedure by the API; otherwise, it is described as isc_info_sql_stmt_insert.

Examples

In the first example it is assumed that T1 has a primary key (e.g. on F1):

1.
```
UPDATE OR INSERT INTO T1 (F1, F2)
VALUES (:F1, :F2);
```

The second example returns the updated or inserted ID: 2.
```
UPDATE OR INSERT INTO EMPLOYEE (ID, NAME)
VALUES (:ID, :NAME)
RETURNING ID;
```

Here the decision to INSERT or to UPDATE is based on F1, be it the primary key or not: 3.
```
UPDATE OR INSERT INTO T1 (F1, F2)
VALUES (:F1, :F2)
MATCHING (F1);
```

In this example, in case ID already existed, the OLD contents of field NAME is returned: 4.
```
UPDATE OR INSERT INTO EMPLOYEE (ID, NAME)
VALUES (:ID, :NAME)
RETURNING OLD.NAME;
```

See also:
INSERT
UPDATE
RETURNING

UPPER()

Converts a string to all uppercase.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

See also:
CAST()
and the user-defined (external) function lower()

WHENEVER

Traps SQLCODE errors and warnings.

Availability: DSQL ESQL ISQL PSQL

(Syntax currently not included because of possible copyright issues.)

WITH [RECURSIVE] (CTE) [2.1]

(no contents yet)

Document history

Revision History		
0.1	FI	First Beta

License note

Glossary

The majority of definitions can be found in the relevant IBExpert subject areas. This glossary includes a number of miscellaneous definitions that could not be allotted to individual IBExpert subjects.

If you are looking for a specific definition in the online documentation, please use the search function. Should you not be able to find the definition you are looking for, please contact documentation@ibexpert.com.

.NET

Microsoft's framework for Web services and component software was introduced in 2000 and is pronounced "dot-net."

.NET supports all the web-based features and functions, including XML and the web services protocols such as SOAP and UDDI. .NET applications run on intranets as well as public Internet sites, thus .NET is an all-inclusive web-oriented software architecture for internal and external use.

The .NET Framework created by Microsoft is a software development platform focused on rapid application development (RAD), platform independence and network transparency. It has introduced a new programming language environment that compiles all source code into an intermediate language. .NET languages are compiled into the Microsoft Intermediate Language (MSIL), which is executed by the Common Language Runtime (CLR) software in the Windows computer. The MSIL is similar to Java's bytecode, except that whereas Java is one language, .NET supports multiple programming languages such as Microsoft's C# and VB.NET. A subset of the CLR has been standardized by ECMA so that third parties can port non-Microsoft programming languages and create runtime environments for operating systems other than Windows.

It erases the boundaries between applications and the Internet. Instead of interacting with an application or a single web site, .NET will connect the user to an array of computers and services that will exchange and combine objects and data.

.NET has brought new functionalities and tools to the application programming interface (API). These innovations allow programmers to develop applications for both Windows and the web as well as components and services (web services).

* / Wildcard

The asterisk (*) or so-called wildcard is used, for example, when selecting all or any data (or data sets) meeting a certain condition.

Example

```
SELECT * FROM EMPLOYEE
WHERE EMPLOYEE.PHONE_EXT='250';
```

All data sets containing the value `250` in the `PHONE_EXT` column in the `EMPLOYEE` table are fetched.

Aggregate functions

A function that performs a computation on a set of values rather than on a single value, to calculate group-level totals and statistics. For example, finding the average or mean of a list of numbers is an aggregate function.

All database management and spreadsheet systems support a set of aggregate functions that can operate on a set of selected records or cells.

Aggregate functions perform calculations over a series of values, such as the columns retrieved with a `SELECT` statement. These include `AVG()`, `COUNT()`, `MAX()`, `MIN()`, `SUM()`.

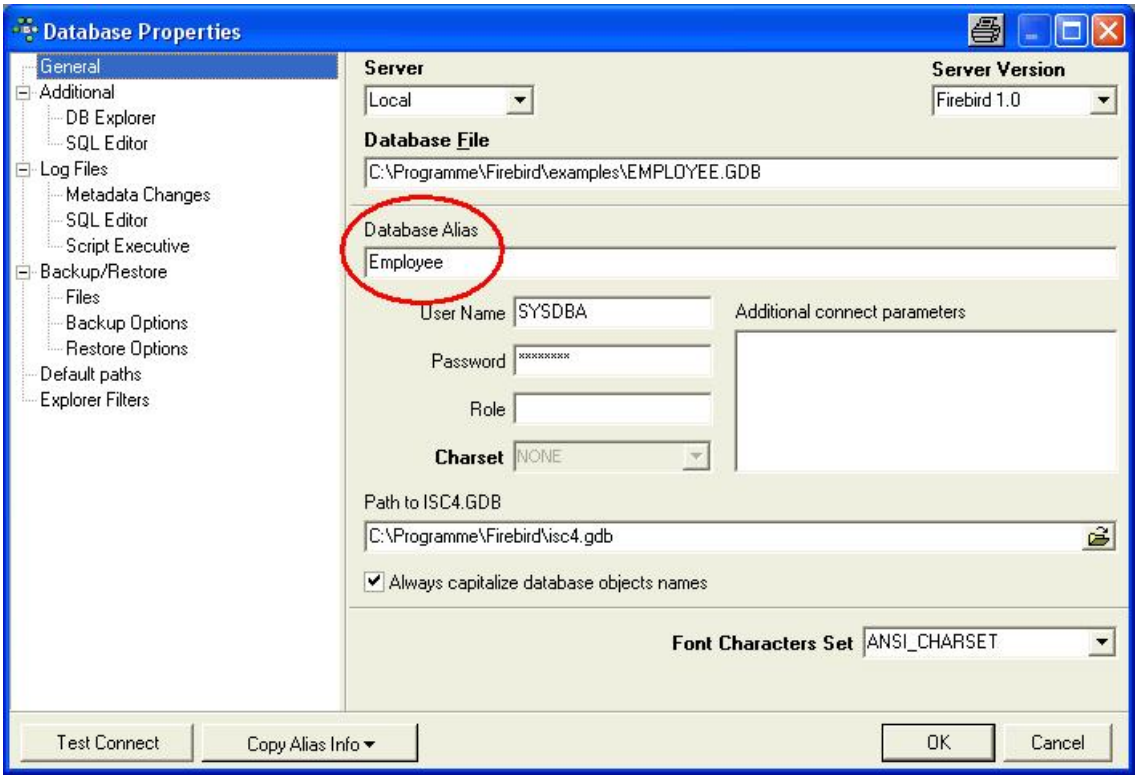
See also:
Conversion Functions

Alias

An alias is a pseudonym. A database alias is a name chosen by the developer for day-to-day use, as a logical and preferable alternative to the usually formally named `gdb` or `fdb` file, which is often named in accordance to internal company norms.

The alias indicates the location of the database tables. If the database is stored on a server, the alias also specifies the necessary connection parameters.

It is also used in SQL language to simplify input (saves repeatedly typing the same long database object and field names).



Please refer to the *Configuring Firebird* chapter, Alias, files and paths for detailed information about Firebird database aliases.

See also:
Firebird 2.x Administration Handbook, Alias names

API (Application Program Interface)

API is the abbreviation for Application Program Interface, which is a set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

Most operating environments, such as MS Windows, provide an API so that programmers can write applications consistent with the operating environment. Although APIs are designed for programmers, they are ultimately of advantage to users because they guarantee that all programs using a common API will have similar interfaces. This makes it easier for users to learn new programs.

Source: http://www.webopedia.com/

See also:
ISAPI
NSAPI

Application

An application is a program or group of programs designed for end users. Software can be divided into two general classes: systems software and applications software. Systems software consists of low-level programs that interact with the computer at a very basic level. This includes operating systems, compilers, and utilities for managing computer resources.

In contrast, applications software (also called end-user programs) includes database programs, word processors, and spreadsheets. Figuratively speaking, applications software sits on top of systems software because it is unable to run without the operating system and system utilities.

An application comprises the executing file, along with any other files, that a program needs to function fully. The word application is often used synonymously with the word program.

Source: http://www.webopedia.com/

ASCII

ASCII is an acronym for the American Standard Code for Information Interchange. Pronounced ask-ee, ASCII is a code for representing English characters as numbers, with each letter assigned a number from `0` to `127`. For example, the ASCII code for uppercase `M` is `77`. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Text files stored in ASCII format are sometimes called ASCII files. Text editors and word processors are usually capable of storing data in ASCII format, although ASCII format is not always the default storage format. Most data files, particularly if they contain numeric data, are not stored in ASCII format. Executable programs are never stored in ASCII format.

The standard ASCII character set uses just 7 bits for each character. There are several larger character sets that use 8 bits, which gives them 128 additional characters. The extra characters are used to represent non-English characters, graphics symbols, and mathematical symbols. Several companies and organizations have proposed extensions for these 128 characters. The DOS operating system uses a superset of ASCII called extended ASCII or high ASCII. A more universal standard is the ISO Latin 1 set of characters, which is used by many operating systems, as well as web browsers.

Source: http://www.webopedia.com/

BDE (Borland Database Engine)

BDE is the abbreviation for the Borland Database Engine, the heart of Firebird/InterBase. IBExpert uses this database engine to access and retrieve data. It allows multiple sessions, each one being treated as a "virtual" user.

Benchmark

Benchmarks are normed testing techniques, used to evaluate and compare the performance of IT systems, according to certain predefined criteria. They are a vital tool when the performance of databases and/or hardware needs to be assessed objectively.

Many hardware manufacturers and also trade magazines have developed their own benchmark tests, which they use when reviewing a class of products. When comparing benchmark results, it is important to know exactly what the benchmarks are designed to test.

See also:
IBExpert Benchmarks

BLR (Binary Language Representation)

As Firebird/InterBase internally does not understand SQL, all statements (queries, updates, metadata manipulation) are internally represented in a binary notation. When stored procedure or trigger code is compiled, it is translated into BLR and the BLR representation is kept in a Blob subtype field. This translation is performed only once, which is why stored procedures are good for efficiency. The command-line tool isql shows the BLR representation of stored procedures (and triggers, constraints and table definitions) after issuing a `SET BLOB ALL` command and then using a `SELECT` statement to get the appropriate BLR fields from the system tables which are accessed as `RDB$RELATIONS`.

CGI (Common Gateway Interface)

Abbreviation of Common Gateway Interface, a specification for transferring information between a World Wide Web server and a CGI program. A CGI program is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any programming language, including C, Perl, Java, or Visual Basic.

CGI programs are the most common way for web servers to interact dynamically with users. Many HTML pages that contain forms, for example, use a CGI program to process the form's data once it's submitted. Another increasingly common way to provide dynamic feedback for web users is to include scripts or programs that run on the user's machine rather than the web server. These programs can be Java applets, Java scripts, or ActiveX controls. These technologies are known collectively as client-side solutions, while the use of CGI is a server-side solution because the processing occurs on the web server.

One problem with CGI is that each time a CGI script is executed, a new process is started. For busy web sites, this can slow down the server noticeably. A more efficient solution, but one that it is also more difficult to implement, is to use the server's API, such as ISAPI or NSAPI. Another increasingly popular solution is to use Java servlets.

Source: http://www.webopedia.com/

Client/Server

The main part of the database intelligence is contained in a server program (e.g. InterBase/Firebird). The operation is sent from the client to the server and is processed there, and the resulting data transferred back to the client.

Client-server architecture is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers).

Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power.

Another type of network architecture is known as a peer-to-peer architecture because each node has equivalent responsibilities. Both client/server and peer-to-peer architectures are widely used, and each has unique advantages and disadvantages.

Client-server architectures are also sometimes called two-tier architectures.

CLSID

A CLSID is the abbreviation for class identifier. It is a globally unique identifier that identifies a COM class object. The CLSID structure wraps the COM class identifier structure, which serves as a unique identifier for a specific COM class. If your server or container allows linking to its embedded objects, you need to register a CLSID for each supported class of objects.

Comdiag

Comdiag is an InterBase/Firebird windows-based program to aid diagnosis of problems that may arise when connecting to InterBase/Firebird servers and the databases managed by those servers.

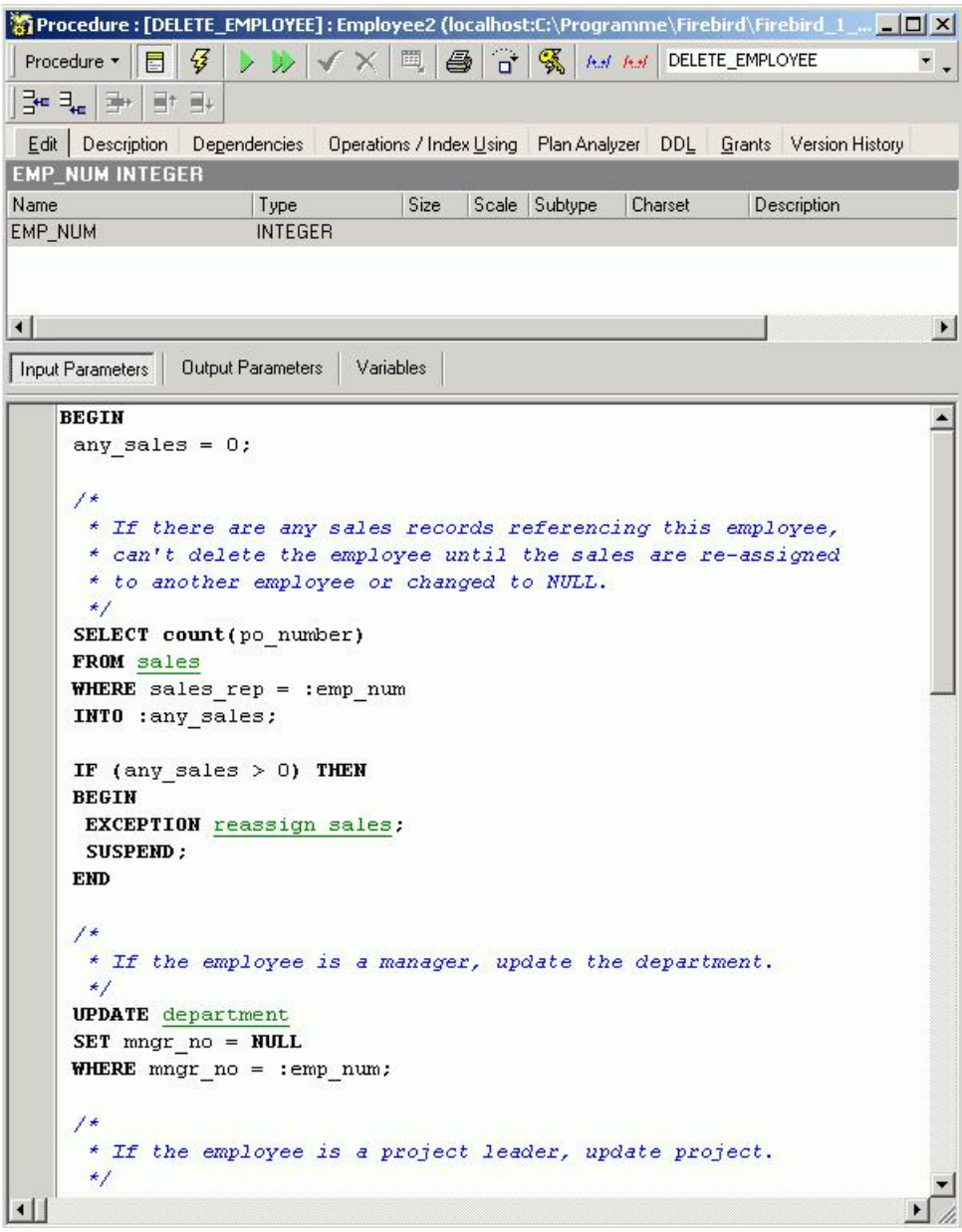
It validates all InterBase DLLs when connecting the server to the database and checks that the various protocol stacks are correctly installed and loaded.

Further information can be found under the IBExpert Services menu item, Communication Diagnostics.

Comments

Comments can be incorporated anywhere in an InterBase/Firebird ISQL script, as well as in the procedure body of a stored procedure. The following character sequences are used to determine a comment.

```
/* Comment */
```



Comments can span multiple lines, but a comment cannot be embedded in another comment. They can also be incorporated in a Firebird script, determined by the following character sequence:

```
-- Comment
```

Comments introduced in this way in Firebird can only cover a single line, i.e. each new line must begin with `--`. Firebird however also understands the InterBase syntax.

See also:
Comment Selected/Uncomment Selected

Compile and Commit / Rollback

A transaction is committed, if all statements in the transactions were performed successfully and the whole transaction was completed without error. By committing a transaction, the instructions entered are interpreted and saved permanently to disk or cancelled. In IBExpert the

icon or [Ctrl + F9] can be used to perform this task. The *Compile* dialog shows whether the modifications, insertions or deletions are correct; the *Commit* button finally writes the alterations permanently to the database.

A transaction is rolled back, if the alterations are cancelled or revoked by the operator, or if an active transaction is perceived by another transaction to be "dead" and so set in a rolled-back condition. Rollback also aborts the compile actions, should errors have been reported or modifications be necessary.

See also:
Data Transaction: `COMMIT` and `ROLLBACK`
`COMMIT`
`ROLLBACK RETAIN` syntax

Conditional Test

Conditional test is an expression that evaluates to logical `TRUE` or `FALSE`. If the statement `TRUE`, the statements in the `THEN` clause are executed; if `FALSE`, the statements in the optional `ELSE` clause are executed. Parentheses around the conditional test are required.

Please also refer to IF ... THEN ... ELSE.

See also:
Comparison Operators

Constant

In programming, a constant is a value that never changes. The other type of values that programs use is variables, symbols that can represent different values throughout the course of a program.

A constant can be

- a number, such as `25` or `3.6`
- a character, such as a or `$`
- a character string, such as `"this is a string"`

Source: http://www.webopedia.com/

Conversion functions

Conversion functions transform datatypes, either converting them from one type to another, or by changing the scale or precision of numeric values, or by converting `CHARACTER` datatypes to all uppercase. These include `CAST()`, `EXTRACT()`], `UPPER()`.

DBMS (Database Management System)

A collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMSs, ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

- computerized library systems
- automated teller machines
- flight reservation systems
- computerized parts inventory systems

From a technical standpoint, DBMSs can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Requests for information from a database are made in the form of a query, which is a stylized question. For example, the query

```
SELECT ALL WHERE NAME = "SMITH" AND AGE > 35
```

requests all records in which the NAME field is SMITH and the AGE field is greater than 35. The set of rules for constructing queries is known as a query language. Different DBMSs support different query languages, although there is a semi-standardized query language called SQL (Structured Query Language). Sophisticated languages for managing database systems are called fourth-generation languages, or 4GLs for short.

The information from a database can be presented in a variety of formats. Most DBMSs include a report writer program that enables you to output data in the form of a report. Many DBMSs also include a graphics component that enables you to output information in the form of graphs and charts.

Source: http://www.webopedia.com/

DDE (Dynamic Data Exchange)

DDE is an acronym for Dynamic Data Exchange, an interprocess communication (IPC) system built into the Macintosh, Windows, and OS/2 operating systems. DDE enables two running applications to share the same data.

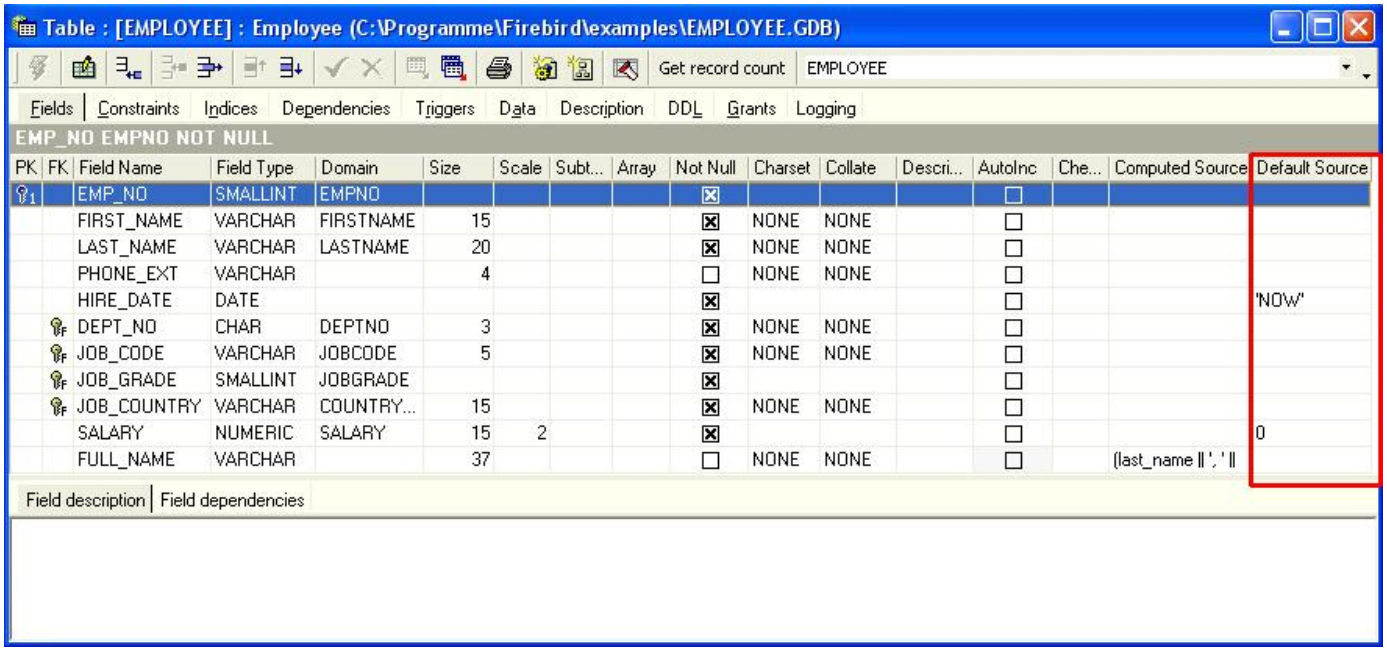
Although the DDE mechanism is still used by many applications, it is being supplanted by OLE, which provides greater control over shared data.

Source: http://www.webopedia.com/

Default

The DEFAULT parameter allows a standard value to be defined, should the user not enter a specific value. A DEFAULT value can be defined for a domain or a field. The default value predefined in the domain, can be overridden by the default value entry in the column/field definition following this domain.

In IBExpert it can be specified when creating a new table and fields or when creating a domain.



DLL (Dynamic Link Library)

DLL is the abbreviation for Dynamic Link Library. DLLs are library files with the suffix DLL. These are executable modules, containing source code or resources, which can access other DLLs or applications. DLLs enable multiple applications, source code and resource to be used collectively in a Windows environment.

See also:
User-Defined Function (UDF)
`DECLARE EXTERNAL FUNCTION`

Event

An action or occurrence detected by a program. Events can be user actions, such as clicking a mouse button or pressing a key, or system occurrences, such as running out of memory. Most modern applications, particularly those that run in Macintosh and Windows environments, are said to be event-driven, because they are designed to respond to events.

A database event can be anything relative to the rows in a table or values in fields. Coordinated and monitored by the Firebird/InterBase Event Manager.

Expression

An expression is a group of symbols that represent a value.

In programming, an expression is any legal combination of symbols that represents a value. Each programming language and application has its own rules for what is legal and illegal. For example, in the C language `x+5` is an expression, as is the character string `"MONKEYS"`.

Every expression consists of at least one operand and can have one or more operators. Operands are values, whereas operators are symbols that represent particular actions. In the expression

```
x + 5
```

`x` and `5` are operands, and `+` is an operator.

Expressions are used in programming languages, database systems, and spreadsheet applications. For example, in database systems, you use expressions to specify which information you want to see. These types of expressions are called queries.

Expressions are often classified by the type of value that they represent. For example:

- **Boolean expressions:** Evaluate to either `TRUE` or `FALSE`
- **Integer expressions:** Evaluate to whole numbers, like `3` or `100`
- **Floating-point expressions:** Evaluate to real numbers, like `3.141` or `-0.005`
- **String expressions:** Evaluate to character strings

Source: http://www.webopedia.com/

See also:
Datatypes

FBK Files

FBK is the standard suffix used for Firebird backup database file names.

This is not compulsory, in fact a Firebird or InterBase backup database may be named with any suffix. This standardization does however provide a certain conformity, of particular importance if a database is to be administrated long term by numerous people.

FDB Files

FDB is the standard suffix used for Firebird database file names. It is derived from the InterBase standard, `.GDB`.

This is not compulsory, in fact an Firebird or InterBase database may be named with any suffix. This standardization does however provide a certain conformity, of particular importance if a database is to be administrated long term by numerous people.

FTP (File Transfer Protocol)

FTP is an abbreviation of File Transfer Protocol, the protocol for exchanging files over the Internet. FTP works in the same way as HTTP for transferring web pages from a server to a user's browser and SMTP for transferring electronic mail across the internet in that, like these technologies, FTP uses the internet's TCP/IP protocols to enable data transfer.

FTP is most commonly used to download a file from a server using the internet or to upload a file to a server (e.g., uploading a web page file to a server).

Source: http://www.webopedia.com/

GBK Files

`.GBK` is the standard suffix used for Borland InterBase backup database file names.

This is not compulsory, in fact an InterBase or Firebird backup database may be named with any suffix. This standardization does however provide a certain conformity, of particular importance if a database is to be administrated long term by numerous people.

GDB Files

`.GDB` is the standard suffix used for Borland InterBase database file names. It originates back to the days when the Interbase Corporation was still called Groton Database Systems.

This is not compulsory, in fact an InterBase or Firebird database may be named with any suffix. This standardization does however provide a certain conformity, of particular importance if a database is to be administrated long term by numerous people.

See also:
`.FDB` files

GRC Files

.GRC files are IBExpert Database Designer files.

GUID (Globally Unique Identifier)

Short for Globally Unique Identifier, a unique 128-bit number that is produced by the Windows OS or by some Windows applications to identify a particular component, application, file, database entry, and/or user. For instance, a website may generate a GUID and assign it to a user's browser to record and track the session. A GUID is also used in a Windows registry to identify COM DLLs. Knowing where to look in the registry and having the correct GUID yields a lot information about a COM object (i.e., information in the type library, its physical location, etc.). Windows also identifies user accounts by a username (computer/domain and username) and assigns it a GUID. Some database administrators even will use GUIDs as primary key values in databases.

GUIDs can be created in a number of ways, but usually they are a combination of a few unique settings based on specific point in time (e.g., an IP address, network MAC address, clock date/time, etc.).

Source: http://www.webopedia.com/

See also:
CLSID

Hashing / Hash Values

Producing hash values for accessing data or for security. A hash value (or simply hash), also called a message digest, is a number generated from a string of text. The hash is substantially smaller than the text itself, and is generated by a formula in such a way that it is extremely unlikely that some other text will produce the same hash value.

Hashes play a role in security systems where they're used to ensure that transmitted messages have not been tampered with. The sender generates a hash of the message, encrypts it, and sends it with the message itself. The recipient then decrypts both the message and the hash, produces another hash from the received message, and compares the two hashes. If they're the same, there is a very high probability that the message was transmitted intact.

Hashing is also a common method of accessing data records. Consider, for example, a list of names:

- John Smith
- Sarah Jones
- Roger Adams

To create an index, called a hash table, for these records, you would apply a formula to each name to produce a unique numeric value. So you might get something like:

- 1345873 John Smith
- 3097905 Sarah Jones
- 4060964 Roger Adams

Then to search for the record containing Sarah Jones, you just need to reapply the formula, which directly yields the index key to the record. This is much more efficient than searching through all the records till the matching record is found.

Source: http://www.webopedia.com/

HTML (HyperText Markup Language)

Short for HyperText Markup Language, the authoring language used to create documents on the World Wide Web. HTML is similar to SGML (Standard Generalized Markup Language), although it is not a strict subset. HTML defines the structure and layout of a web document by using a variety of tags and attributes. The correct structure for an HTML document starts with `<HTML><HEAD>` (enter here what document is about), `<BODY>` and ends with `</BODY></HTML>`. All the information you'd like to include in your web page fits in between the `<BODY>` and `</BODY>` tags.

There are hundreds of other tags used to format and layout the information in a web page. Tags are also used to specify hypertext links. These allow web developers to direct users to other web pages with only a click of the mouse on either an image or word(s).

Source: http://www.webopedia.com/

See also:
Declaring character sets in XML and HTML
Generate HTML documentation in IBExpert

HTTP (HyperText Transfer Protocol)

Short for HyperText Transfer Protocol, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the web server directing it to fetch and transmit the requested web page.

The other main standard that controls how the World Wide Web works is HTML, which covers how web pages are formatted and displayed.

HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement web sites that react intelligently to user input. This shortcoming of HTTP is being addressed in a number of new technologies, including ActiveX, Java, JavaScript and cookies.

Source: http://www.webopedia.com/

Hyperlink

A hyperlink is an element in an electronic application or document that links to another place in the same application/editor/text or to an entirely different editor/text. Typically, you click on the hyperlink to follow the link. Hyperlinks are the most essential ingredient of all hypertext systems, including the World Wide Web.

IDE (Integrated Development Environment)

Abbreviated as IDE, a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger. Visual Studio, Delphi, JBuilder, FrontPage and DreamWeaver are all examples of IDEs.

ISAPI (Internet Server Application Programming Interface)

The Internet Server Application Programming Interface (ISAPI) is the API of Internet Information Services (IIS), Microsoft's collection of Windows-based network services. ISAPI was designed to model N-tier architecture. ISAPI enables programmers to develop web-based applications that run much faster than

conventional CGI programs because they're more tightly integrated with the web server. In addition to IIS, several web servers from companies other than Microsoft support ISAPI.

See also:
NSAPI

LIP (Log Information Page)

The log information pages (LIP) for the write-ahead log (WAL) are not currently used, though code to use them is included conditionally in Firebird.

NSAPI (Netscape Server Application Programming Interface)

Short for Netscape Server Application Programming Interface, an API for Netscape's Web servers. NSAPI enables programmers to create web-based applications that are more sophisticated and run much faster than applications based on CGI scripts.

See also:
ISAPI

OAT (Oldest Active Transaction)

The Oldest Active Transaction (OAT) is the earliest transaction in the database, recorded by the versioning engine in the TIP (Transaction Inventory Page) that is currently active or open.

See also:
Oldest Active Transaction (OAT)
OIT

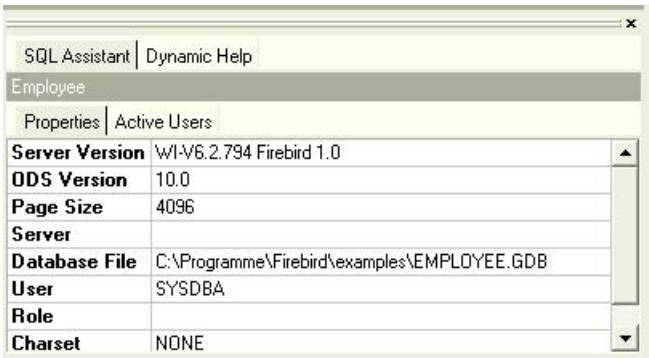
ODBC (Open DataBase Connectivity)

ODBC (pronounced as separate letters) is short for Open DataBase Connectivity, a standard database access method developed by the SQL Access group in 1992. The goal of ODBC is to make it possible to access any data from any application, regardless of which database management system (DBMS) is handling the data. ODBC manages this by inserting a middle layer, called a database driver, between an application and the DBMS. The purpose of this layer is to translate the application's data queries into commands that the DBMS understands. For this to work, both the application and the DBMS must be ODBC-compliant - that is, the application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them. Since version 2.0, the standard supports SAG SQL.

Source: http://www.webopedia.com/

ODS Version

ODS = On-Disk Structure.



The ODS version shows with which database version the database was created, e.g. InterBase 5 = 9, InterBase 6 = 10.0, InterBase 6.5 = 10.1, InterBase 7 = 11.

For more information about the InterBase On-Disk Structure, please refer to Ann Harrison's article, *Space Management in InterBase*.

See also:
SQL Assistant

OIT (Oldest Interesting Transaction)

The Oldest Interesting Transaction (OIT) is the earliest transaction in the database, recorded by the versioning engine in the TIP (Transaction Inventory Page) with a status other than committed. Every transaction prior to that one represents an unbroken chain of insertions and updates into the database.

See also:
OAT
Oldest Interesting Transaction (OIT)

OLAP (Online Analytical Processing)

Short for Online Analytical Processing, a category of software tools that provides analysis of data stored in a database. OLAP tools enable users to analyze different dimensions of multidimensional data. For example, it provides time series and trend analysis views. OLAP often is used in data mining.

The chief component of OLAP is the OLAP server, which sits between a client and a database management system (DBMS). The OLAP server understands how data is organized in the database and has special functions for analyzing the data. There are OLAP servers available for nearly all the major database systems.

Source: http://www.webopedia.com/

See also:
Data Analysis

OLE (Object Linking and Embedding)

OLE is an abbreviation of Object Linking and Embedding, pronounced as separate letters or as oh-leh. OLE is a compound document standard developed by the Microsoft Corporation. It enables you to create objects with one application and then link or embed them in a second application. Embedded objects retain their original format and links to the application that created them.

Support for OLE is built into the Windows and Macintosh operating systems. A competing compound document standard developed jointly by IBM, Apple Computer, and other computer firms is called OpenDoc.

Source: http://www.webopedia.com/

Operand

In all computer languages, expressions consist of two types of components: operands and operators. Operands are the objects that are manipulated and operators are the symbols that represent specific actions. For example, in the expression

```
5 + x
```

x and 5 are operands and + is an operator. All expressions have at least one operand.

Source: http://www.webopedia.com/

See also:
Comparison Operators

Operator

An operator is a symbol that represents a specific action. For example, a plus sign (+) is an operator that represents addition. The basic mathematic operators are + addition, - subtraction, * multiplication, / division.

In addition to these operators, many programs and programming languages recognize other operators that allow you to manipulate numbers and text in more sophisticated ways. For example, Boolean operators enable you to test the truth or falsity of conditions, and relational operators let you compare one value to another. For example, the expression

```
x < 5
```

means x is less than 5. This expression will have a value of TRUE if the variable x is less than 5; otherwise the value of the expression will be FALSE.

Relational operators are sometimes called comparison operators. Expressions that contain relational operators are called relational expressions.

Source: http://www.webopedia.com/

Orphan pages

Orphan pages are unassigned disk space that should be returned to free space. They are physically allocated and registered on the page inventory page (PIP).

GFIX, the repair and modification tool is able to combat orphan pages in the database file.

Parameter

1. Characteristic. For example, specifying parameters means defining the characteristics of something. In general, parameters are used to customize a program. For example, filenames, page lengths, and font specifications could all be considered parameters.

2. In programming, the term parameter is synonymous with argument, a value that is passed to a routine.

Source: http://www.webopedia.com/

PHP

Self-referentially short for PHP: Hypertext Preprocessor, an open source, server-side, HTML embedded scripting language used to create dynamic Web pages.

In an HTML document, PHP script (similar syntax to that of Perl or C) is enclosed within special PHP tags. Because PHP is embedded within tags, the author can jump between HTML and PHP (similar to ASP and Cold Fusion) instead of having to rely on heavy amounts of code to output HTML. And, because PHP is executed on the server, the client cannot view the PHP code.

PHP can perform any task that any CGI program can do, but its strength lies in its compatibility with many types of databases. Also, PHP can talk across networks using IMAP, SNMP, NNTP, POP3, or HTTP.

PHP was created sometime in 1994 by Rasmus Lerdorf. During mid 1997, PHP development entered the hands of other contributors.

Source: http://www.webopedia.com/

PIP (Page Inventory Page)

The Page Inventory Page (PIP) is one of the ten page types defined in InterBase/Firebird. The PIP is used along with the pointer page for space management.

Every page in the database is represented by one bit in the PIP, this bit indicating whether the page is currently in use. PIPs occur at fixed intervals in the database, the interval being determined by the database page size. PIPs are never released.

For those interested in more detailed information, Ann Harrison's article, *Space Management in InterBase*, provides an in-depth insight into page types and their roles.

See also:
Firebird for the Database Expert: Episode 2 - Page Types
TID
TIP

RDBMS (Relational Database Management System)

RDBMS is the abbreviation for Relational Database Management System and is pronounced as separate letters, a type of database management system (DBMS) that stores data in the form of related tables. Relational databases are powerful because they require few assumptions about how data is related or how it will be extracted from the database. As a result, the same database can be viewed in many different ways.

An important feature of relational systems is that a single database can be spread across several tables. This differs from flat-file databases, in which each database is self-contained in a single table. Almost all full-scale database systems are RDBMS's. Small database systems however, use other designs that provide less flexibility in posing queries.

From a technical standpoint, DBMSs can differ widely. In addition to the relational DBMS, there are also network, flat, and hierarchical DBMS's. These all refer to the way a DBMS organizes information internally. The internal organization can affect how quickly and flexibly you can extract information.

Source: http://www.webopedia.com/

Regular Expression

In computing, a regular expression (abbreviated as *regexp* or *regex*, with plural forms *regexps*, *regexes*, or *regexen*) is a string that describes or matches a set of strings, according to certain syntax rules. Regular expressions are used by many text editors and utilities to search and manipulate bodies of text based on certain patterns. Many programming languages support regular expressions for string manipulation. For example, Perl and Tcl have a powerful regular expression engine built directly into their syntax. The set of utilities (including the editor ed and the filter grep) provided by Unix distributions were the first to popularize the concept of regular expressions.

Many modern computing systems provide wildcard characters in matching filenames from a file system. This is a core capability of many command-line shells and is known as globbing. Wildcards differ from regular expressions in that they can only express very restrictive forms of alternation.

Source: http://en.wikipedia.org/

Regular expressions explained

Regular expressions look ugly for novices, but really it's a very simple (well, usually simple!), easy to handle and a powerful tool.

Some examples

Real number (e.g.`'13.88e-4'`, `'-7E2'`):

```
([+\-]?\d+(\.\d+)?([eE][+\-]?\d+)?)
```

Phone number (e.g. `'+7(812) 555-5555'`,`'(20)555-55-55'`,`'555-5555'`):

```
((\+\d *)?(\(\d{2,4}\) *)?\d{3}(-\d*)*)
```

E-mail address (e.g. `'anso@mail.ru'`,`'anso@mailbox.alkor.ru'`):

```
([_a-zA-Z\d\-\.]+@[_a-zA-Z\d\-]+(\.[_a-zA-Z\d\-]+)+)
```

Internet URL (e.g. `'http://www.paycash.ru'`,`'ftp://195.5.138.172/default.htm'`):

```
([Ff][Tt][Pp]|[Hh][Tt][Tt][Pp])://([_a-zA-Z\d\-]+(\.[_a-zA-Z\d\-]+))((/[ _a-zA-Z\d\-\\\.]+)+)*
```

Detailed explanation

Any single character matches itself, unless it is a metacharacter with a special meaning described below.

A series of characters matches that series of characters in the target string, so the pattern `bluh` would match `bluh` in the target string. Quite simple eh ?

You can cause characters that normally function as metacharacters to be interpreted literally by prefixing them with a `\`. For example, `^` match beginning of string, but `\^` match character `^`, `\\` match `\` and so on.

You can specify a character class, by enclosing a list of characters in `[]`, which will match any one character from the list. If the first character after the `[` is `^`, the class matches any character not in the list.

Within a list, the `-` character is used to specify a range, so that `a-z` represents all characters between `a` and `z`, inclusive. If you want `-` itself to be a member of a class, put it at the start or end of the list, or escape it with a backslash.

The following all specify the same class of three characters: `az]`, `[az`, and `[a\-z]`. All are different from `[a-z]`, which specifies a class containing twenty-six characters. If you want `']'` you may place it at the start of list or escape it with a backslash.

Examples of queer ranges: `[\n-\x0D]` match any of `#10,#11,#12,#13`.

`[\d-t]` match any digit, `'-'` or `'t'`. `[]-a]` match any char from `']'..'a'`.

Characters may be specified using a metacharacter syntax much like that used in `C: \n` matches a newline, `\t` a tab, `\r` a carriage return, `"\f"` a form feed, etc. More generally, `\xnn`, where `nn` is a string of hexadecimal digits, matches the character whose ASCII value is `nn`.

Finally, the `.` metacharacter matches any character except `\n` (unless you use the `/s` modifier - see below). You can specify a series of alternatives for a pattern using `|` to separate them, so that `fee|fie|foe` will match any of `fee`, `fie`, or `foe` in the target string (as would `f(e|i|o)e`). The first alternative includes everything from the last pattern delimiter `(`, `[`, or the beginning of the pattern) up to the first `|`, and the last alternative contains everything from the last `|` to the next pattern delimiter. For this reason, it's common practice to include alternatives in parentheses, to minimize confusion about where they start and end.

Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching `foo|foot` against `barefoot`, only the `foo` part will match, as that is the first alternative tried, and it successfully matches the target string. (This might not seem important, but it is important when you are capturing matched text using parentheses.)

Also remember that `|` is interpreted as a literal within square brackets, so if you write `[fee|fie|foe]` you're really only matching `[feio|]`.

The bracketing construct `( ... )` may also be used for define r.e. subexpressions (after parsing you may find subexpression positions, lengths and actual values in `MatchPos`, `MatchLen` and `Match` properties of `TRegExpr`, and substitute it in template strings by `TRegExpr.Substitute`).

Subexpressions are numbered based on the left to right order of their opening parenthesis.

The first subexpression has the number `'1'` (whole r.e. match has number `'0'` - you may substitute it in `RegExpr.Substitute` as `'$0'` or `'$&'`).

Any item of a regular expression may be followed with digits in curly brackets.

A short list of metacharacters

`^`	Start of line
`$`	End of line
`.`	Any character
`\`	Quote next character
`*`	Match zero or more
`+`	Match one or more
`{n}`	Match exactly `n` times
`{n,}`	Match at least `n` times
`{n,m}`	Match at least `n` but not more than `m` times
`[aeiou0-9]`	Match `a`, `e`, `i`, `o`, `u`, and `0` thru `9`;
`[^aeiou0-9]`	Match anything but `a`, `e`, `i`, `o`, `u`, and `0` thru `9`
`\w`	Matches an alphanumeric character (including `_`)
`\W`	A non alphanumeric
`\d`	Matches a numeric character
`\D`	A non-numeric
`\s`	Matches any space (same as `[ \t\n\r\f]`)
`\S`	A non space

You may use `\w`, `\d` and `\s` within character classes.

By default, the `^` character is only guaranteed to match at the beginning of the string, the `$` character only at the end (or before the new line at the end) and perl does certain optimizations with the assumption that the string contains only one line. Embedded newlines will not be matched by `^` or `$`.

You may, however, wish to treat a string as a multi-line buffer, such that the `^` will match after any newline within the string, and `$` will match before any newline. At the cost of a little more overhead, you can do this by using the `m` modifier on the pattern match operator.

To facilitate multi-line substitutions, the `.` character never matches a new line unless you use the `s` modifier, which in effect tells `TRegExpr` to pretend the string is a single line - even if it isn't.

List of modifiers (Note: only "i", "s" and "r" implemented)

- `i` Do case-insensitive pattern matching (using installed in your system local settings).
- `s` Treat string as single line. That is, change `.` to match any character whatsoever, even a new line, which it normally would not match. The `s` modifier without `m` will force `^` to match only at the beginning of the string and `$` to match only at the end (or just before a new line at the end) of the string. Together, as ms, they let the `.` match any character whatsoever, while yet allowing `^` and `$` to match, respectively, just after and just before new lines within the string.
- `r` Non-standard modifier.

Perl extensions

`(?imsxr-imsxr)` You may use it into r.e. for modifying modifiers by the fly, for example, `(?i)Saint-Petersburg` - will match string `'Saint-petersburg'` and `'Saint-Petersburg'`, but `(?i)Saint-(?-i)Petersburg` - will match only `'Saint-Petersburg'`.

If this construction is inlined into a subexpression, then it effects only into this subexpression

`(?i)(Saint-)?Petersburg` - will match `'Saint-petersburg'` and `'saint-petersburg'`, but `(?i)Saint-)?Petersburg` - will match `'saint-Petersburg'`, but not `'saint-petersburg'`. `(?#text)` - A comment. The text is ignored.

Source: (c) 1999 Andrey V. Sorokin, anso@mail.ru

SMP (Symmetric Multi-Processing)

Short for Symmetric Multiprocessing, a computer architecture that provides fast performance by making multiple CPUs available to complete individual processes simultaneously (multiprocessing). Unlike asymmetrical processing, any idle processor can be assigned any task, and additional CPUs can be added to improve performance and handle increased loads. A variety of specialized operating systems and hardware arrangements are available to support SMP. Specific applications can benefit from SMP if the code allows multithreading.

SMP uses a single operating system and shares common memory and disk input/output resources. Both UNIX and Windows NT support SMP.

Source: http://www.webopedia.com/

SMTP (Simple Mail Transfer Protocol)

SMTP is the de facto standard for e-mail transmissions across the Internet. SMTP is a relatively simple, text-based protocol, in which one or more recipients of a message are specified (and in most cases verified to exist) along with the message text and possibly other encoded objects. The message is then transferred to a remote server using a procedure of queries and responses between the client and server. Either an end-user's email client, a.k.a. MUA (Mail User Agent), or a relaying server's MTA (Mail Transport Agents) can act as an SMTP client.

An email client knows the outgoing mail SMTP server from its configuration. A relaying server typically determines which SMTP server to connect to by looking up the MX (Mail eXchange) DNS record for each recipient's domain name (the part of the email address to the right of the at (@) sign). Conformant MTAs (not all) fall back to a simple A record in the case of no MX. Some current mail transfer agents will also use SRV records, a more general form of MX, though these are not widely adopted. (Relaying servers can also be configured to use a smart host.)

The SMTP client initiates a TCP connection to server's port 25 (unless overridden by configuration). It is quite easy to test an SMTP server using the telnet program.

SMTP is a "push" protocol that does not allow one to "pull" messages from a remote server on demand. To do this a mail client must use POP3 or IMAP. Another SMTP server can trigger a delivery in SMTP using ETRN.

Source: http://en.wikipedia.org/wiki/Smtp

Statement

A statement is the smallest unit of a program. Statements are separated in InterBase/Firebird by a semicolon.

A statement is an instruction written in a high-level language. A statement directs the computer to perform a specified action. A single statement in a high-level language can represent several machine-language instructions. Programs consist of statements and expressions.

Source: http://www.webopedia.com/

String

A string is a series of characters manipulated as a group. A character string differs from a name in that it does not represent anything - a name stands for some other object.

A character string is often specified by enclosing the characters in single or double quotes. For example, `WASHINGTON` would be a name, but `'WASHINGTON'` and `"WASHINGTON"` would be character strings.

Source: http://www.webopedia.com/

TID (Transaction ID)

Each user performs transactions, and each transaction is given its own ID. The TID (Transaction IDs) are numbered sequentially, i.e. transaction ID 10 was started before the transaction with the ID 11.

The TIPs contain all transactional information in an array of bits, two per transaction, which indicate the state of the transaction. The transaction ID is an index into this array.

When the transaction number is allocated to a transaction, the user also receives a copy of the TIP (Transaction Inventory Page), which comprises the status of all transactions. If a data set is inserted or modified, the TID is entered next to the alteration. These simple rules are all that is needed to implement the InterBase/Firebird versioning.

A transaction can only see those transactions with a lower TID than its own. Furthermore, all other transactions that were still active at that point in time when the transaction was started, are invisible to the transaction.

The TIP copy, provided when the TID number is allocated, can be used to monitor the status of all other transactions at the point in time when the transaction was started. The only way to obtain a newer, more up-to-date TIP is to request a new TID.

For example, user A has a TID 10, user B has a TID 11 or higher. He could also have a TID 9 or lower, when his transaction was still active at the point in time when user A began his transaction with the TID 10. Otherwise he would not be able to alter the data set X. User B modifies the data set with his active transaction.

Now user A modifies data set X. When the transaction is posted, User A receives a deadlock error or an update conflict, providing the Transaction Isolation Level is set at *repeatable read*. this message informs user A that his modification cannot be carried out, as another user - in this case user B - has modified the data set. The programmer can decide at this point, how the program reacts to this situation.

TIP (Transaction Inventory Page)

The Transaction Inventory Page (TIP) is one of the ten page types defined in InterBase/Firebird. Each and every user transaction is consecutively numbered, using the InterBase/Firebird Transactions Inventory Page (TIP) (also known as the Transaction Information Page). These transaction numbers are used by the InterBase/Firebird versioning engine to ensure that users always receive a consistent view of the database. It shows the status of each and every transaction in the database, and adheres to two main rules:

1. Only those transactions are visible, whose `ID <= own ID`.
2. Only those transactions are visible, which were already committed at the time the own transaction was started.

Transactions are shown with one of the following four status values:

Table: Values in the Transaction Information Pages

Status Code	Description
A	Transaction is active, or in process
C	Transaction was committed. The changes made by this transaction can be applied if necessary to show a consistent view of the database.
R	Transaction was rolled back. The changes made by this transaction should be ignored.
L	Limbo transaction. This transaction was part of an operation involving more than one database within an embedded SQL application.

For example, `1C` = first transaction committed, `2A` = second transaction is active, `3C` = third transaction is rolled back, `4L` = Transaction is in limbo (i.e. when a transaction is dependent upon another transaction in another database = two-phase commit). This information is important for the garbage collection.

The TIPs contain this information in an array of bits, two per transaction, that indicate the state of the transaction. The transaction ID (TID) is an index into this array.

Special transactions IDs

InterBase/Firebird tracks three special positions within the transaction history:

1. The Oldest Interesting Transaction (OIT) is the earliest transaction in the database with a status other than committed. Every transaction prior to that one represents an unbroken chain of insertions and updates into the database.
2. The Oldest Active Transaction (OAT) is the earliest transaction in the database that is currently active or open.
3. The Next Transaction Number is the ID that is used for the next transaction that starts.

You can find these numbers in the IBExpert Database Statistics display within the Server Manager, or using the gstat `-h` command in isql.

When you start a transaction, InterBase/Firebird makes a copy of the TIP into the server memory cache assigned to your process, starting from the page holding the OIT and finishing with the page holding the OAT.

Whenever the database is backed up and restored, the transaction inventory is wiped out and the next transaction number is set to 1.

There is also a mechanism in the InterBase/Firebird server TIP page, to allow a local TIP page for each user. The local TIP page is generated the minute a new user presses the Execute [F9] key. Please refer to TID (Transaction ID) for further information.

The advantage of such a system is that older records are held ready. The disadvantage for users, who execute, but need a considerable time before finally committing is that the local TIP becomes very large, as it always begins at the oldest active transaction, so that it is possible using this technique, for one transaction to hold everything up and slow the transaction processing for everyone. If a system becomes increasingly slow with time, it is almost always due to the fact that TIP pages are being filled further and further with transaction information, because the first transaction has not been committed. 99% of local TIPs are held in the RAM, until there are no further pages free.

- *Note:* If you are only doing a `SELECT` in your transaction, you should always `COMMIT` to avoid creating an "interesting" transaction (transaction with a status code other than committed in the TIP).

All TIPs are of the page size defined when creating the database. 16,000 transactions fit, for example, onto a 4K page.

TIPs and Server Crashes

If a server crashes or hangs during user transactions, the InterBase/Firebird server simply looks at the TIP, and rolls back all operations that were still active. This means that an InterBase/Firebird server can be rapidly restarted. As soon as the operating system is up and running, InterBase/Firebird is also up and running. Forced writes however influence the sequence in which is written:

1. IBExpert Database Properties / Forced Writes - when committing InterBase/Firebird saves all data sets to the hard drive and then to the TIP.
2. Without forced writes the process is minimally quicker, but on a Windows platform, Windows decides what should be saved to file, where and when; and the data pages are saved to file last i.e. the TIP changes are written first and then the data sets, which could possibly lead to inconsistencies.

Therefore forced writes are extremely important when working on a Windows platform. Without forced writes, the computer needs to be extremely secure.

Transaction

A transaction is a single task with a number of specific characteristics. An application can perform one or more operations, within the context of a transaction, each of which must be completed in sequence.

One of the main tools used by relational databases to maintain data integrity is the transaction. A transaction is a task with a number of specific characteristics:

1. An application can perform one or more operations within the context of a transaction, each of which must be completed in sequence. An operation consists of, as a rule, one SQL statement, such as `SELECT`, `INSERT`, `UPDATE`, or `DELETE`.
2. The changes performed by the transaction can be committed if all of the operations in the transaction are completed. Until the results of the transaction are committed, the changes made to the database are invisible to other users.
3. A transaction can also be rolled back. In this case, as far as other database users are concerned, the data never changed.

Because of these characteristics, transactions ensure that complex operations on the database are performed completely. Transactions provide complete protection against operations not being completely processed, therefore ensuring data integrity.

A transaction can be in one of the following four states:

1. in limbo
2. Committed
3. Rolled back
4. Active

Transaction Mask

A transaction mask is an array of two bit pairs that represents the state of all transactions starting with the oldest interesting and ending with the next transaction. The oldest interesting transaction is the first transaction in the database after transaction zero) whose state is not committed. Transaction zero is the system transaction and is always active. The next transaction is the transaction after the one that started most recently.

In the Classic architecture, each connection maintains its own copy of the transaction mask. In shared server architectures, each server maintains a single copy of the transaction mask. In Classic, and in particular on machines with memory sizes that were typical in the early 90's, you could eat up a lot of memory describing a system that had a few hundred thousand transactions between the oldest interesting and the next transaction, even if you only use two bits per transaction.

Transaction Number Column

For every table you create, including system tables, InterBase/Firebird maintains an extra column for the transaction number. When you insert or update a column as part of a transaction, the transaction number is written to this column, so that InterBase/Firebird knows which transaction is controlling that row of the table. Even when you delete a row as part of the transaction, the number is written to the row until the transaction is committed or rolled back, in case there is a problem, or in case the transaction is a lengthy one.

The InterBase/Firebird versioning engine uses this transaction number to ensure that each user receives a consistent view of the database at a moment in time. This is known as a *repeatable read*.

Active Transactions

A transaction is active, if one of the following conditions is true:

- The transaction has not yet started.
- The transaction has started but not yet completed.
- The transaction has started, could not however complete successfully, due to for example, a system crash or communication problems etc.

The actual status of each transaction is recorded in the TIP (Transaction Inventory Page). In fact, the only alteration that occurs when a transaction is committed is the alteration to the status in the TIP from active to committed.

Transactions in Limbo

InterBase/Firebird's transaction mechanism, like most databases, can only handle transactions within a single database. However within an embedded SQL application, InterBase/Firebird can perform operations on more than one database at a time.

With a logical transaction that spans databases, InterBase/Firebird handles the operations within each database as separate transactions, and sequences them using a two-phase commit model, to ensure that both transactions complete or that neither completes. When InterBase/Firebird is ready to commit or rollback such a multidatabase transaction, it first changes the transaction status from active to limbo. It then performs the commit or rollback operation. Finally the transaction status is changed from limbo to committed.

Transactions in limbo are transactions that have been started by the `PREPARE` command within the framework of a two-phase commit. The transaction may or may not still be running. This transaction may become relevant at any point in time and all changes made so far may be committed or rolled back. Such

alterations made by such transactions can neither be examined or ignored; they can neither be defined as executed or aborted. They can therefore not simply be removed from the database.

However for a database backup to be fully performed without aborting, such transactions in limbo need to be ignored in the backup. Only those most recent, committed transactions are backed up. It allows a database to be backed up, before recovering corrupted transactions. Generally in limbo transactions should be recovered before a backup is performed.

Note: BDE clients use only single-database transactions, even if the client application accesses two or more databases. Embedded SQL? and InterBase/Firebird API provide methods for programming distributed transactions.

See also:
Firebird for the database expert: Episode 4 - OAT, OIT and Sweep

Two-Phase Commit

A transaction spanning multiple InterBase/Firebird databases is automatically committed in two phases. A two-phase commit guarantees that the transaction updates either all of the databases involved or none of them - data is never partially updated.

In the first phase of a two-phase commit, InterBase/Firebird prepares each database for the commit by writing the changes from each subtransaction to the database. This subtransaction is the part of a multi-database transaction that involves only one database. In the second phase, InterBase marks each subtransaction as committed in the order that it was prepared.

If a two-phase commit fails during the second phase, some subtransactions are committed and others are not. A two-phase commit can fail if a network interruption or disk crash makes one or more databases unavailable. Failure of a two-phase commit causes in limbo transactions, i.e. transactions that the server does not know whether to commit or roll back.

It is possible that some records in a database are inaccessible due to their association with a transaction that is in a limbo state.

Note: The Borland Database Engine (BDE), as of version 4.5, does not exercise the two-phase commit or distributed transactions capabilities of InterBase/Firebird, therefore applications using the BDE never create limbo transactions.

Variable

A symbol or name that stands for a value. For example, in the expression

```
x+y
```

x and y are variables. Variables can represent numeric values, characters, character strings, or memory addresses.

Variables play an important role in computer programming because they enable programmers to write flexible programs. Rather than entering data directly into a program, a programmer can use variables to represent the data. Then, when the program is executed, the variables are replaced with real data. This makes it possible for the same program to process different sets of data.

Every variable has a name, called the variable name, and a datatype. A variable's datatype indicates what sort of value the variable represents, such as whether it is an integer, a floating-point number, or a character.

The opposite of a variable is a constant. Constants are values that never change. Because of their inflexibility, constants are used less often than variables in programming.

Source: http://www.webopedia.com/

WAL (Write Ahead Log)

WAL (Write Ahead Log)

The Log Information Pages (LIP) for the write-ahead log are not currently used, though code to use them is included conditionally in Firebird.