IBExpert KG

An Introduction to IBExpert & Firebird

An Introduction to IBExpert & Firebird

Holger Klemt

Debra J. Miles

An Introduction to IBExpert and Firebird

Copyright   IBExpert KG

First edition: September 2009

Printed and bound in the Federal Republic of Germany for IBExpert KG, Holger Klemt, Im Gewerbepark 8, 27798 Hude, Germany.

www.ibexpert.com

Authors: Holger Klemt, Debra J. Miles

Cover design: Debra J. Miles

Published and printed by IBExpert KG, Hude, Germany.

Contents

Chapter 1: Install Firebird....................11

1. Installation using the Firebird Installer....................12
2. ZIP installation....................14
3. Performing a client-only install....................14
4. Install Firebird as an application....................15
5. Installing multiple Firebird servers....................15

Chapter 2: Install IBExpert....................17

1. Download and install IBExpert on Windows....................17
2. Installation....................17
3. Installing IBExpert under Linux....................18

Chapter 3: Register a Firebird database in IBExpert....................21

Chapter 4: Programming the Firebird Server....................25

Chapter 5: An Introduction to Stored Procedures....................27

1. INITALL....................28

Chapter 6: Basic SQL Commands....................31

1. Simple SELECT commands....................31
2. Adding a WHERE clause....................33
3. CONTAINING 34
4. ORDER BY....................34
5. SELECT across multiple tables....................34
6. Sub-SELECTs in fields and WHERE clauses....................35
7. UNION SELECT....................35
8. IN operator....................36
9. EXISTS operator....................36
10. INSERT and UPDATE with values....................37
11. DELETE....................37
12. CREATE, ALTER and DROP....................38
13. Defining code templates in IBExpert....................38
14. Analyzing SQL Performance....................38
15. Optimizing SQL statements....................40

Chapter 7: Creating your First Database....................43

1. Developing a data model....................43
2. Naming conventions....................43
3. Relationships....................44
4. 1:1....................44
5. n:1....................45
6. n:m....................45
7. Data modeling using IBExpert's Database Designer....................45
8. Create database....................46

Chapter 8: Database Designer....................49

1. Model Options....................50
2. Export....................51
3. Print....................51

Chapter 9: Create a Database Object....................53

1. Database objects....................53
2. Adding a new table to the db1 demo database....................53

3. Primary key....53
4. Foreign key....54
5. Create test data....56
6. 253 changes of table left....56

Chapter 10: More about Database Objects 59
1. Domain....59
2. Table....60
3. View....60
4. Stored Procedure....62
5. Trigger....65
6. Generator/Sequence....66
7. Exception....67
8. UDF (User-Defined Function)....67
9. Blob filter....67
10. Role....68
11. Index....68

Chapter 11: More about Data Types....73
1. Blob - Binary Large OBject 73
2. CHAR and VARCHAR 74
3. INTEGER, SMALL INTEGER and BIG INTEGER 75
4. FLOAT and DOUBLE PRECISION 76
5. NUMERIC and DECIMAL 76
6. DATE, TIME, TIMESTAMP 77
7. Array 78
8. Boolean 79
9. NOT NULL, NULL 80

Chapter 12: Writing Stored Procedures & Triggers....81
1. SET TERM....81
2. Stored procedure....83
3. Simple procedures....83
4. FOR EXECUTE ... DO 85
5. WHILE ... DO....87
6. LEAVE and BREAK....87
7. EXECUTE statement....88
8. Recursions and modularity....88
9. Debugging....89
10. Optimizing procedures....91
11. Complex SELECTs or selectable stored procedures?....93
12. Trigger....94
13. Using procedures to create and drop triggers....94
14. Using domains in stored procedures....95

Chapter 13: SP/Triggers/Views Analyzer....97

Chapter 14: Writing Exceptions....101

Chapter 15: Logging....103
1. Understanding the log file....104

Chapter 16: Database Backup & Restore....107
1. Why is a database backup and restore important?....107
2. Backup Database....108
3. Restore Database....110
4. Automating the database backup and restore....114

Chapter 17: Reporting....115
1. ibec_CreateReport....116

2. ibec_ExportReport....117
3. Job Automation with the IBExpertJobScheduler....119
Chapter 18: Data Analysis....121
1. Cube Structure....121
2. Cube....122
3. Data Analysis Cube Manager....123
4. Data Analysis Calculated Measures Manager....124
Chapter 19: Data Export & Import....127
1. Export....127
2. Import 129
3. Secure data transfer....132
Chapter 20: Data & Metadata Manipulation....133
1. Extract Metadata....133
2. Search Metadata....139
3. SQL Editor Special Features....140
4. Copy Database Object....142
5. Database Comparer & Table Data Comparer....142
Chapter 21: Script Executive & Script Language Extensions....147
1. Executing multiple scripts from a single script....148
2. Create multiple CSV files from a script....148
3. Script Language Extensions....149
Chapter 22: Firebird 2.0 Blocks....151
Chapter 23: IBEBlock....157
Chapter 24: Database Documentation....165
1. Object and field descriptions....165
2. Template short cuts....165
3. Print metadata....165
4. Generate HTML documentation....167
Chapter 25: User Manager....169
1. Users page....169
2. Password....170
3. Roles page....170
4. Membership page....170
Chapter 26: Grant Manager....171
1. Granting access to stored procedures....172
2. Using the GRANT AUTHORITY option....172
Chapter 27: Database Statistics....173
Chapter 28: Optimizing Database Performance....177
1. Operating systems....177
2. Optimal hard disk use....177
3. Optimizing hardware configuration....178
4. Temporary files....178
5. Memory configuration....178
6. Optimizing OS configuration....179
7. Firebird benchmarks tests....179
8. Optimizing the database....179
9. Parameters for optimal performance....180
10. The Firebird Optimizer and index statistics....180
11. Automating the recalculation of index statistics....180
Chapter 29: Avoiding Server Problems....183

1. Typical causes of server problems....183
2. Detect and avoid server problems....184
3. Communication Diagnostics....184
Chapter 30: Command-Line Utilities....187
1. IBExpert's IBEScript.exe....187
2. IBEScript.dll....188
3. Firebird/InterBase Command-Line Tools....189

Foreword

If you develop SQL databases, professionally or as a hobby, and need an efficient and powerful tool, you cannot go wrong with IBExpert. It 4enables you in just a short space of time to become acquainted with and achieve a command of the open source database, Firebird, as well as its commercial relative, InterBase. There are powerful and yet easy-to-learn editors for all vital functions. Development of database objects, database models, stored procedure and trigger programming, performance tuning -all th8is and much more can be executed simply and quickly using IBExpert.

The Firebird server is an extremely powerful open source database system,which in spite of its very simple installation and administration, offers all essential functions otherwi8se found only in commercial database systems such as Oracle or Informix. However, following installation the Firebird user has but a few command-line tools at his disposal, there is no powerful GUI tool for data definition and administration included in the kit. This, along with the very limited document-ation, is the first hurdle that Firebird users need to overcome.

This is where IBExpert comes to the rescue, whether in the form of the free, functionally limited Personal Edition, the gratis Educational Version or the commercial Full Version including all modules, which is also available as a 45-day Trial Version. These resources enable the user the acquire the technical proficiency required for professional applications.

This guide is intended as an introduction for both database developers and administrators, enabling …

Thanks to … (Jason Chapman, Paul Beach, Firebird Foundation, ...)

Following the success of open source operating systems, particularly in the area of server installa-tions, the database market segment is now also on the brink of such a breakthrough. Firebird is here certainly one of the most powerful platforms, and IBExpert the ideal complement for discerning database developers and administrators.

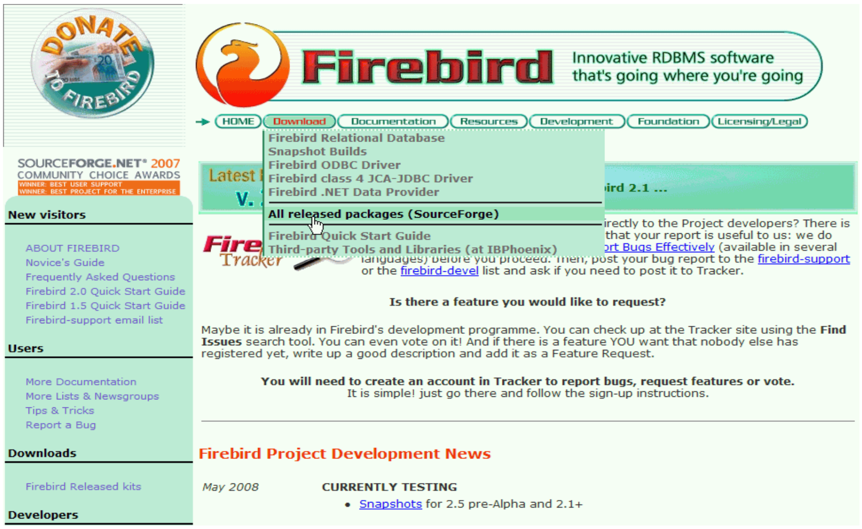
Oldenburg, Germany

September 2009

Chapter 1: 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 few mouse clicks. Firebird offers two server versions, Classic and Superserver. If you are installing Firebird for the first time we recommend installing the Superserver.

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:


DONATE TO FIREBIRD
Firebird
Innovative RDBMS software that's going where you're going
HOME
Download
Documentation
Resources
Development
Foundation
Licensing/Legal
Firebird Relational Database
Snapshot Builds
Firebird ODBC Driver
Firebird class 4 JCA-JDBC Driver
Firebird .NET Data Provider
All released packages (SourceForge)
Firebird Quick Start Guide
Third-party Tools and Libraries (at IBPhoenix)
SOURCEFORGE.NET® 2007 COMMUNITY CHOICE AWARDS
WINNER: BEST USER SUPPORT
WINNER: BEST PROJECT FOR THE ENTERPRISE
New visitors
ABOUT FIREBIRD
Novice's Guide
Frequently Asked Questions
Firebird 2.0 Quick Start Guide
Firebird 1.5 Quick Start Guide
Firebird-support email list
Users
More Documentation
More Lists & Newsgroups
Tips & Tricks
Report a Bug
Downloads
Firebird Released kits
Developers
Latest
V.
ird 2.1 ...
Firebird Tracker

irectly to the Project developers? There is that your report is useful to us: we do ort Bugs Effectively (available in several languages) before you proceed. Then, post your bug report to the firebird-support or the firebird-devel list and ask if you need to post it to Tracker.

Is there a feature you would like to request?

Maybe it is already in Firebird's development programme. You can check up at the Tracker site using the **Find Issues** search tool. You can even vote on it! And if there is a feature YOU want that nobody else has registered yet, write up a good description and add it as a Feature Request.

You will need to create an account in Tracker to report bugs, request features or vote.
It is simple! just go there and follow the sign-up instructions.

Firebird Project Development News

May 2008 **CURRENTLY TESTING**

- Snapshots for 2.5 pre-Alpha and 2.1+

Simply click the *Download* tab and select *All released packages (SourceForge)*. 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. 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 opens: click on the green *Download* button to the right of the Firebird file you require. Select the file(s) you wish to download and then, if required, 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.

1. Installation using the Firebird Installer

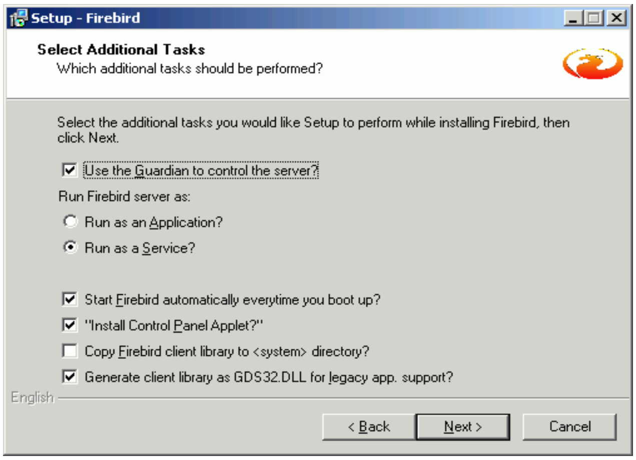
Now double-click the downloaded Firebird file to start the installation. Again, please refer to *Windows Platforms* and *Posix Platforms* on the following pages for installation details for the various platforms.

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:


Setup - Firebird

Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Firebird, then click Next.

[✓] Use the Guardian to control the server?

Run Firebird server as:

( ) Run as an Application?

(●) Run as a Service?

[✓] Start Firebird automatically everytime you boot up?

[✓] "Install Control Panel Applet?"

[ ] Copy Firebird client library to <system> directory?

[✓] Generate client library as GDS32.DLL for legacy app. support?

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 the 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 any 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*, part of the Firebird software package.

The *IBExpertInstanceManager* service creates a replacement for the Firebird Guardian, which is important if you have more than one Firebird server installed, because the Firebird Guardian only works with the Firebird default instance. Further information regarding the IBExpertInstanceManager can be found in *Chapter 1: 15. Installing multiple Firebird servers.*

1.1 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 the Guardian. Should the server application terminate abnormally, the Guardian will attempt to restart it.

1.2 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 also refer to *Firebird 2 Migration & Installation: Installing on POSIX platforms* at www.ibexpert.com/doc and 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. Shell scripts have been provided, but in some cases, the release notes may advise modification of the scripts as well as some manual adjustments.

2. ZIP installation

Another way to install Firebird is from a ZIP file. This method is more flexible for embedded installations, and 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 appropriate batch file. Download the appropriate 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.

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:

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 concludes by starting the service.

3. 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` or `fbclient.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:

3.1 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.

3.2 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` 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.

Since Firebird 2.1 the Installer offers the possibility to install multiple instances. IBExpert has its own *IBExpertInstanceManager* as one of the HK-Software Services Control Center services (see *5. Installing multiple Firebird servers* below).

4. Install Firebird as an application

To run Firebird as an application, use the following parameter -a:

```
C:\Program Files\Firebird\Firebird_2_1\bin>fbserver –a
```

This can, for example, be copied to any subdirectory of your application and controlled from the application so that when it starts, the Firebird server also starts. Furthermore it is possible, for example, directly specify the use of a different port. That way files just need to be added to your individual setup, with the `firebird.conf` file port specification adjusted accordingly. It is not advisable to use port 3050, the default Firebird port, because it is used by every other Firebird server. If you leave it on 3050 you may encounter problems if other Firebird installations are present.

When you are starting the Firebird server as an application, you do not need to install anything. Simply copy the data to the customer's workgroup server and start it from there.

5. Installing multiple Firebird servers

IBExpertInstanceManager is one of the modules in the *HK-Software Services Control Center* and and *IBExpert Server Tools* and can be used to install several instances of the Firebird server on a single Windows machine using different ports.

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. You can even create one instance per database if you wish.

5.1 How to specify instances

First ensure that there is already a single Firebird version installed on your machine using the default Firebird installer. A current IBExpert full version should also be installed. The service, `hkIM.exe`, can be found in the *IBExpertInstanceManager* directory. This service also creates a replacement for the Firebird Guardian, which is important because the Firebird Guardian only works with the Firebird default instance.

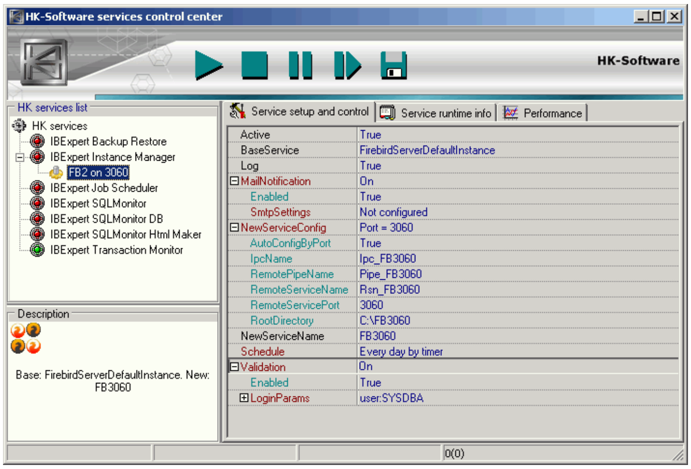
In IBExpert this can be found in the IBExpert *Services* menu, *HK-Software Services Control Center*.

The default settings include those options which can be user-specified for all individual instances.

To create an instance right-click on the *IBExpertInstanceManager* service and select *Add task*. Click on this task on the left, set the task's *Active* parameter to *True*, and then select the *BaseService* from the list of Firebird instances installed on the PC.

When the *FirebirdServerDefaultInstance* is selected (if you are creating your first instance this will be the only option), the necessary information will be copied from the Firebird version just installed. Simply specify the port number for the Firebird instance to be created. All other instance configuration settings will be generated automatically. There are further options to set up mail notification, schedules and validation parameters. Validation is simply a test connection to the new instance's `security.fdb/security2.fdb`, using the instance's port number. The SYSDBA password can be different for each instance, if wished.

To rename the task, click on the task name when the service is inactive with the [Ctrl] key pressed down. Specifications can be saved and the service can be started (or restarted if it was already running). When properly configured the running task should show runtime info on the first run. This can be viewed on the *Service runtime info* page.



HK services
- IBExpert Backup Restore
- IBExpert Instance Manager
  - FB2 on 3060
- IBExpert Job Scheduler
- IBExpert SQLMonitor
- IBExpert SQLMonitor DB
- IBExpert SQLMonitor Html Maker
- IBExpert Transaction Monitor

Active	True
BaseService	FirebirdServerDefaultInstance
Log	True
⊟MailNotification	On
Enabled	True
SmtpSettings	Not configured
⊟NewServiceConfig	Port = 3060
AutoConfigByPort	True
IpcName	Ipc_FB3060
RemotePipeName	Pipe_FB3060
RemoteServiceName	Rsn_FB3060
RemoteServicePort	3060
RootDirectory	C:\FB3060
NewServiceName	FB3060
Schedule	Every day by timer
⊟Validation	On
Enabled	True
⊞LoginParams	user:SYSDBA

Base: FirebirdServerDefaultInstance. New: FB3060

A new directory `C:\FB3060` has automatically been created by IBExpert. All files from the original Firebird installation have been copied into this directory. The `firebird.conf` has been manipulated to meet the specifications made in the *IBExpertInstanceManager* (`RootDirectory`, `RemoteServiceName`, `RemoteServicePort`, `IpcName`, `RemotePipeName`).

After the instance, which now runs from this new directory, has been successfully created, the original Firebird server should be uninstalled.

The next Firebird version can then be installed and the above procedure repeated to create a second instance. The new Firebird installation does not recognize the presence of the first Firebird instance as a Firebird server, as it's running on a different port and in a different directory. Therefore this newly installed Firebird version will also be the `FirebirdServerDefaultInstance`.

In this way it is possible to create as many instances as may be required without any conflicts.

Chapter 2: Install IBExpert

1. Download and install IBExpert on Windows

1.1 Customer Version

IBExpert can be downloaded from the IBExpert download pages. There are a number of versions - please refer to the IBExpert website at www.ibexpert.com for further information.

All registered databases are stored in the directory, `C:\Documents and Settings\%user%\Applicationdata\HK-Software\IBExpert` or, if used, in the IBExpert *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 current IBExpert version can be found by scrolling down to `setup_customer.exe`: these files comprise the full range of all IBExpert Developer Studio tools and components.

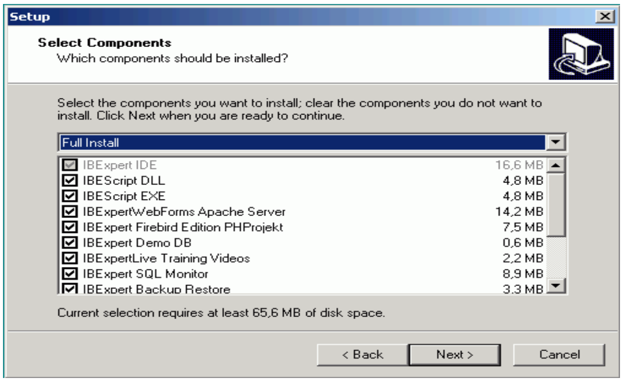
Customers installing their first fully licensed IBExpert customer version will be asked to register the product the first time the application is started. Please check that the computer name and company name which appears in the *Registration* window is the same as the computer name and company name quoted on the 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.

1.2 Trial Version

For those wishing to download the IBExpert Trial Version, go to the *Download / Trial* page at www.ibexpert.com and click *Download* to access the download area. The file you require is `setup_trial.exe` file.

2. Installation

Double-click the `EXE` file to start the installation. The IBExpert Customer and Trial versions both offer the full selection of all IBExpert Developer Studio Tools:


Setup

Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full Install

✓	IBExpert IDE	16,6 MB
✓	IBEScript DLL	4,8 MB
✓	IBEScript EXE	4,8 MB
✓	IBExpertWebForms Apache Server	14,2 MB
✓	IBExpert Firebird Edition PHProjekt	7,5 MB
✓	IBExpert Demo DB	0,6 MB
✓	IBExpertLive Training Videos	2,2 MB
✓	IBExpert SQL Monitor	8,9 MB
✓	IBExpert Backup Restore	3,3 MB

Current selection requires at least 65,6 MB of disk space.

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*.

3. Installing IBExpert under Linux

As the many Linux distributions vary widely, the following is limited to a detailed IBExpert installation under ubuntu 8.1.0. It may be necessary to make certain adjustments when installing on other Linux distributions.

3.1 Install Wine

You will need to open a shell to install Wine (the graphical interface cannot be used because you need to be able to log in as root to install these tools). Run the installation as root or through `kdesu` or `sudo` programs. This article uses `sudo` commands in its examples.

Open the *Konsole* (found under *Applications/System Tools*), and log in as `superuser`:

```
sudo su
```

entering the password when prompted (`[sudo] password for xxx`:).

Firstly you need to download the newest Wine version which can be found at: http://winehq.org/site/download. At http://winehq.org/site/download_deb you can find the most up-to-date version for Debian derivatives, including ubuntu.

Using ubuntu 8.1.0 the following command automatically adds the newest Wine version to the sources:

```
sudo wget
http://wine.budgetdedicated.com/apt/sources.list.d/hardy.list -O
/etc/apt/sources.list.d/winehq.list
```

Then you simply need to enter:

```
sudo apt-get install wine & sudo apt-get update
```

to install the newest version. Don't run IBExpert before doing the next steps. If you have done, you will probably need to delete the `.wine` directory.

Upon completion of the installation enter:

```
        winecfg
```

This will open a configuration dialog, which can immediately be closed again. This command automatically creates a `.wine` folder in the `Home` directory. The next step entails the execution of the following two commands which run a script in order to obtain a native DCOM98:

```
        wget http://kegel.com/wine/winetricks
```

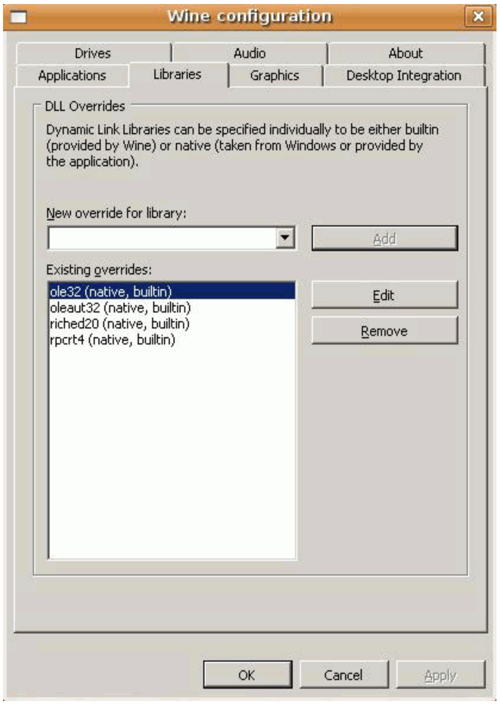
and

```
        sh winetricks dcom98
```

(alternative site: http://wiki.winehq.org/NativeDcom)

Now both the `msls31.dll` and the `riched20.dll` need to be copied into the `.wine/drive_c/windows/system32` directory. These files can be found, for example, in a Windows system.

Finally the following needs to be added to the `windcfg` file: `riched20.dll` should be entered on the *Libraries* page under *New override for library:*. Click on the *Add* button and you should see `riched20.dll` appear in the list below:



Dynamic Link Libraries can be specified individually to be either builtin (provided by Wine) or native (taken from Windows or provided by the application).

New override for library:

Existing overrides:

ole32 (native, builtin)
oleaut32 (native, builtin)
riched20 (native, builtin)
rpcrt4 (native, builtin)

You should now be able to run most Windows applications.

3.2 Install IBExpert under Wine

Before installing IBExpert, open the Wine configuration and, on the *Applications* page, select *Windows 98* from the *Windows Version* list. This is only necessary for the IBExpert installation, and can be changed back immediately to the Windows version of your choice as soon as IBExpert has been installed.

Now you need to enter the command:

```
wine <IBExpert InstallationFile.exe>
```

to install IBExpert. The installation procedure runs in exactly the same way as described for Windows (refer to the beginning of this chapter).

Upon completion of the installation, you only need to make one adjustment. Under the IBExpert *Options* menu item, *Environment Options,* you need to specify the *Default Client Library* path to the `fbclient.dll` or `gds32.dll`. This can be found either in a Windows installation or the Windows Server Version installed with Wine without the developer and server components (if you're not sure which client library you need, install both) under: `~/.wine/drive_c/windows/system32`. Please note that all names and extensions must be written in lower case.

Then it only remains to reset the *Windows Version* in the Wine configuration to the version of your choice and IBExpert can now connect to any Firebird (or InterBase) server.

3.3 Install Firebird under Linux

If you are not accessing remotely to a Firebird database already installed on another machine, you will need to install Firebird locally on your own computer. Firebird 2.x (Superserver for Linux x86 as a compressed tarball) can be downloaded from the official Firebird website: http://firebirdsql.org. Then go to the *Download* directory and extract the package using:

```
tar -xf FirebirdSS-2.0.1.*
```

Now go to the extracted directory and install the server as root:

```
sudo sh install.sh
```

You will of course need a directory to store the databases. The example below uses `/srv/firebird`:

```
sudo mkdir /srv/firebird
sudo chown firebird:firebird /srv/firebird
```

In order to connect from the local machine to the server, you will need to specify the following in IBExpert in the *Database Registration*:

```
server: remote
servername:localhost (oder 127.0.0.1)
```

Chapter 3: Register a Firebird database in IBExpert

For the purpose of illustrating the many Firebird and IBExpert features featured in this book, we will use the demo database, DB1, which can be found as an SQL script, `db1.sql`, in the IBExpert Developer Studio directory, `\IBExpertDemoDB`.

After starting IBExpert, open the *Tools* menu item, *Script Executive*, or alternatively use the key shortcut [Ctrl + F12]. The Script Executive can be used to execute any valid script. This also allows you to execute scripts with more than one command. If you wish to follow the demonstration below, type [Ctrl + L] to open the following demo script:

```
 C:\program files\HK-Software\IBExpert Developer Studio\
IBExpertDemoDB\db1.sql.
```

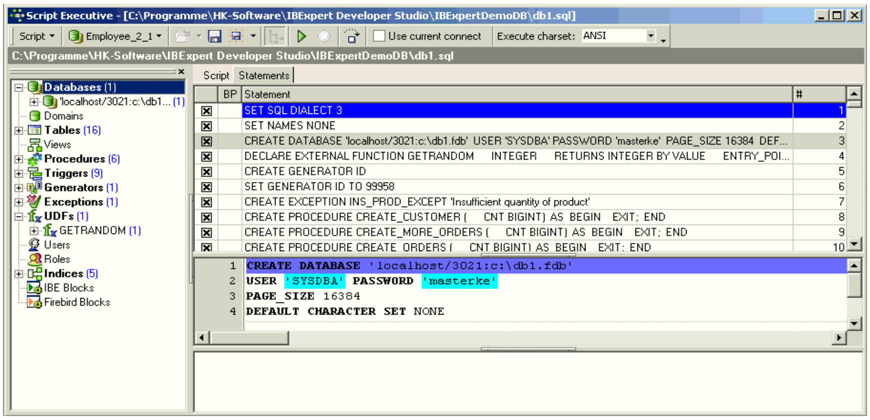
Important: To use this script, it is necessary to copy the `dll` files from this directory to the Firebird or InterBase UDF directory `\program files\firebird\firebird_2_1\UDF\`, otherwise the script will not work properly. To use the script with InterBase, replace

```
select first ...
```

statements with:

```
select .... rows ..
```

statements. When using Firebird, no alterations are required.



Script Executive - [C:\Programme\HK-Software\IBExpert Developer Studio\IBExpertDemoDB\db1.sql]

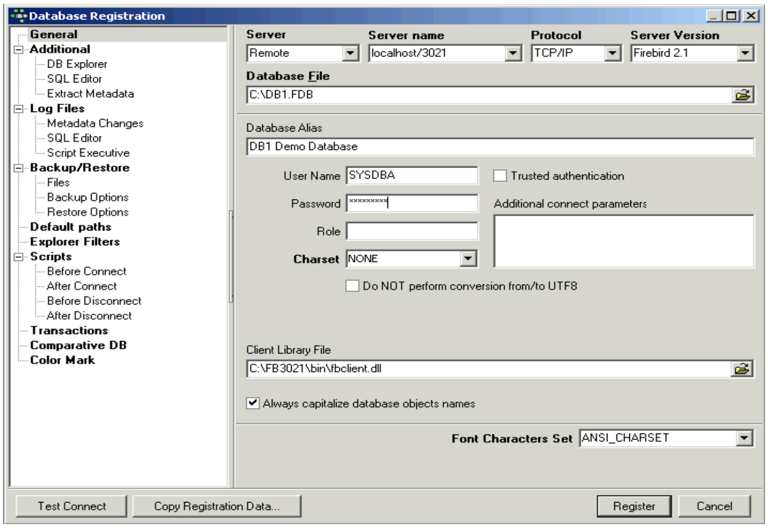
	BP	Statement	#
☒		SET SQL DIALECT 3	1
☒		SET NAMES NONE	2
☒		CREATE DATABASE 'localhost/3021:c:\db1.fdb' USER 'SYSDBA' PASSWORD 'masterke' PAGE_SIZE 16384 DEF...	3
☒		DECLARE EXTERNAL FUNCTION GETRANDOM INTEGER RETURNS INTEGER BY VALUE ENTRY_POI...	4
☒		CREATE GENERATOR ID	5
☒		SET GENERATOR ID TO 99958	6
☒		CREATE EXCEPTION INS_PROD_EXCEPT 'Insufficient quantity of product'	7
☒		CREATE PROCEDURE CREATE_CUSTOMER (CNT BIGINT) AS BEGIN EXIT; END	8
☒		CREATE PROCEDURE CREATE_MORE_ORDERS (CNT BIGINT) AS BEGIN EXIT; END	9
☒		CREATE PROCEDURE CREATE_ORDERS (CNT BIGINT) AS BEGIN EXIT; END	10

```
CREATE DATABASE 'localhost/3021:c:\db1.fdb'
USER 'SYSDBA' PASSWORD 'masterke'
PAGE_SIZE 16384
DEFAULT CHARACTER SET NONE
```

After the script has been loaded into the *Script Executive*, the tree view on the left displays the content created by the script in a tree structure; the right-hand window displays all statements required to create a database (more about SQL statements in *Chapter 6, Basic SQL Commands*). You can now execute this script using the [F9] key or the green arrow icon.

This script has created a database with table structure commensurate to a simple DVD store. It can be used to generate a classic database with a large amount of test data for a commercial environment.

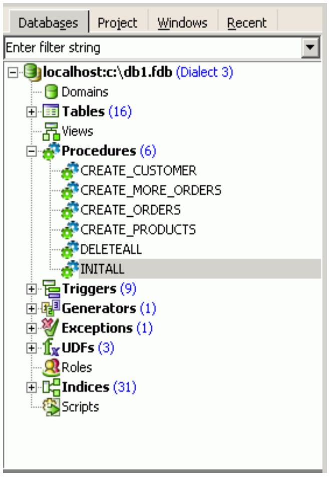
After executing the script, the majority of tables are initially still empty, except for some `tmp_*` tables, which are used for creating sample data. To fill the demo database from these `tmp` tables or indeed, perform any sort of work on this database, you first need to register the it in IBExpert, using the *Database* menu item, *Register Database*.



Server: Remote
Server name: localhost/3021
Protocol: TCP/IP
Server Version: Firebird 2.1
Database File: C:\DB1.FDB
Database Alias: DB1 Demo Database
User Name: SYSDBA
Trusted authentication: ☐
Password: *********
Additional connect parameters:
Role:
Charset: NONE
☐ Do NOT perform conversion from/to UTF8
Client Library File: C:\FB3021\bin\fbclient.dll
☑ Always capitalize database objects names
Font Characters Set: ANSI_CHARSET

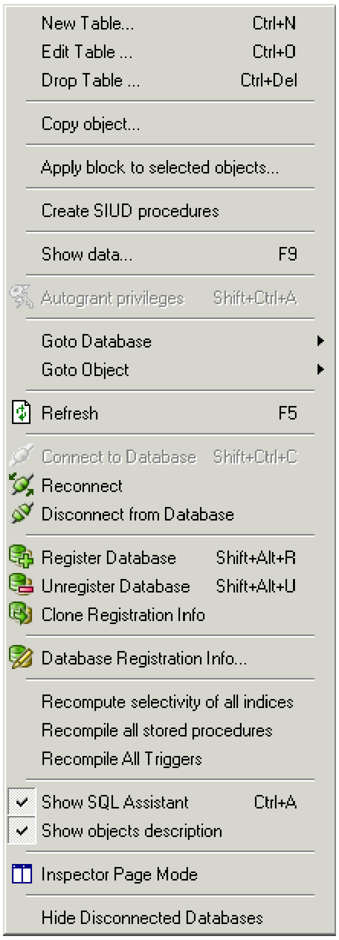
In the database registration dialog you should use the settings as illustrated above. The default password for the user SYSDBA on a newly installed Firebird server is `masterkey`.

After registering the database in IBExpert, you can open the it by double-clicking on the database alias name in the *Database Explorer,* the main navigator, considerably simplifying the work with InterBase/ Firebird databases and database objects.:



localhost:c:\db1.fdb (Dialect 3)
Domains
Tables (16)
Views
Procedures (6)
CREATE_CUSTOMER
CREATE_MORE_ORDERS
CREATE_ORDERS
CREATE_PRODUCTS
DELETEALL
INITALL
Triggers (9)
Generators (1)
Exceptions (1)
UDFs (3)
Roles
Indices (31)
Scripts

The individual database objects may be opened or closed by double-clicking (or using the space bar) on the object name. Details concerning a selected database or database object can be viewed in the *SQL Assistant* directly below the Database Explorer. In addition to the main menus at the top of the screen, many IBExpert modules - including the Database Explorer - have their own right-click context-sensitive menu.


New Table... Ctrl+N
Edit Table ... Ctrl+O
Drop Table ... Ctrl+Del
Copy object...
Apply block to selected objects...
Create SIUD procedures
Show data... F9
Autogrant privileges Shift+Ctrl+A
Goto Database
Goto Object
Refresh F5
Connect to Database Shift+Ctrl+C
Reconnect
Disconnect from Database
Register Database Shift+Alt+R
Unregister Database Shift+Alt+U
Clone Registration Info
Database Registration Info...
Recompute selectivity of all indices
Recompile all stored procedures
Recompile All Triggers
Show SQL Assistant Ctrl+A
Show objects description
Inspector Page Mode
Hide Disconnected Databases

So now you've created your first database with a little help; next we need to fill it with data.

Chapter 4: 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 programming 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. Delphi 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 (Procedural SQL) is initially not so easy to write as IDEs, because 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.

Chapter 5: An Introduction to Stored Procedures

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 it can also handle more complex operations than just simple comparisons. Firebird/InterBase uses stored procedures as the programming environment for integrating active processes in the database.

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. It can be invoked directly from applications, or can be substituted for a table or view in a `SELECT` statement; it can receive input parameters and return values to applications. It is similar to a trigger, but is not automatically executed or bound to a specific table.

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. Firebird and InterBase stored procedure and trigger language is known as PSQL.

Stored procedures offer the following advantages when implementing applications:

1. They are fast. As program execution occurs on the server, there is reduction of network traffic by off-loading application processes from the client to the server.
2. Splitting up of complex tasks into smaller and more logical modules. Stored procedures can be invoked by each other.
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. And as stored procedures are part of InterBase or Firebird, it is irrelevant which front end is subsequently used, be it Delphi, PHP or other.
5. 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. PSQL even offers functions that are not available in SQL, e.g. `IF…WHEN…ELSE`, `DECLARE VARIABLE`, `SUSPEND`, etc.
6. 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.

A 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.

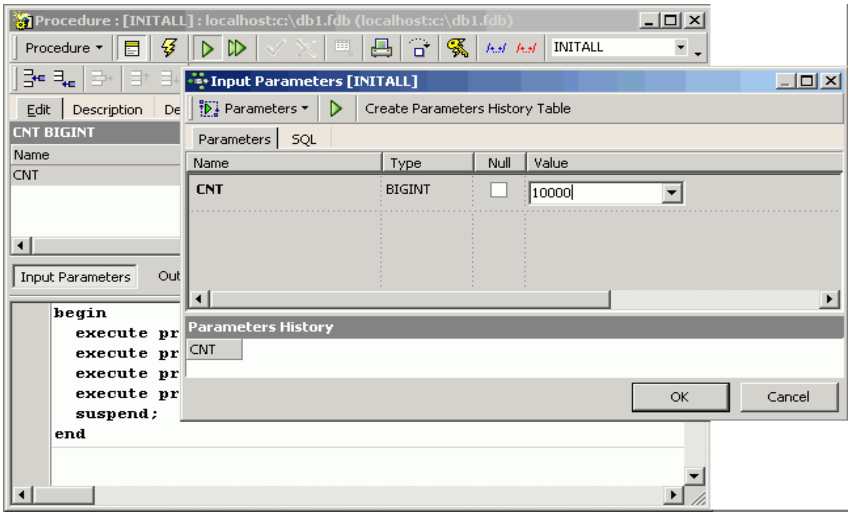
We will illustrate the use of procedures by using the `INITALL` procedure to fill the demo database with data.

1. INITALL

A common problem when writing database systems is to start with a completely empty database, as this gives you an artificial view of how the database is going to work and how the program is going to work in a real environment. When the new system is installed it seems to work fine, but as more data is added certain things start slowing down considerably. It is imperative that database systems be tested and analyzed using realistic test data quantities. The same applies to bench mark testing. It's worthless testing and comparing systems with 100 data sets, when you know the system will be dealing with 100,000,000 data sets within the year.

`INITALL` is a stored procedure which populates the database with data. Open the procedure by double-clicking on the procedure name in the *Database Explorer*. Run the procedure with [F9]. You will be asked to enter a parameter - the number of data sets you wish to be generated.

A reasonable sized database can be created with the parameter 10,000.


Procedure : [INITALL] : localhost:c:\db1.fdb (localhost:c:\db1.fdb)
Procedure
INITALL
Edit
Description
Input Parameters [INITALL]
Parameters
Create Parameters History Table
Parameters
SQL
Parameters History
OK
Cancel
Input Parameters

Name	Type	Null	Value
CNT	BIGINT	☐	10000

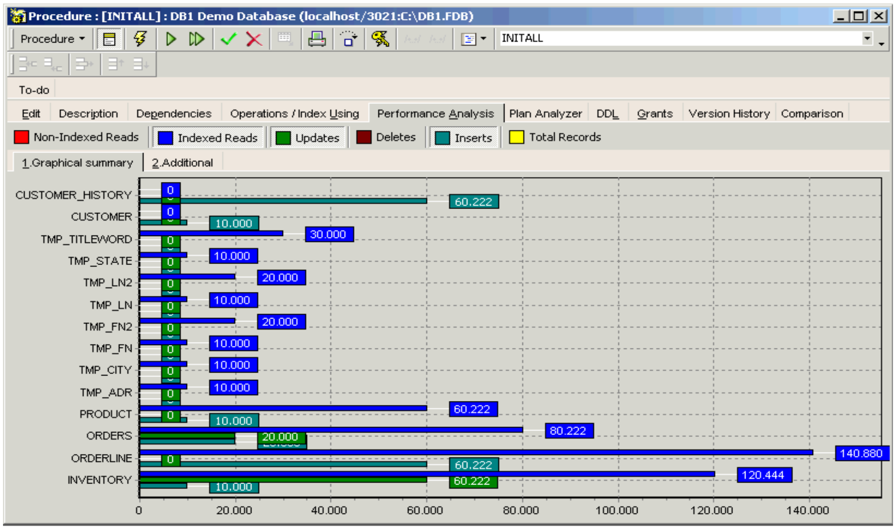
```
begin
  execute pr
  execute pr
  execute pr
  execute pr
  suspend;
end
```

`INITALL` invokes four other procedures:

- `DELETEALL`: deletes any data that might still be stored in the database from previous tests.
- `CREATE_CUSTOMER, CREATE_PRODUCTS` and `CREATE_ORDERS`: then proceeds to generate customers, products and orders.

If the parameter 10,000 has been specified `INITALL` creates 10,000 customers, 10,000 products, 20,000 orders and approximately 60,000 order positions. Altogether around 800,000 operations are performed to create this data, this in itself providing a simple benchmark test to compare different hardware systems and/or database versions. For example, on a standard laptop Firebird 1 took around 60 seconds to generate the data, Firebird 1.5 about 50 seconds and Firebird 2 approximately 30 seconds.

The *Performance Analysis* page offers an insight as to what Firebird has actually had to do, in order to create this data:



For example, on the `INVENTORY` table Firebird has carried out 140,880 reads, 10,000 inserts and 60,222 updates. The IBExpert *Performance Analysis* is explained in more detail in *Chapter 5, 14.3: Performance Analysis*.

Important: Although all this data has been created for the database, until it has been committed, you will notice no increase in the database size. *Commit* is the command which finally writes all alterations to the hard disk. It is at this point that all other users connected to the database can see your alterations. If you're not happy with the data you've generated and want to cancel, you can simply roll back all that the procedure has performed ,to return to the original database status.

If you now disconnect from the database, you will be able to see your actual database size. The database without any data content was about 4 MB; with a 10,000 data set generation it is now 25 MB. Hence using the parameter 100,000, a database size of approximately 250 MB is created, with 1,000,000 a database size of 2,5 GB.

After executing and closing the procedure, you can open the table `customers` to see if the records have been created.

Now you've seen how this works, you can of course adapt the database structure and procedures accordingly to generate a test database commensurate with your enterprise's requirements.

Writing stored procedures is looked at more closely in *Chapter 12: Writing Stored Procedures & Triggers*.

Chapter 6: Basic SQL Commands

You can find a reference of the most important commands in the IBExpert online *SQL Language Reference*, and the full range of Firebird 2.0 commands in the *Firebird 2 Language Reference Update* (both at www.ibexpert.com/doc). 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 group 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).

In order to follow the examples in this section and to offer the chance to play around with Firebird SQLs, we will continue to use the demo database, `db1.fdb`.

1. Simple SELECT commands

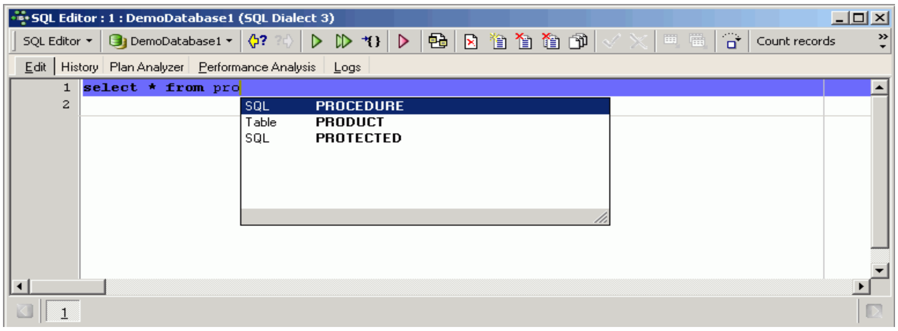
The most basic SELECT command is:

```
    select * from <table_name>
```

where * is a so-called wild card. Let's take an example using our demo database, and enter the query in the IBExpert *SQL Editor* (found in the *Tools* menu) 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.



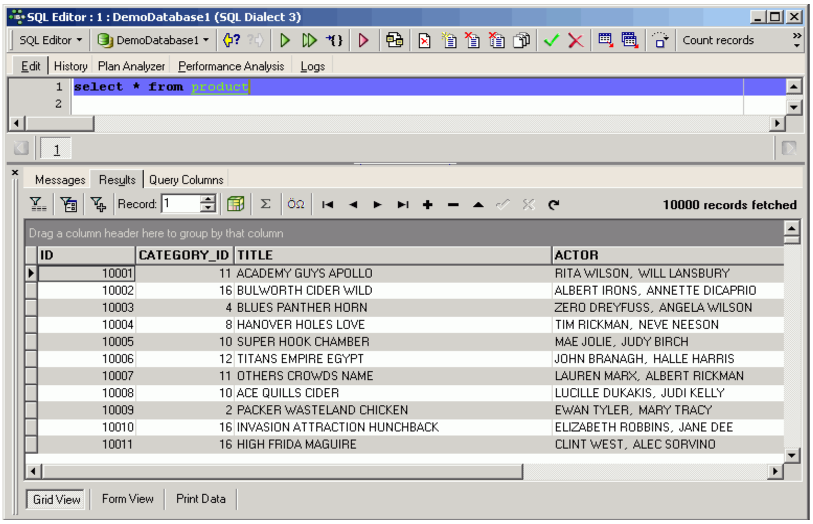
```
select * from pro
```

SQL	PROCEDURE
Table	PRODUCT
SQL	PROTECTED

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, which meet the conditions of the query (in this case all fields of all data sets in the `product` table):

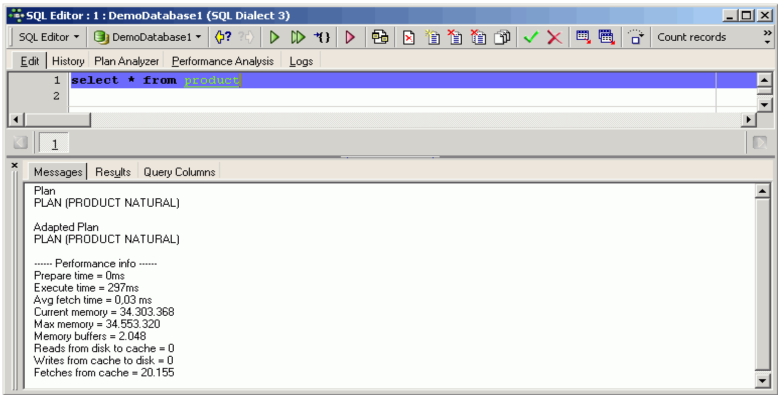


```
select * from product
```

10000 records fetched

ID	CATEGORY_ID	TITLE	ACTOR
10001	11	ACADEMY GUYS APOLLO	RITA WILSON, WILL LANSBURY
10002	16	BULWORTH CIDER WILD	ALBERT IRONS, ANNETTE DICAPRIO
10003	4	BLUES PANTHER HORN	ZERO DREYFUSS, ANGELA WILSON
10004	8	HANOVER HOLES LOVE	TIM RICKMAN, NEVE NEESON
10005	10	SUPER HOOK CHAMBER	MAE JOLIE, JUDY BIRCH
10006	12	TITANS EMPIRE EGYPT	JOHN BRANAGH, HALLE HARRIS
10007	11	OTHERS CROWDS NAME	LAUREN MARX, ALBERT RICKMAN
10008	10	ACE QUILLS CIDER	LUCILLE DUKAKIS, JUDI KELLY
10009	2	PACKER WASTELAND CHICKEN	EWAN TYLER, MARY TRACY
10010	16	INVASION ATTRACTION HUNCHBACK	ELIZABETH ROBBINS, JANE DEE
10011	16	HIGH FRIDA MAGUIRE	CLINT WEST, ALEC SORVINO

Please note that in IBExpert you can specify 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. These specifications can be made in the *Options* menu item, *Environment Options / Tools / SQL Editor*. On the *Messages* page (to the left of the *Results* page) you can see a summary of how Firebird attained the information.



```
select * from product
```

```
Plan
PLAN (PRODUCT NATURAL)

Adapted Plan
PLAN (PRODUCT NATURAL)

------ Performance info ------
Prepare time = 0ms
Execute time = 297ms
Avg fetch time = 0,03 ms
Current memory = 34.303.368
Max memory = 34.553.320
Memory buffers = 2.048
Reads from disk to cache = 0
Writes from cache to disk = 0
Fetches from cache = 20.155
```

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 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.

2. 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, you can't necessarily assess the values in the resulting statistical data, so 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, 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
```

3. 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.

4. 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
```

5. 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.

Now what we want to see is the category that these films are associated with:

```
select p.title, c.txt
from product p
join category c on c.id=p.category_im
```

`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 customers 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* (found in the *Tools* menu). 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.

6. 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!

7. UNION SELECT

`SELECT`s enable you to 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 data types, i.e. you cannot mix `INTEGER`s 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 (which will be a maximum of 30) for the `UNION SELECT`.

8. 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)
```

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 considerably slow performance.

9. 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 and 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.

10. 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, e.g. 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 SELECTed 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.

11. 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!

12. CREATE, ALTER and DROP

If you're just starting off, we would not recommend creating all database objects by writing SQL. Use IBExpert's *Database Explorer* to create and manipulate all your databases and database objects. This is fully documented in the IBExpert online documentation at www.ibexpert.com/doc.

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.

13. Defining code templates in IBExpert

By now you should have had some practice at writing DDL and DML code. You will probably have already noticed that certain commands or series of commands occur repeatedly. To save time and the frustration of repeated typing errors, IBExpert offers two aids to speed up your day-to-day work.

In the IBExpert SQL Editor you can quickly find your most commonly used queries by clicking on the number buttons at the bottom of the *Edit* page. The *History* page offers you a summary of all saved SQLs for the current connected database.

Other pieces of code can be stored as templates. There is even the option to automatically insert the current date, time and author (*Options* menu item, *Keyboard templates*).

14. Analyzing SQL Performance

To support the analysis and optimization of all SQLs, IBExpert offers a *Performance Analysis* in the *SQL Editor*, *Visual Query Builder*, *Procedure Editor* and *Trigger Editor*, a *Plan Analyzer* in the *SQL Editor*, *Procedure Editor* and *Trigger Editor*, and *Logging* functions in all object editors.

14.1 Plan Analyzer

The *Plan Analyzer* 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, with statistics and a summary of the plan and performance listed in the lower panel.

The plan is an Firebird/InterBase 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 *Recompute selectivity* button can be used to recompute the selectivity of all indices.

14.2 Recompute selectivity of all indices

Index statistics are used by the Firebird/InterBase 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 *Database Explorer*, as well as in the SQL Editor's *Plan Analyzer*.

An example illustrating the relevance of index statistics can be found in *Chapter 28: 11. Automating the recalculation of index statistics*.

14.3 Performance Analysis

The *Performance Analysis* displays information showing how much effort was required by Firebird/InterBase to carry out an executed query or procedure. The analysis is performed after a `SELECT` statement is opened or a stored procedure started. The main advantage here of course, is the possibility to compare the performance of different queries and procedures.

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.

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 (*Non-Indexed* or *Indexed Reads*, *Upates*, *Deletes*, *Inserts*, *Total Records*) 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*.

In the *SQL Editor* the lower panel displays the query plan, along with a summary of the performance information included on the *Additional* page.

Additional

The *Additional* page displays a statistical report of the information included in the *Graphical Summary*,along with certain additional information, such as query time, memory and operation.

The analysis displayed on the *Additional* page can also be documented using the *Copy Analysis to Clipboard* button.

Query time shows the time needed to prepare for the execution of the query/procedure, along with the execution time and average fetch time.

The Memory statistics display the current memory used by the server,the maximum memory used by the server during execution of the query/procedure, and buffers, which is 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.

The *Operations* statistics display the number of data pages that were read from the database file to the memory, written and fetched, whilst executing the query/procedure, including reads, writes and fetches.

Pay attention to the *Writes* statistic: 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.

And regarding *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 Firebird/InterBase 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 will be 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.

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. `EXECUTE BLOCK` statements are also logged here.

15. 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.

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 wild card (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 Firebird server. If this process appears to be somewhat slow, check the parameters in `firebird.conf`.

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 (highlighted in red) 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 issuing 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 Firebird/InterBase can operate in the cache. Fetches are the internal operations in Firebird/InterBase, so when one query is slower than the other, it may not be visible directly in the graphical view, for example when Firebird/InterBase 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 the online documentation, *Firebird administration using IBExpert,* at www.ibexpert.com/doc.

Chapter 7: Creating your First Database

1. 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 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 referred to below (*3. Relationships*). Don't get bogged down by the details at this initial stage; attributes can be added later. 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.

2. 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.

Name things simply and logically: call a spade a spade, not a "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.

3. 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. A comprehensive guide to Firebird/InterBase keys and constraints can be found in the IBExpert online documentation at www.ibexpert.com/doc.

There are various kinds of relationships between data, which need to be taken into consideration when defining the constraints:

4. 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 always 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 is 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 is used to store all other information, the third table the RTF articles themselves, and the fourth table the full-text search contents.

5. 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.

6. n:m

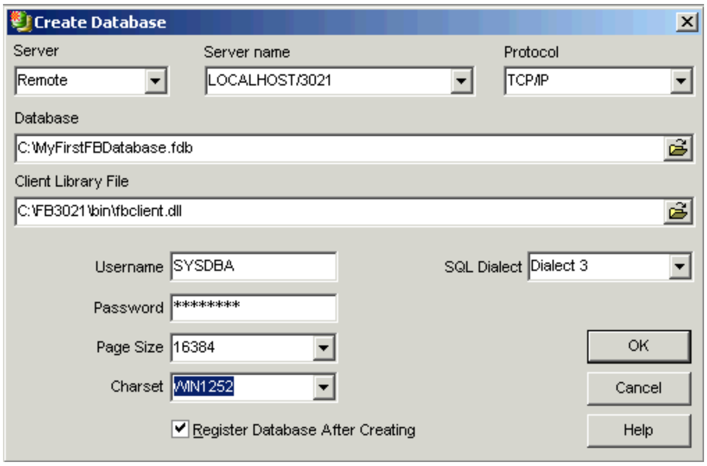
A classic example can be seen in `db1`: one customer can purchase several products and a single product can be purchased by many customers. To make this happen you need to have some linking table in the middle. The `db1` example shows the link from customer to orders; orders is linked to `orderline` 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. Further information regarding the IBExpert *Table Editor* can be found in C*hapter 9: Create a Database Object.*

7. 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. They can be saved to file, exported and printed. Please refer to *Chapter 8: Database Designer* for details.

8. 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* or the respective icon in the *Database* toolbar. The *Create Database* dialog appears:



Server: Remote
Server name: LOCALHOST/3021
Protocol: TCP/IP
Database: C:\MyFirstFBDatabase.fdb
Client Library File: C:\FB3021\bin\fbclient.dll
Username: SYSDBA
SQL Dialect: Dialect 3
Password: ********
Page Size: 16384
Charset: WIN1252
[✓] Register Database After Creating

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 *Server name* and *Protocol*. The drop-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, 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, the *Server name* and *Protocol* fields are automatically blended out, as they are in this case irrelevant.

The *Server name* must be known when accessing remotely. The following syntax should be used:

- Windows `SERVER_NAME:C:\path\database.fdb`
- Linux `SERVER_NAME:/path/database.fdb`

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 Firebird/InterBase 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.conf`), the database connection path would be:

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

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

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. Although the introduction of the new local Firebird protocol, XNET, in Firebird 2.0 has solved many of the former problems of the previous local transport protocol (often referred to as IPC or IPServer).

Select your database location by clicking on the folder icon to the right of this field, specifying the full path, decide on your database name, 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 Firebird/InterBase does not support databases on mapped drive letters. The database suffixes do not have to adhere to the forms offered in the list.

The *Client Library File* field displays the path and client library file name, as specified in the *Default Client Library* option, found in the IBExpert *Options* menu item, *Environment Options / Preferences*. This can, of course, be overwritten if wished.

Only those user 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, `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.

The passwords are encrypted in the `ISC4.GDB`, `security.fdb` or `security2.fdb`. 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.

Firebird/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.

Under *SQL Dialect* either *Dialect 1* (up to and including InterBase 5) or *Dialect 3* (InterBase 6/Firebird) needs to be specified. For new projects it is recommended that dialect 3 be specified.

Page Size is for 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 administrative 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.

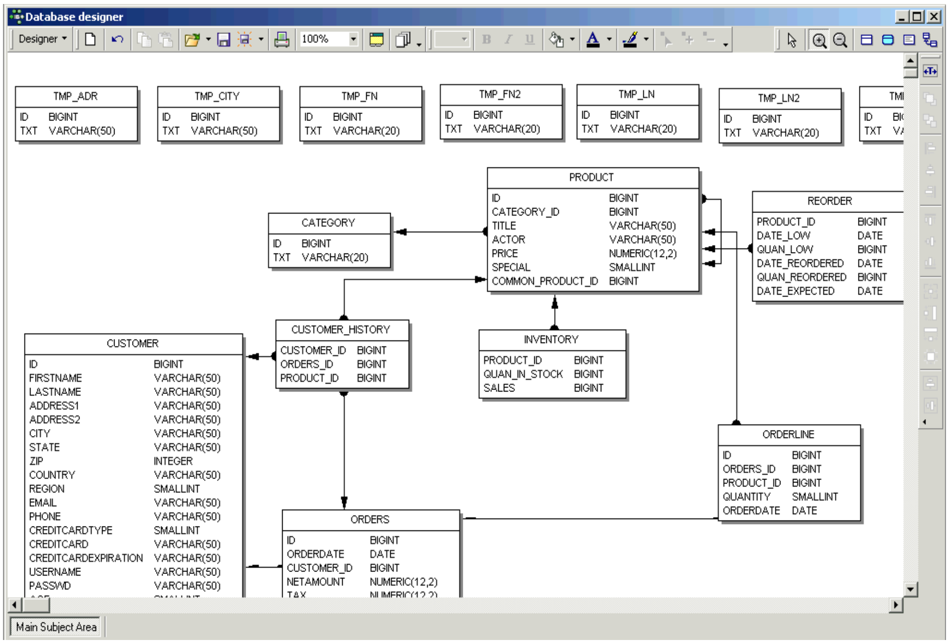
Under *Charset* the default character set can be defined for the database. (A default character set can be specified as default for all new databases in the IBExpert *Options* menu item, *Environment Options*, under *Default character set*.) This character set is useful, when the database created is to be used for foreign languages as it is applicable for all areas of the database unless overridden by the domain or field definition. It controls not only the available characters that can be stored and displayed, but also the collation order. Given a specific character set, a specific collation order can be specified when data is selected, inserted, or updated in a column. If not specified, the parameter defaults to `NONE`, i.e. values are stored exactly as typed. 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.

The *Register Database After Creating* check box automatically generates the *Database Registration* dialog so that the database can be registered for use in IBExpert. The *Register Database* dialog however offers many further options. We recommend clicking this check box (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 *Database Explorer* (the main navigational area at 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.

Chapter 8: 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 optically, 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.

Started from the IBExpert *Tools* menu, the *Designer* can be used to open an existing diagram from file, or a create a new diagram. To illustrate the many features of the *Database Designer* we will reverse engineer the demo database, db1. Select the *Reverse Engineer ...* menu item, select your demo database from the list of registered databases and start.



TMP_ADR	
ID	BIGINT
TXT	VARCHAR(50)

TMP_CITY	
ID	BIGINT
TXT	VARCHAR(50)

TMP_FN	
ID	BIGINT
TXT	VARCHAR(20)

TMP_FN2	
ID	BIGINT
TXT	VARCHAR(20)

TMP_LN	
ID	BIGINT
TXT	VARCHAR(20)

TMP_LN2	
ID	BIGINT
TXT	VARCHAR(20)

PRODUCT	
ID	BIGINT
CATEGORY_ID	BIGINT
TITLE	VARCHAR(50)
ACTOR	VARCHAR(50)
PRICE	NUMERIC(12,2)
SPECIAL	SMALLINT
COMMON_PRODUCT_ID	BIGINT

CATEGORY	
ID	BIGINT
TXT	VARCHAR(20)

REORDER	
PRODUCT_ID	BIGINT
DATE_LOW	DATE
QUAN_LOW	BIGINT
DATE_REORDERED	DATE
QUAN_REORDERED	BIGINT
DATE_EXPECTED	DATE

CUSTOMER_HISTORY	
CUSTOMER_ID	BIGINT
ORDERS_ID	BIGINT
PRODUCT_ID	BIGINT

INVENTORY	
PRODUCT_ID	BIGINT
QUAN_IN_STOCK	BIGINT
SALES	BIGINT

CUSTOMER	
ID	BIGINT
FIRSTNAME	VARCHAR(50)
LASTNAME	VARCHAR(50)
ADDRESS1	VARCHAR(50)
ADDRESS2	VARCHAR(50)
CITY	VARCHAR(50)
STATE	VARCHAR(50)
ZIP	INTEGER
COUNTRY	VARCHAR(50)
REGION	SMALLINT
EMAIL	VARCHAR(50)
PHONE	VARCHAR(50)
CREDITCARDTYPE	SMALLINT
CREDITCARD	VARCHAR(50)
CREDITCARDEXPIRATION	VARCHAR(50)
USERNAME	VARCHAR(50)
PASSWD	VARCHAR(50)

ORDERLINE	
ID	BIGINT
ORDERS_ID	BIGINT
PRODUCT_ID	BIGINT
QUANTITY	SMALLINT
ORDERDATE	DATE

ORDERS	
ID	BIGINT
ORDERDATE	DATE
CUSTOMER_ID	BIGINT
NETAMOUNT	NUMERIC(12,2)

The above illustration displays the sample db1.fdb database, with the *Model Options - Links* option, *Automatically trace links* switched on, and a number of display options found under *Model Options Table* activated. 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.

Templates can be used (IBExpert menu item *Environment Options / Templates*) to create foreign and constraint names automatically. It is also possible to customize the highlighting of variables. Use *Options / Editor Options / Color* to choose color and font style for variables. (Custom colors are saved in and restored from a `grc` file.) Alternatively, existing objects may be dragged and dropped from the *Database Explorer* (also from the *Project View* tree) and *SQL Assistant* into the main editing area.

The *Model Navigator* in the *SQL Assistant* enables you to navigate models quickly. The *Database Explorer* 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.

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

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. You can use the right-click context-sensitive menus to lock visual objects to protect them against casual modification of size and position.

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 and executed, to apply these alterations to the database itself.

Generation of the update database script includes the processing of generators, triggers, exceptions and procedures. View dependencies are also taken into account when the script is generated.

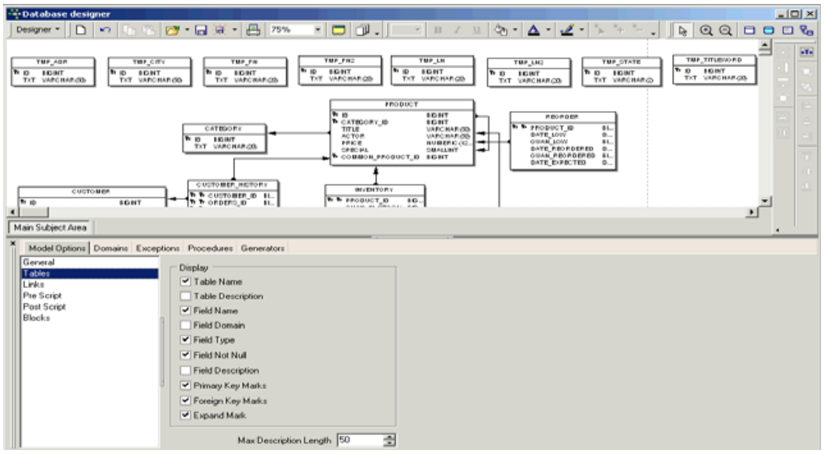
If `INIT` statements need to be specified you will need to use the model prescript (*Model Options*). Otherwise statements such as `SET NAMES`, `SET SQL DIALECT`, `CREATE DATABASE` will be removed from the resulting `CREATE DATABASE` script.

1. 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* window. By clicking on a subject on the left, further options are offered.

The pre- and postscript options provide the option 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.



- [✓] Table Name
- [ ] Table Description
- [✓] Field Name
- [ ] Field Domain
- [✓] Field Type
- [✓] Field Not Null
- [ ] Field Description
- [✓] Primary Key Marks
- [✓] Foreign Key Marks
- [✓] Expand Mark

Max Description Length 50

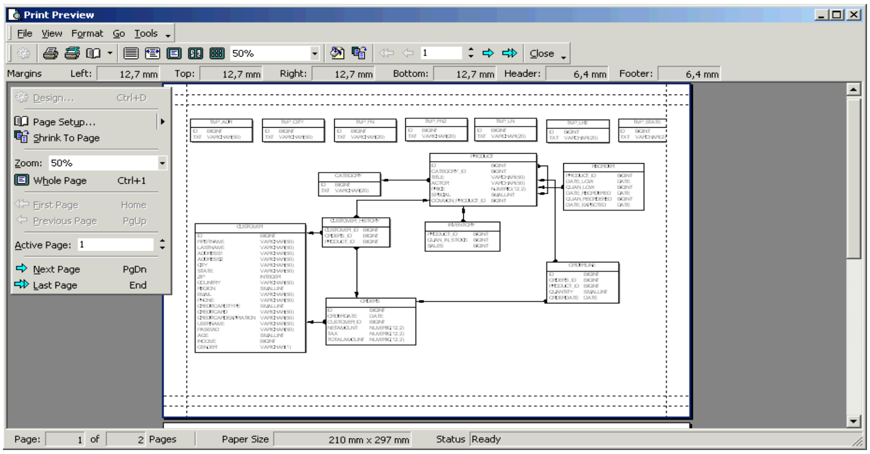
2. Export

The database model can be exported, either as a bitmap (`.bmp`) or an enhanced metafile (`.emf`). Simply load the model to be exported, click the *Export* menu item, and specify the name and format.

3. 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.

It is possible to store printing options between sessions. You can 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:



Zoom: 50%

Active Page: 1

Margins Left: 12,7 mm Top: 12,7 mm Right: 12,7 mm Bottom: 12,7 mm Header: 6,4 mm Footer: 6,4 mm

Page: 1 of 2 Pages Paper Size 210 mm x 297 mm Status Ready

Chapter 9: Create a Database Object

1. Database objects

Firebird/InterBase offer the following database objects:

- Domain
- Table
- View
- Stored Procedure
- Trigger
- Generator/Sequence
- Exception
- UDF (User-Defined Function)
- Role
- Index

The number of objects in a database is unlimited. The individual database object are described in more detail in *Chapter 10: More about Database Objects .*

2. Adding a new table to the db1 demo database

To illustrate the creation of a database object using IBExpert we will create a new table in our `db1` database and call it `CUSTOMERGROUP`.

Ensure you are connected to your `db1` database. Click on the *Tables* node and use either the *Database* menu item, *Create Table*, the right-click context-sensitive menu item, *Create Table*, or the key combination, [Ctrl + N] to open the IBExpert *Table Editor*.

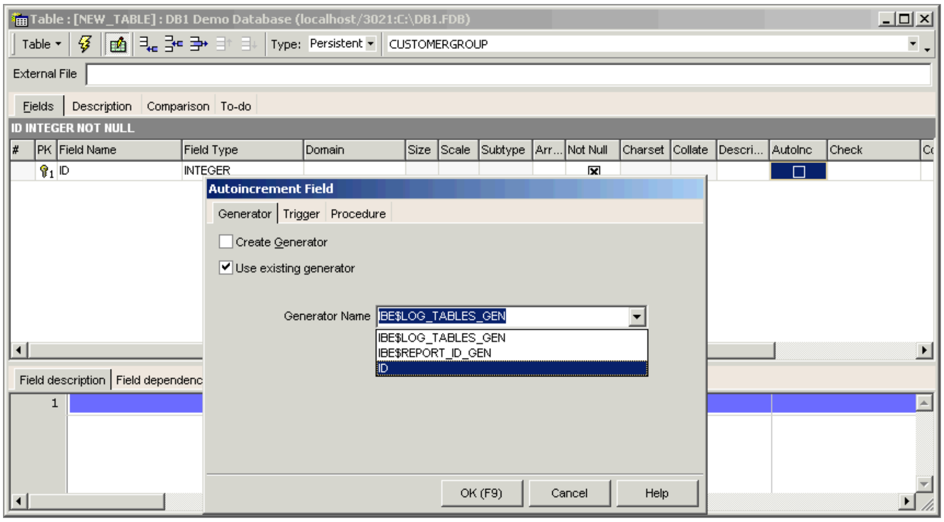
Firstly name the table `CUSTOMERGROUP` (top right). The default value *Persistent* can be left as it is (this feature allows options for the creation of temporary tables).

3. Primary key

Then create a few fields beginning with the 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. 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`.

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` data type 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.

So, to create the first field for this table, simply click on the *Field Name* `NEW_FIELD` and overwrite it with `ID`. By double-clicking (or using the space bar) on the empty *PK* field to the left of the *Field Name*, the key symbol appears, an the `NOT NULL` check box is marked. By double-clicking (or again, using the space bar) on the *AutoInc* field, check *Use existing Generator* and select `ID` from the drop-down list:



Further fields can then be added using the *Add Field* icon.

4. 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* on the *Constraints* page.

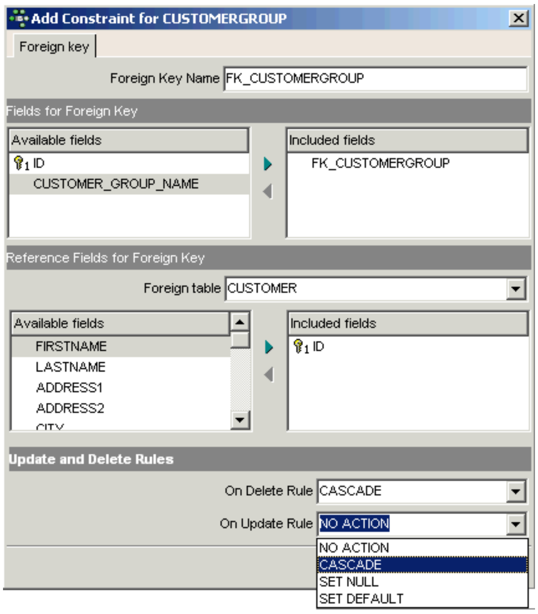
When a primary key:foreign key relationship links to a single row in another table, what is known as a virtual row is created. The columns in that second table provide additional description about the primary key of the first table. This is also known as a 1:1 relationship. A foreign key can also point to itself. Firebird enables you to reference recursive data and even represent tree structures in this way.

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).

In our table storing customer groups, it needs to be determined which customers belong to which group.

When defining a foreign key, it is necessary to specify update and delete rules to ensure referential integrity. 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:

- **NO ACTION**: throws an exception if there is a existing relationship somewhere in another table.
- **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.
- **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`.
- **SET DEFAULT**: the foreign key column is set to its default value when a primary key field is deleted.



Foreign Key Name: FK_CUSTOMERGROUP

Fields for Foreign Key – Available fields: ID, CUSTOMER_GROUP_NAME; Included fields: FK_CUSTOMERGROUP

Reference Fields for Foreign Key – Foreign table: CUSTOMER; Available fields: FIRSTNAME, LASTNAME, ADDRESS1, ADDRESS2, CITY; Included fields: ID

On Delete Rule: CASCADE

On Update Rule: NO ACTION

To define a foreign key for our new table, create a new field, `FK_CUSTOMER`, defined as `NOT NULL`, and use the context-sensitive right-click menu item, *Create Foreign Key …* Simply link to the ID field in the `CUSTOMER` table, define the actions on update and delete and, job done!

We have then added just one further field, `CUSTOMER_GROUP_NAME (varchar(30))`, and after committing our work to write it to the database, our table is ready.

5. Create test data

As the rest of the `db1` database is already filled with data, this table should also be filled with test data. IBExpert provides a solution: the *Test Data Generator*, found in the IBExpert *Tools* menu. This feature is almost self-explanatory: select the table and number of data sets. Check those fields to the left of the field *Name*, and select the options for each field individually on the right-hand side:

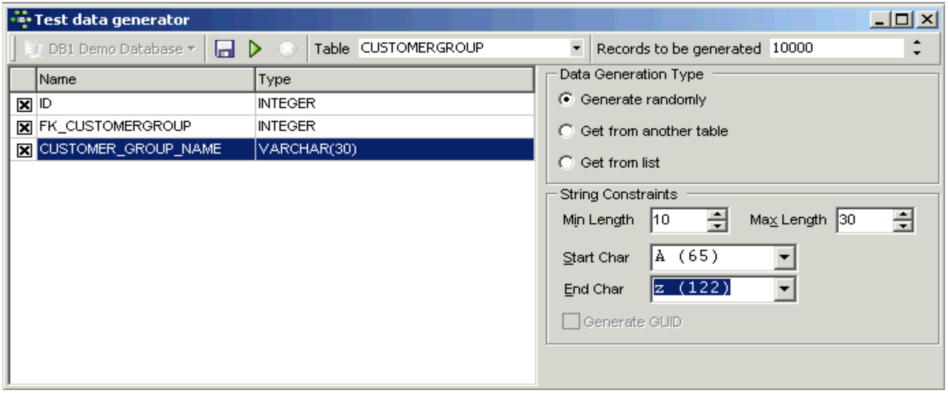


Table: CUSTOMERGROUP
Records to be generated: 10000

	Name	Type
☒	ID	INTEGER
☒	FK_CUSTOMERGROUP	INTEGER
☒	CUSTOMER_GROUP_NAME	VARCHAR(30)

Data Generation Type
- ● Generate randomly
- ○ Get from another table
- ○ Get from list

String Constraints
- Min Length: 10
- Max Length: 30
- Start Char: A (65)
- End Char: z (122)
- ☐ Generate GUID

Then start the automatic data generation using the *Generate* icon or [F9].

6. 253 changes of table left

Each table in an Firebird/InterBase 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.

This obligatory cleanup after many metadata changes is in itself a useful feature, however it also means that users who regularly use `ALTER TRIGGER` to deactivate triggers during e.g. bulk import operations are forced to backup and restore much more often then needed. Since changes to triggers don't imply structural changes to the table itself, Firebird (since version 1.0) does not increment the table change counter when `CREATE`, `ALTER` or `DROP TRIGGER` is used. One thing has remained though: once the counter is at 255, you can no longer create, alter or drop triggers for that table.

Chapter 10: More about Database Objects

Firebird/InterBase administrates the database data in database objects. These are the fundamental building blocks of the database and are part of the database's metadata. 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.

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

1. Domain

A domain is a user-defined custom data type 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 fields/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. You can define a number of characteristics including: data type, an optional default value, optional disallowing of `NULL` values, an optional `CHECK` constraint and an an optional collation clause.

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. The domain default, collation clause and `NOT NULL` settings ca be overridden by the field definition, and a field based on a domain can add additional `CHECK` constraints to the domain's `CHECK` constraint.

A domain can be created, modified and dropped as all other Firebird/InterBase objects in the IBExpert Database 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, Firebird/InterBase 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.

2. 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 data type.

3. View

3.1 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 updatable 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 implemented 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, but 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 `SELECT`s, 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 `SELECT`s exactly as if they were tables:

```
select * from Vw_Product_Short
```

Further information can be found in the IBExpert online documentation at www.ibexpert.com/doc.

3.2 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 (check box 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 updatable 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:

- 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.)
- 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.

- The SELECT statement does not include a DISTINCT operator. This restriction might have the effect of removing duplicate rows, making it impossible for Firebird/InterBase to determine which row to update.
- The SELECT statement does not include aggregate functions or the GROUP BY or HAVING operators.
- The SELECT statement does not include stored procedures or user-defined functions.
- The SELECT statement does not contain joined tables.
- 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 a combination of user-defined referential constraints, triggers, and unique indexes.

3.3 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 Firebird/InterBase 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 Firebird/InterBase returns an error.

4. 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, and can be called by client applications. They are pre-compiled, so they don't need to be sent over the network and parsed every time, they are just executed. Procedures can take parameters and - like SELECTs - give back their data in the form of a table. They are similar to triggers, but is not automatically executed or bound to a specific table.

They are written in Firebird/InterBase procedure and trigger language, also known as PSQL.

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. Firebird/InterBase uses stored procedures as the programming environment for integrating active processes in the database.

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:

- 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.
- 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.
- 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.
- 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.
- 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.
- 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:

- 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.
- 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.
- If the operation must be performed periodically or frequently.
- If the operation is performed in the same manner by a number of different processes, or processes within the application, or by different applications.

All SQL scripts can be incorporated into a stored procedure and up to ten SQLs incorporated in one single 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.

4.1 Executing stored procedures

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:

- from within another stored procedure.
- from within a trigger.
- from an application.

When a procedure is executed from within an Firebird/InterBase 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

4.2 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.

4.3 Non-select procedures

Execute or non-select procedures perform an action and do not return any results.

5. 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 Firebird/InterBase Event Manager.

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 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.

Triggers take no input parameters and do not return values.

A trigger is never called directly. Instead, when an application or user attempts to `INSERT`, `UPDATE` or `DELETE` a row in a table, any triggers associated with that table and operation automatically execute, or fire. Triggers defined for `UPDATE` on non-updatable views fire even if no update occurs.

Several triggers can be created for one event. The `POSITION` parameter determines the sequence in which the triggers are executed.

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).

5.1 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.

Specify who is allowed to access your application, or raise an exception when certain unwanted applications attempt to access your database. Database triggers are also a really nice feature for protocols, enabling you for example to create your own login mapping with IP addresses an so on.

5.2 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.

In IBExpert database triggers can be created, edited and deleted in the same way as table-bound triggers.

5.3 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.

NEW and OLD context variables

In triggers (but not in stored procedures), Firebird/InterBase provides two context variables that maintain information about the row being inserted, updated or deleted:

- OLD.columnName refers to the current or previous values in a row being updated or deleted. It is not relevant for INSERT triggers.
- 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.

6. Generator/Sequence

Generators are automatic sequential counters, spanning the whole database. They are necessary because all operations in Firebird/InterBase 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 BIGINT (SQL dialect 3) since InterBase 6/Firebird (in

SQL dialect 1 it is 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.

A database can contain any number of generators and they can be used and updated in any transaction. They are the only transaction-independent part of Firebird/InterBase. 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.

7. 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.

The maximum size of an exception message was raised in Firebird 2.0 from 78 to 1021 bytes.

8. UDF (User-Defined Function)

A user-defined function (UDF) is used to perform tasks that Firebird/InterBase can't. It is an external database function written entirely in another language, such as C++ or Pascal, to perform data manipulation tasks not directly supported by Firebird/InterBase.

UDFs can be called from Firebird/InterBase 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.

9. Blob filter

Blob filters are routines for blobs. They are user-written programs that convert data stored in Blob columns from one subtype 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.

10. 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.

Users must specify the role at connect time.

11. Index

Indices are a sorted list of pointers into tables, to speed data access. An index can be ascending or descending, and can also be defined as unique if wished. If the indexed field is unique there is only one pointer.

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, Firebird/InterBase automatically generates a corresponding ascending index and enforces the uniqueness constraint demanded by the relational model.

If you wish to ascertain just how many indices already exist for individual tables in a database, query the following from the system table, `RDB$INDICES`:

```
select * from RDB$INDICES where
RDB$INDICES.RDB$RELATION_NAME='MYTABLE'
```

or view the indices list under the *Indices* node in the Database Explorer.

System tables and indices can be viewed in the IBExpert Database Explorer by activating the *Show System Tables* and *Show System Indices* check options, found in the *Database registration Info* on the *Additional* page.

Indices are updated every time a new data set is inserted, or rather, the index-referenced field is updated. Firebird/InterBase 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, Firebird/InterBase 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.

Firebird/InterBase 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 Firebird/InterBase 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).

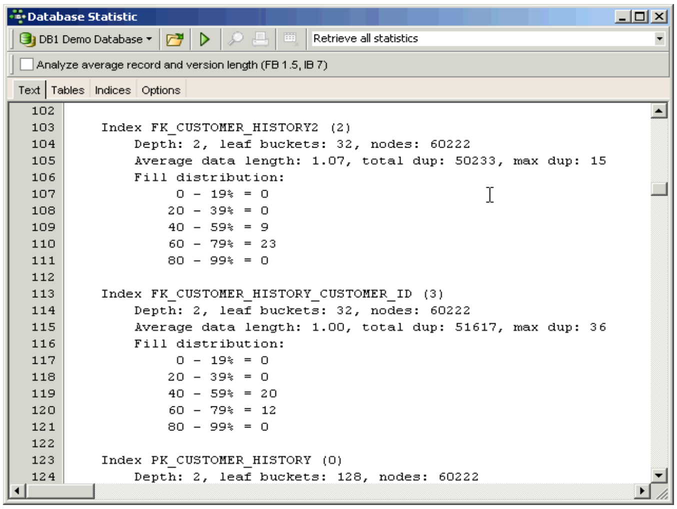
Indices can always be set after the database is actually in use, based on the performance requirements. For further details and examples please refer to *Chapter 14:14.3 Performance Analysis*.

11.1 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. A good selectivity is close to 0 - it's the result of: 1/distinct values.

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. Although the Optimizer only uses indices with a selectivity < 0.01 when there are no other appropriate indices available.

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



Retrieve all statistics

☐ Analyze average record and version length (FB 1.5, IB 7)

```

    Index FK_CUSTOMER_HISTORY2 (2)
        Depth: 2, leaf buckets: 32, nodes: 60222
        Average data length: 1.07, total dup: 50233, max dup: 15
        Fill distribution:
             0 - 19% = 0
            20 - 39% = 0
            40 - 59% = 9
            60 - 79% = 23
            80 - 99% = 0

    Index FK_CUSTOMER_HISTORY_CUSTOMER_ID (3)
        Depth: 2, leaf buckets: 32, nodes: 60222
        Average data length: 1.00, total dup: 51617, max dup: 36
        Fill distribution:
             0 - 19% = 0
            20 - 39% = 0
            40 - 59% = 20
            60 - 79% = 12
            80 - 99% = 0

    Index PK_CUSTOMER_HISTORY (0)
        Depth: 2, leaf buckets: 128, nodes: 60222
```

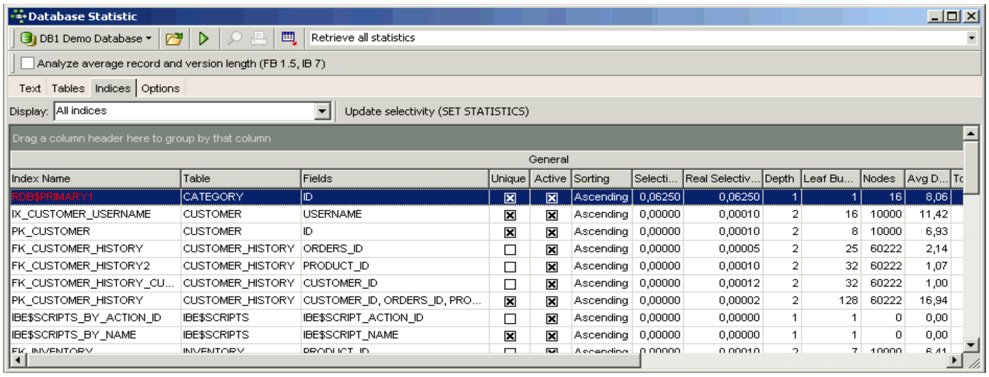
an index. Your benchmark is always the ID - the primary key, because that will always have complete unique values in it, and therefore the lowest selectivity.

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 *Index Editor*, IBExpert *Services* menu item, *Database Statistics*, in the *Database* menu, or in the right-click *Database Explorer* menu).

Only the creator of an index can use `SET STATISTICS`. Please note that `SET STATISTICS` does not rebuild an index; to rebuild an index, use `ALTER INDEX`.

The recalculation of selectivity can be automated to ensure the most efficient use of 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.


Database Statistic
DB1 Demo Database
Retrieve all statistics
Analyze average record and version length (FB 1.5, IB 7)
Text
Tables
Indices
Options
Display: All indices
Update selectivity (SET STATISTICS)
Drag a column header here to group by that column

General												
Index Name	Table	Fields	Unique	Active	Sorting	Selecti...	Real Selectiv...	Depth	Leaf Bu...	Nodes	Avg D...	Tc
RDB$PRIMARY1	CATEGORY	ID	☒	☒	Ascending	0,06250	0,06250	1	1	16	8,06	
IX_CUSTOMER_USERNAME	CUSTOMER	USERNAME	☒	☒	Ascending	0,00000	0,00010	2	16	10000	11,42	
PK_CUSTOMER	CUSTOMER	ID	☒	☒	Ascending	0,00000	0,00010	2	8	10000	6,93	
FK_CUSTOMER_HISTORY	CUSTOMER_HISTORY	ORDERS_ID	☐	☒	Ascending	0,00000	0,00005	2	25	60222	2,14	
FK_CUSTOMER_HISTORY2	CUSTOMER_HISTORY	PRODUCT_ID	☐	☒	Ascending	0,00000	0,00010	2	32	60222	1,07	
FK_CUSTOMER_HISTORY_CU...	CUSTOMER_HISTORY	CUSTOMER_ID	☐	☒	Ascending	0,00000	0,00012	2	32	60222	1,00	
PK_CUSTOMER_HISTORY	CUSTOMER_HISTORY	CUSTOMER_ID, ORDERS_ID, PRO...	☒	☒	Ascending	0,00000	0,00002	2	128	60222	16,94	
IBE$SCRIPTS_BY_ACTION_ID	IBE$SCRIPTS	IBE$SCRIPT_ACTION_ID	☐	☒	Ascending	0,00000	0,00000	1	1	0	0,00	
IBE$SCRIPTS_BY_NAME	IBE$SCRIPTS	IBE$SCRIPT_NAME	☒	☒	Ascending	0,00000	0,00000	1	1	0	0,00	
FK_INVENTORY	INVENTORY	PRODUCT_ID	☐	☒	Ascending	0,00000	0,00010	2	7	10000	6,41	

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

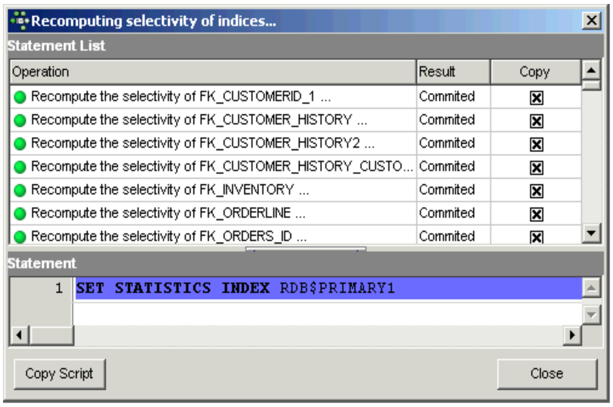
If the developer wishes to override Firebird/InterBase's automatic index selection, and determine the index search sequence himself, this must be specified in SQL. Each index needs to be named and entered individually.

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.

11.2 Recompute selectivity of all indices

Indices statistics are used by the Firebird/InterBase 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 Database Explorer.



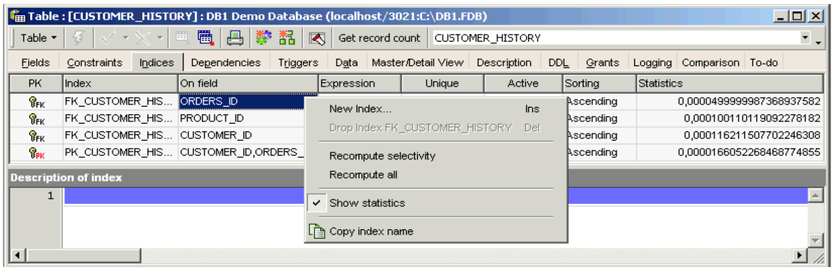
Operation	Result	Copy
● Recompute the selectivity of FK_CUSTOMERID_1 ...	Commited	☒
● Recompute the selectivity of FK_CUSTOMER_HISTORY ...	Commited	☒
● Recompute the selectivity of FK_CUSTOMER_HISTORY2 ...	Commited	☒
● Recompute the selectivity of FK_CUSTOMER_HISTORY_CUSTO...	Commited	☒
● Recompute the selectivity of FK_INVENTORY ...	Commited	☒
● Recompute the selectivity of FK_ORDERLINE ...	Commited	☒
● Recompute the selectivity of FK_ORDERS_ID ...	Commited	☒

```
SET STATISTICS INDEX RDB$PRIMARY1
```

You do not need to shut down the database to recompute the selectivity of indices. Individual indices can be recomputed directly in the *Index Editor*, in the SQL Editor on the *Plan Analyzer* page (simply click the *Recompute selectivity* button), or manually in the SQL Editor using the command:

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

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



PK	Index	On field	Expression	Unique	Active	Sorting	Statistics
FK	FK_CUSTOMER_HIS...	ORDERS_ID				Ascending	0,000049999987368937582
FK	FK_CUSTOMER_HIS...	PRODUCT_ID				Ascending	0,00010011019092278182
FK	FK_CUSTOMER_HIS...	CUSTOMER_ID				Ascending	0,000116211507702246308
PK	PK_CUSTOMER_HIS...	CUSTOMER_ID,ORDERS_				Ascending	0,0000166052268468774855

The same *Recomputing Selectivity* dialog as above is then displayed. The new statistical values can be viewed for individual tables in the *Index Editor* and the *Table Editor* on the *Indices* page (providing the statistics are blended in using the right-click menu item *Show Statistics*).

11.3 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 function also useful, for example, when backing up an older InterBase version (e.g. v6) and restoring in a newer version, such as InterBase 2007 or Firebird 2.1, as Firebird/InterBase simply copies the data and metadata into the new version when restoring. If a variable name, that is a keyword in the stored procedure, is wrong, it is unfortunately not recognized during the backup and restore procedure 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 Database Explorer.

Chapter 11: More about Data Types

Firebird/InterBase tables are defined by the specification of columns, which accommodate appropriate information in each column using data types, for example, numerical (`NUMERIC`, `DECIMAL`, `INTEGER`), textual (`CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`), date (`DATE`, `TIME`, `TIMESTAMP`) or blobs.

The data type 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 data type can be defined in IBExpert using the *Database Explorer*, by creating a domain or creating a new field in the *Create Table* or *Table Editors*.

It can of course, also be defined using SQL directly in the IBExpert *SQL Editor*.

1. Blob - Binary Large OBject

A blob is a data type 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. 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, 8K page size up to 32 GB and so on.

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.

Since Firebird 2.1 text blobs can masquerade as long `VARCHAR`s. At various levels of evaluation, the Firebird engine now treats text blobs that are within the 32,765-byte size limit as though they were `VARCHAR`. String functions like `CAST`, `LOWER`, `UPPER`, `TRIM` and `SUBSTRING` will work with these blobs, as well as concatenation and assignment to string types. You can even access blob contents using `CONTAINING` and `LIKE`. `ORDER BY` however should not be used on blobs, as it sorts and displays the blob fields in the order that they were created and not according to content.

Firebird/InterBase 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 *Database Explorer* or the IBExpert *SQL Editor*.

Blob specification includes the subtype, segment size and, if wished, the character set.

When the *Grid* 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*.

1.1 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 *Database Explorer*).

A blob segment size can be defined, to increase the performance when inputting and outputting blob data. This is because all blob contents are stored in blocks, and are fetched via these blocks.

When a blob is extracted, the Firebird/InterBase 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.

1.2 Subtype

Subtypes are used to categorize the data type 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 Firebird/InterBase:

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 Firebird/InterBase.
Negative value	User-defined blob subtypes.

Blob fields can be specified using the *Domain Editor* or the *Table Editor* (started from the IBExpert *Database Explorer*).

The specification of a user-defined blob subtype has no effect upon Firebird/InterBase, as the Firebird/InterBase 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, Firebird/InterBase assumes `0` = binary data.

2. CHAR and VARCHAR

Firebird/InterBase provides two basic data types to store text or character information: `CHAR` and `VARCHAR` (blobs also allow character storage using the subtype text).

`CHAR` and `VARCHAR` are data types 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, Firebird/InterBase 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. 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.

2.1 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.

3. INTEGER, SMALL INTEGER and BIG INTEGER

INTEGER (INT, SMALLINT and BIGINT)

INTEGER data types are used to store whole numbers. 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
BigInt	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: 15 bits are for the number and 1 bit for the sign. It is usually preferable to use an INTEGER data type 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 Firebird/InterBase 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*.

`SMALLINT`s can also be used for `BOOLEAN` data types e.g. true/false, male/female.

4. FLOAT and DOUBLE PRECISION

`FLOAT` data types 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 data type `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 data type `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` data type is implemented in the majority of Firebird/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*.

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.

5. NUMERIC and DECIMAL

The `NUMERIC` data type 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 data types 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 EUR 12.40 with NUMERIC(4,2) into EUR 1,240).

Firebird/InterBase supports a number of options for specifying or not specifying PRECISION and SCALE:

- If neither PRECISION nor SCALE are specified, Firebird/InterBase defines the column as INTEGER instead of NUMERIC and stores only the integer portion of the value.
- When using SQL Dialect 1, if just PRECISION is specified, Firebird/InterBase converts the column to a SMALLINT, INTEGER or DOUBLE PRECISION data type, based on the number of significant digits being stored.

In SQL Dialect 3, if just PRECISION is specified, Firebird/InterBase converts the column to a SMALLINT, INTEGER or INT64 data type, based on the number of significant digits being stored.

It is important to distinguish between the two dialects, because since INT64 is an INTEGER data type, 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 data type should only be used for fields that are later to be used as part of a calculation.

Firebird/InterBase converts the columns as follows:

Definition	*Data type created*
Decimal(1) - Decimal(4)	Small Integer
Decimal(5) - Decimal(9)	Integer
Decimal(10) - Decimal(18)	Int (64)

Note that if a DECIMAL(5) data type is specified, it is actually possible to store a value as high as a DECIMAL(9) because Firebird/InterBase uses the smallest available data type to hold the value. For a DECIMAL(5) column, this is an INTEGER, which can hold a value as high as a DECIMAL(9).

6. DATE, TIME, TIMESTAMP

Firebird and InterBase: DATE, TIME, TIMESTAMP, NOW.

Since Firebird 2.0: CURRENT_DATE, CURRENT_TIMESTAMP.

The DATE data type stores values which represent a date. 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 Firebird/InterBase 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 data type `DATE` means current time and date of the server; there is also `TODAY` (only date; the time is always set at midnight, `YESTERDAY`, `TOMORROW`).

The `TIME` data type is an SQL dialect 3 data type. `TIME` is a 32-bit field type of `TIME` values. The range is from 0:00 AM to 23:59:9999 PM.

`TIMESTAMP` is an SQL dialect 3 data type. `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.

Note: `CURRENT_TIMESTAMP` and `'NOW'` are not exactly the same - `CURRENT_TIMESTAMP` represents the statement time and `'NOW'` represents the current timestamp. For example, if you perform a long running update on a lot of records, you will see that using `CURRENT_TIMESTAMP` produces the same value for all records (the timestamp when the update statement was started); using `'NOW'`, you will see different values for each record, since the value is taken on a record level.

It is possible to specify the display format of a date/time field under *Environment Options / Grid / Display Formats*.

7. Array

Firebird/InterBase 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. It 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.

Arrays can be declared as a domain or directly in the table definition following the data type 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 array dimensions are specified 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 2 x 5 = 10 measurements can be logged. If counting is to begin at, for example, 0, the array definition is as follows:

```
    [0:2, 0:5]
```

When using arrays, it is important to be aware of the advantages and limitations.

7.1 Advantages of arrays

- InterBase operations can be performed upon the total data type 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.
- 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.

7.2 Array limitations

- A user-defined function can only access one element in an array.
- 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.
- The data stored in this way cannot be selected per index; each query always accesses the fields unindexed.

8. Boolean

Firebird/InterBase does not offer a native `BOOLEAN` data type. 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:

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` data type without further concern. It is particularly suitable from a Delphi point of view, as Pascal `BOOLEAN`s work in a similar manner.

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.

9. NOT NULL, 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` 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 data types.

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>`), and just because a customer's telephone number is not known (`<null>`), this does not mean that the customer has no telephone, and he most certainly will not have the telephone number "0"!

Chapter 12: Writing Stored Procedures & 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 at www.ibexpert.com/doc in the *Structured Query Language* chapter, *Stored Procedure and Trigger Language*.

Firebird/InterBase 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 ...`

and the 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>`

Both stored procedure and trigger statements includes SQL statements that are conceptually nested inside the main statement. In order for Firebird/InterBase 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.

1. SET TERM

Normally Firebird/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, Firebird /InterBase assumes that the command has finished when it arrives at the first semi colon.

In order for Firebird/InterBase 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
```

e.g. `SET TERM ^;`

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:

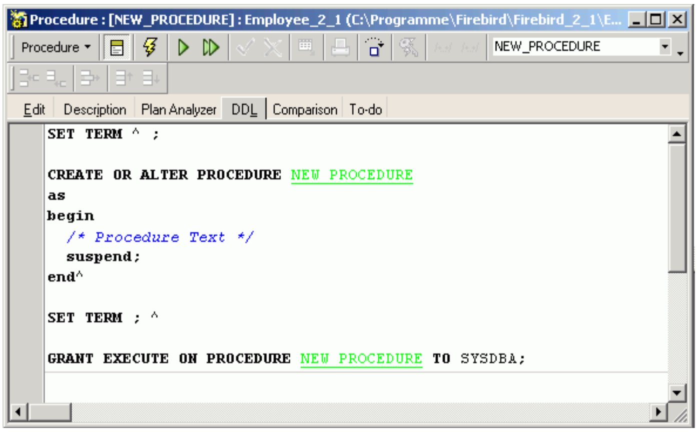
```
        SET TERM OLD_TERMINATOR NEW_TERMINATOR
```

e.g. `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).

When using the IBExpert Procedure Editor, the procedure templates already include this code, so there is no need 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:


Procedure : [NEW_PROCEDURE] : Employee_2_1 (C:\Programme\Firebird\Firebird_2_1\E...
Procedure
NEW_PROCEDURE
Edit
Description
Plan Analyzer
DDL
Comparison
To-do

```
SET TERM ^ ;

CREATE OR ALTER PROCEDURE NEW_PROCEDURE
as
begin
  /* Procedure Text */
  suspend;
end^

SET TERM ; ^

GRANT EXECUTE ON PROCEDURE NEW_PROCEDURE TO SYSDBA;
```

2. Stored procedure

Firebird/InterBase uses stored procedures as the programming environment for integrating active processes in the database. 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.

3. 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
```

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

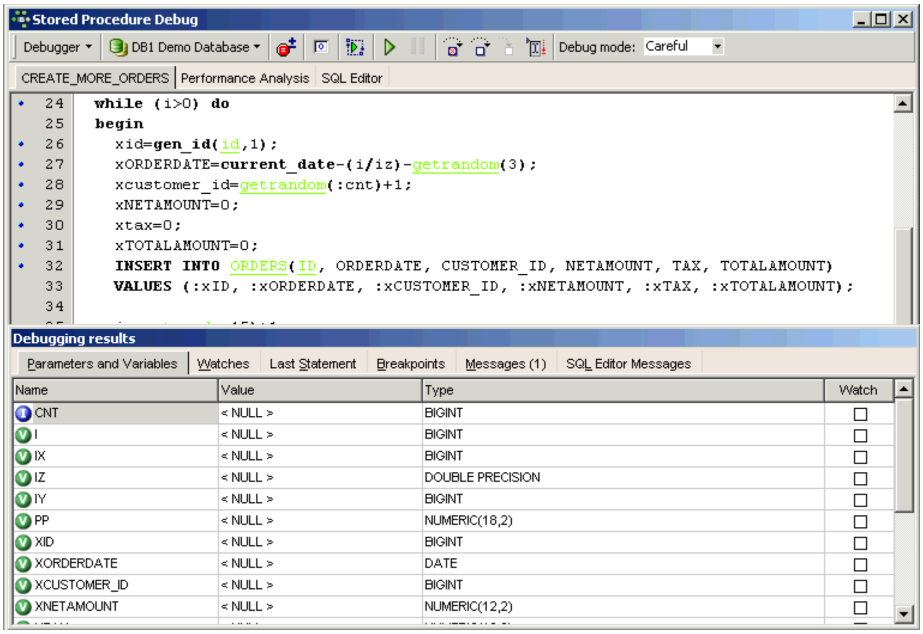
A good example of this is a typical employee table in a large hierarchical company, where the table has a column containing a pointer to the employees' boss. Every employee has a boss, and the bosses have bosses, who may also have bosses. If you wished to see a list of all bosses for one individual or 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 no one (the CEO).

9. 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.

9.1 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.



Debug mode: Careful

```
while (i>0) do
begin
  xid=gen_id(id,1);
  xORDERDATE=current_date-(i/iz)-getrandom(3);
  xcustomer_id=getrandom(:cnt)+1;
  xNETAMOUNT=0;
  xtax=0;
  xTOTALAMOUNT=0;
  INSERT INTO ORDERS(ID, ORDERDATE, CUSTOMER_ID, NETAMOUNT, TAX, TOTALAMOUNT)
  VALUES (:xID, :xORDERDATE, :xCUSTOMER_ID, :xNETAMOUNT, :xTAX, :xTOTALAMOUNT);
```

Name	Value	Type	Watch
CNT	< NULL >	BIGINT	☐
I	< NULL >	BIGINT	☐
IX	< NULL >	BIGINT	☐
IZ	< NULL >	DOUBLE PRECISION	☐
IY	< NULL >	BIGINT	☐
PP	< NULL >	NUMERIC(18,2)	☐
XID	< NULL >	BIGINT	☐
XORDERDATE	< NULL >	DATE	☐
XCUSTOMER_ID	< NULL >	BIGINT	☐
XNETAMOUNT	< NULL >	NUMERIC(12,2)	☐

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.

The *Debugger* can be opened from the *Stored Procedure* or *Trigger Editors* by clicking the *Debug* icon.

Two debug modes are offered: *Careful* and *Fast*. In the default debug mode, *Careful*, a corresponding `SELECT` statement is composed and executed on the server side. The *Fast* mode executes certain statements, such as simple assignments and boolean expressions of `IF/WHILE` statements, on the client side if possible. The *Fast* mode should be used for example, if you need to repeatedly execute a loop, which contains statements that can be calculated on the client side, as this will greatly reduce total execution time. Select the preferred option using the drop-down list in the top right-hand corner before starting the debug process.

The upper half of this dialog displays the SQL text. The object name (if applicable) is displayed in the *Windows* bar. The lower area displays a number of pages.

The first, *Parameters and Variables*, lists the parameters 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 data type. The *Watch* boxes can be checked, to specify which variables should be observed.

The variable contents can be viewed in the *Value* column or by directly by holding the mouse over the variable name in the code itself.

It is possible 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.

Universal triggers which use the context variables `INSERTING/UPDATING/DELETING` can also be debugged here. The debugger interprets these variables as regular input parameters with a `BOOLEAN` data type and they are `FALSE` by default.

The *Watches* page displays those parameters and variables that have been checked for particular observation in the previous window. Following execution, the last internal statement is displayed on the *Statements* page,, along with additional information such as execution time.

The *Breakpoints* 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.

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.

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.

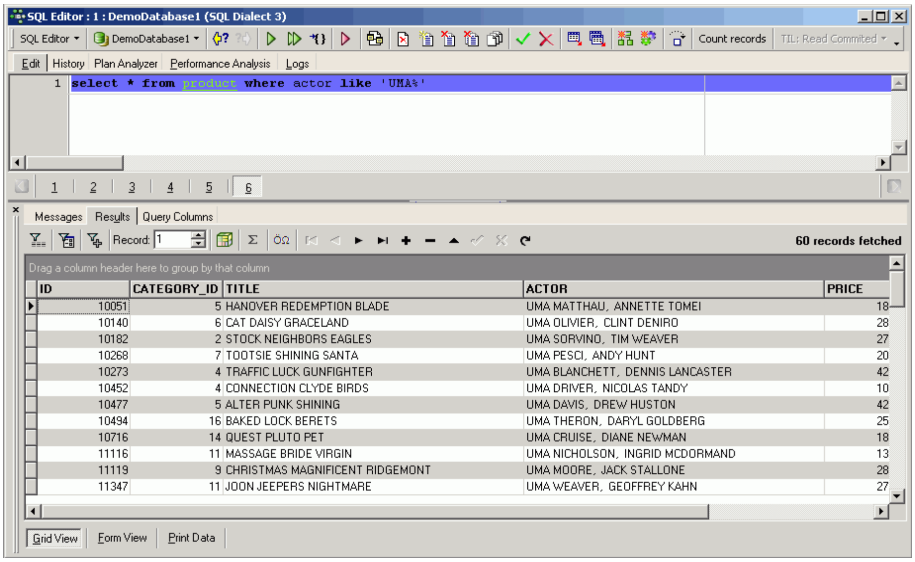
When debugging a procedure, first take a look at the values of the parameters and then use [F8] to go through the procedure step by step ([F9] executes fully). After each step, all variable values can be seen. Don't forget to work with breakpoints [F5].

10. 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 and 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.

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 demo database, `db1`, compare:

```
        select * from product where actor like 'UMA%'
```



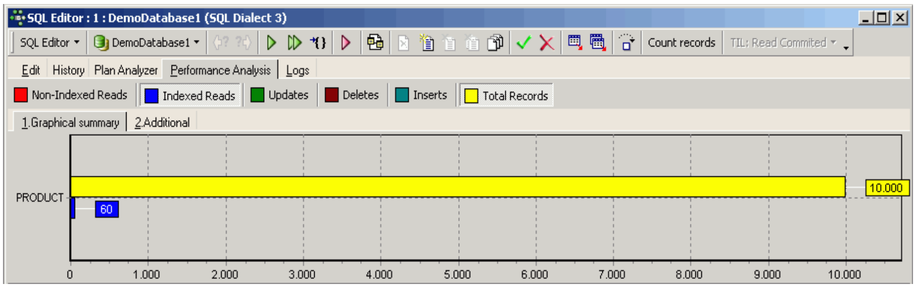
SQL Editor : 1 : DemoDatabase1 (SQL Dialect 3)

```
select * from product where actor like 'UMA%'
```

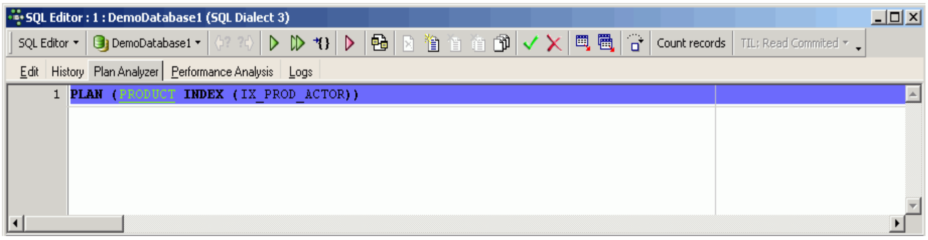
60 records fetched

ID	CATEGORY_ID	TITLE	ACTOR	PRICE
10051	5	HANOVER REDEMPTION BLADE	UMA MATTHAU, ANNETTE TOMEI	18
10140	6	CAT DAISY GRACELAND	UMA OLIVIER, CLINT DENIRO	28
10182	2	STOCK NEIGHBORS EAGLES	UMA SORVINO, TIM WEAVER	27
10268	7	TOOTSIE SHINING SANTA	UMA PESCI, ANDY HUNT	20
10273	4	TRAFFIC LUCK GUNFIGHTER	UMA BLANCHETT, DENNIS LANCASTER	42
10452	4	CONNECTION CLYDE BIRDS	UMA DRIVER, NICOLAS TANDY	10
10477	5	ALTER PUNK SHINING	UMA DAVIS, DREW HUSTON	42
10494	16	BAKED LOCK BERETS	UMA THERON, DARYL GOLDBERG	25
10716	14	QUEST PLUTO PET	UMA CRUISE, DIANE NEWMAN	18
11116	11	MASSAGE BRIDE VIRGIN	UMA NICHOLSON, INGRID MCDORMAND	13
11119	9	CHRISTMAS MAGNIFICENT RIDGEMONT	UMA MOORE, JACK STALLONE	28
11347	11	JOON JEEPERS NIGHTMARE	UMA WEAVER, GEOFFREY KAHN	27

The server returns all data sets beginning with the name `UMA`. If you examine the *Performance Analysis*:



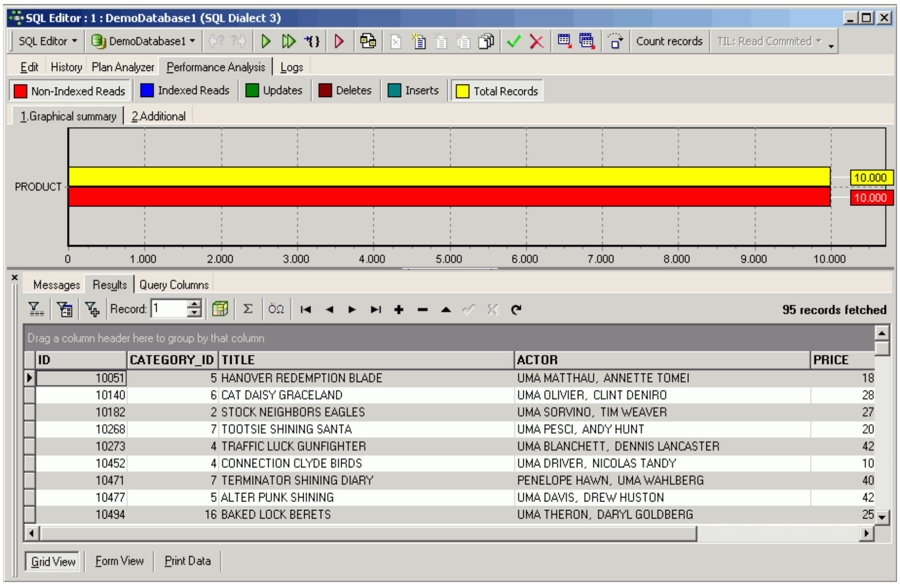
you will see that 60 indexed read operations were performed, and the *Plan Analysis* shows that the `IX_PROD_ACTOR` index was used:



```
PLAN (PRODUCT INDEX (IX_PROD_ACTOR))
```

If however you need to view all records where the name `UMA` appears somewhere in the `ACTOR` field:

```
select * from product where actor like ''
```



95 records fetched

ID	CATEGORY_ID	TITLE	ACTOR	PRICE
10051	5	HANOVER REDEMPTION BLADE	UMA MATTHAU, ANNETTE TOMEI	18
10140	6	CAT DAISY GRACELAND	UMA OLIVIER, CLINT DENIRO	28
10182	2	STOCK NEIGHBORS EAGLES	UMA SORVINO, TIM WEAVER	27
10268	7	TOOTSIE SHINING SANTA	UMA PESCI, ANDY HUNT	20
10273	4	TRAFFIC LUCK GUNFIGHTER	UMA BLANCHETT, DENNIS LANCASTER	42
10452	4	CONNECTION CLYDE BIRDS	UMA DRIVER, NICOLAS TANDY	10
10471	7	TERMINATOR SHINING DIARY	PENELOPE HAWN, UMA WAHLBERG	40
10477	5	ALTER PUNK SHINING	UMA DAVIS, DREW HUSTON	42
10494	16	BAKED LOCK BERETS	UMA THERON, DARYL GOLDBERG	25

Now the server has had to perform 10,000 non-indexed reads to fetch 95 records, rather more than the 60 reads for the 60 resulting records 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 and the *SP/Triggers/Views Analyzer*, check the index plans, not forgetting to recompute the selectivity of your indices regularly. Check for indices on columns used in `WHERE` and `JOIN` clauses. Use the *Plan Analyzer* and *Performance Analysis* to help you compare and improve your more complex procedures.

Another consideration with 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.

11. 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.

12. 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 online 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.

13. 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
```

14. Using domains in stored procedures

Introduced in Firebird 2.1, this feature finally allows developers to declare local variables and input and output arguments for stored procedures using domains in lieu of canonical data types. In earlier Firebird versions it was necessary to write the data type of the domain instead of the domain name. This meant a time-consuming checking of domain data types, which then had to be written in the procedure definition. For example:

```
create procedure insert_orderline(
  article_name varchar(50),
  price decimal(15,2)
  active smallint
)
begin
  ...
end
```

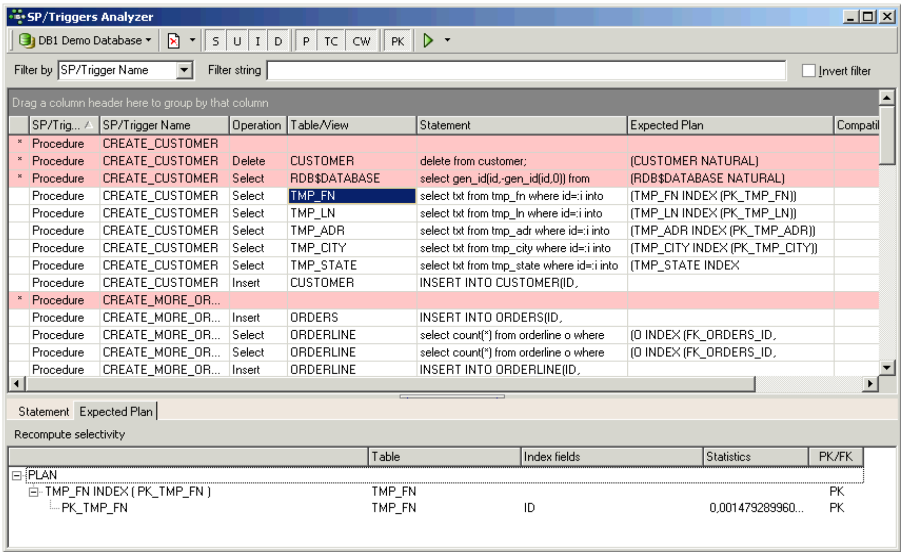
In Firebird 2.1 you can either type the domain name if you also want any `CHECK` clauses and default values to be taken into consideration, or use the `TYPE OF` keyword if you just want the data type. The above example would then look something like this:

```
create procedure insert_orderline(
  article_name string,
  price currency,
  active bool
)
begin
  ...
end
```

Chapter 13: SP/Triggers/Views Analyzer

The *Stored Procedure/Trigger/Views Analyzer* allows you to analyze a selection of actions for all or a filtered selection of procedures, triggers and views in a database, providing information by statement, displaying plans and indices used, issuing compatibility warnings and compiler warnings for all objects analyzed. 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 Firebird/InterBase version, the procedures and triggers appear not to work; here 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 drop-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.



	SP/Trig...	SP/Trigger Name	Operation	Table/View	Statement	Expected Plan	Compatil
*	Procedure	CREATE_CUSTOMER					
*	Procedure	CREATE_CUSTOMER	Delete	CUSTOMER	delete from customer;	(CUSTOMER NATURAL)	
*	Procedure	CREATE_CUSTOMER	Select	RDB$DATABASE	select gen_id(id,-gen_id(id,0)) from	(RDB$DATABASE NATURAL)	
	Procedure	CREATE_CUSTOMER	Select	TMP_FN	select txt from tmp_fn where id=:i into	(TMP_FN INDEX (PK_TMP_FN))	
	Procedure	CREATE_CUSTOMER	Select	TMP_LN	select txt from tmp_ln where id=:i into	(TMP_LN INDEX (PK_TMP_LN))	
	Procedure	CREATE_CUSTOMER	Select	TMP_ADR	select txt from tmp_adr where id=:i into	(TMP_ADR INDEX (PK_TMP_ADR))	
	Procedure	CREATE_CUSTOMER	Select	TMP_CITY	select txt from tmp_city where id=:i into	(TMP_CITY INDEX (PK_TMP_CITY))	
	Procedure	CREATE_CUSTOMER	Select	TMP_STATE	select txt from tmp_state where id=:i into	(TMP_STATE INDEX	
	Procedure	CREATE_CUSTOMER	Insert	CUSTOMER	INSERT INTO CUSTOMER(ID,		
*	Procedure	CREATE_MORE_OR...					
	Procedure	CREATE_MORE_OR...	Insert	ORDERS	INSERT INTO ORDERS(ID,		
	Procedure	CREATE_MORE_OR...	Select	ORDERLINE	select count(*) from orderline o where	(O INDEX (FK_ORDERS_ID,	
	Procedure	CREATE_MORE_OR...	Select	ORDERLINE	select count(*) from orderline o where	(O INDEX (FK_ORDERS_ID,	
	Procedure	CREATE_MORE_OR...	Insert	ORDERLINE	INSERT INTO ORDERLINE(ID,		

	Table	Index fields	Statistics	PK/FK
PLAN				
TMP_FN INDEX (PK_TMP_FN)	TMP_FN			PK
PK_TMP_FN	TMP_FN	ID	0,001479289960...	PK

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.

The user can specify exactly what should be analyzed by deactivating or activating the toolbar icons (*SP/Triggers/Views Analyzer* toolbar):

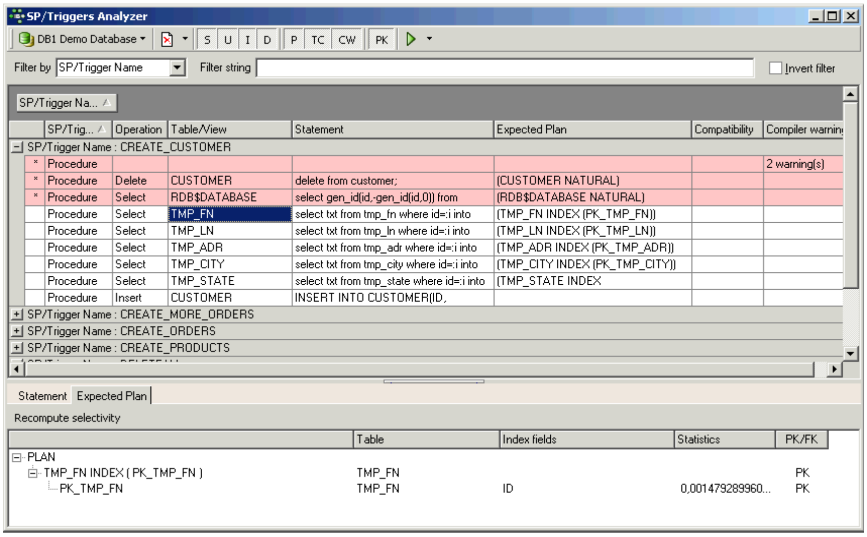
S	All `SELECT` statements are selected, analyzed and displayed
U	All `UPDATE` statements are selected, analyzed and displayed.
I	All `INSERT` statements are selected, analyzed and displayed.
D	All `DELETE` statements are selected, analyzed and displayed.
P	Analysis of plans and indices.
TC	Analysis of 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 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:



Filter by: SP/Trigger Name
Filter string:
Invert filter

SP/Trigger Na...

	SP/Trig...	Operation	Table/View	Statement	Expected Plan	Compatibility	Compiler warnin
− SP/Trigger Name : CREATE_CUSTOMER							
*	Procedure						2 warning(s)
*	Procedure	Delete	CUSTOMER	delete from customer;	(CUSTOMER NATURAL)		
*	Procedure	Select	RDB$DATABASE	select gen_id(id,-gen_id(id,0)) from	(RDB$DATABASE NATURAL)		
	Procedure	Select	TMP_FN	select txt from tmp_fn where id=:i into	(TMP_FN INDEX (PK_TMP_FN))		
	Procedure	Select	TMP_LN	select txt from tmp_ln where id=:i into	(TMP_LN INDEX (PK_TMP_LN))		
	Procedure	Select	TMP_ADR	select txt from tmp_adr where id=:i into	(TMP_ADR INDEX (PK_TMP_ADR))		
	Procedure	Select	TMP_CITY	select txt from tmp_city where id=:i into	(TMP_CITY INDEX (PK_TMP_CITY))		
	Procedure	Select	TMP_STATE	select txt from tmp_state where id=:i into	(TMP_STATE INDEX		
	Procedure	Insert	CUSTOMER	INSERT INTO CUSTOMER(ID,			
+ SP/Trigger Name : CREATE_MORE_ORDERS							
+ SP/Trigger Name : CREATE_ORDERS							
+ SP/Trigger Name : CREATE_PRODUCTS							

Statement | Expected Plan

Recompute selectivity

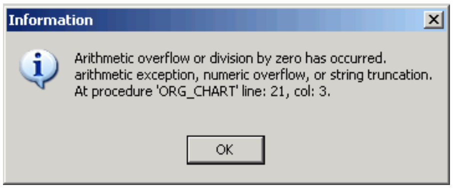
	Table	Index fields	Statistics	PK/FK
− PLAN				
− TMP_FN INDEX (PK_TMP_FN)	TMP_FN			PK
PK_TMP_FN	TMP_FN	ID	0,001479289960...	PK

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. The statements can easily be copied and inserted into a text editor or the IBExpert *SQL Editor*, using the context-sensitive right-click menu.

Chapter 14: Writing Exceptions

Care should be taken when writing exceptions; when you are dealing with hundreds of thousands of data sets, you need to source your problem quickly. For example, a message such as this:


Information

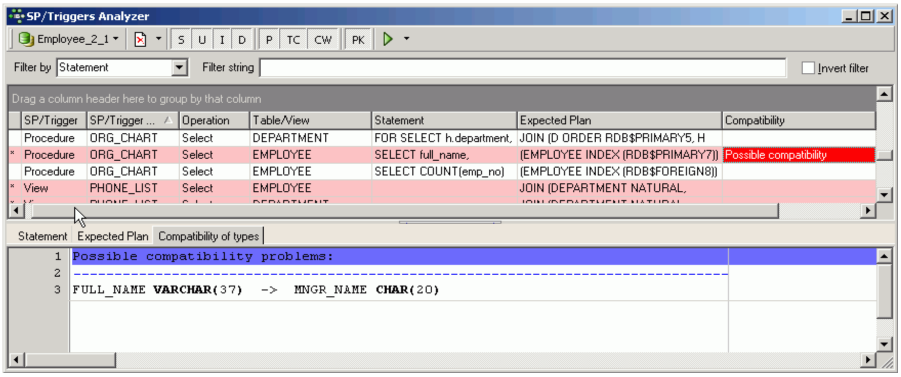
Arithmetic overflow or division by zero has occurred.
arithmetic exception, numeric overflow, or string truncation.
At procedure 'ORG_CHART' line: 21, col: 3.

could well occur after a database has been up and successfully running for years.

To detect such errors on a customer database where a lot of stored procedures with several hundred steps have been written, and which may run several hundred thousand times before the error occurs, is of course difficult.

This particular error message occurred because a data set was altered in the Firebird sample `EMPLOYEE` table; the first record, `Robert Nelson`, was amended to `Robert Joseph Nelson-Katzenberger`. This in itself is not an error, as the fields `FIRST_NAME` and `LAST_NAME` have been specified as `varchar(15)` and `varchar(20)` respectively.

The problem arises when the stored procedure, `ORG_CHART`, is executed. If we take a look at the IBExpert *Tools* menu item, *SP/Triggers/Views Analyzer*, we can see where the problem lies:


SP/Triggers Analyzer

Filter by: Statement   Filter string: Invert filter

Drag a column header here to group by that column

	SP/Trigger	SP/Trigger ...	Operation	Table/View	Statement	Expected Plan	Compatibility
	Procedure	ORG_CHART	Select	DEPARTMENT	FOR SELECT h.department,	JOIN (D ORDER RDB$PRIMARY5, H	
*	Procedure	ORG_CHART	Select	EMPLOYEE	SELECT full_name,	(EMPLOYEE INDEX (RDB$PRIMARY7))	Possible compatibility
	Procedure	ORG_CHART	Select	EMPLOYEE	SELECT COUNT(emp_no)	(EMPLOYEE INDEX (RDB$FOREIGN8))	
*	View	PHONE_LIST	Select	EMPLOYEE		JOIN (DEPARTMENT NATURAL,	

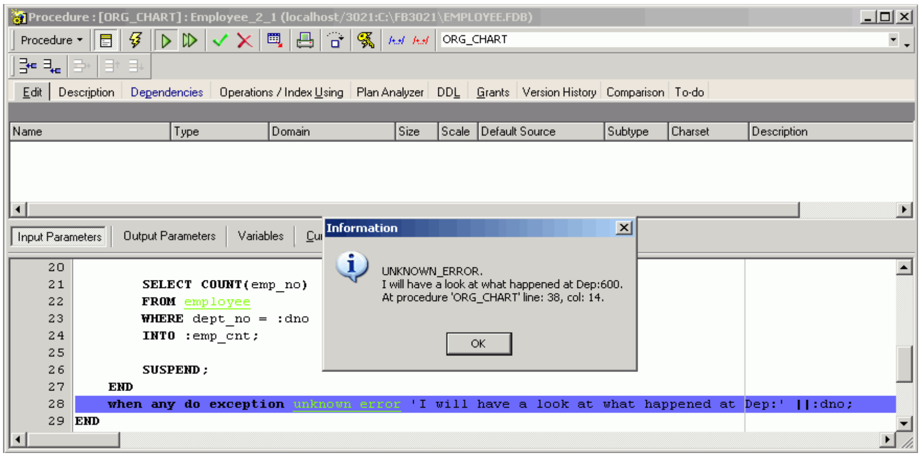
Statement | Expected Plan | Compatibility of types

```
Possible compatibility problems:
--------------------------------------------------------------------------------
FULL_NAME VARCHAR(37)  ->  MNGR_NAME CHAR(20)
```

The stored procedure's output parameter, `MNGR_NAME` has been defined as `varchar(20)`; however the `FULL_NAME` from the `EMPLOYEE` table is specified at `varchar(37)`. So, any `FIRST_NAME_LAST_NAME` combination containing a total of more than 20 characters, will fire an exception.

You can then go on to use the stored procedure debugger to trace the data set that has caused the problem. This process can however be very time-consuming when you have hundreds of thousands of data sets.

In Firebird it is possible to do the following: write an exception which will give you more information when this error occurs. Create a new exception, name it, add the exception message to your procedure, which should appear when the error occurs, e.g. `when any do exception unknown_error`;. And a really useful feature since Firebird 1.5: you can change the text on the fly (i.e. in the procedure itself).It is even possible to combine the message with, for example, the department number:



UNKNOWN_ERROR.
I will have a look at what happened at Dep:600.
At procedure 'ORG_CHART' line: 38, col: 14.

```
        SELECT COUNT(emp_no)
        FROM employee
        WHERE dept_no = :dno
        INTO :emp_cnt;

        SUSPEND;
    END
    when any do exception unknown_error 'I will have a look at what happened at Dep:' ||:dno;
END
```

This is much more useful for the error finding process, as you have already narrowed down the error to a specific department number. This is possible in every stored procedure that you use. The more obvious solution would be, in this case, to alter the return parameter, `MNGR_NAME` from `char(20)` to, for example, `char(40)`.

So, when writing stored procedures, incorporate exception messages that indicate the source of the problem. Use the on-the-fly possibility to define detailed exception messages in your procedures.

Chapter 15: Logging

Databases are 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 a database.

The IBExpert *Log Manager* can be started from the IBExpert *Tools* menu.

Simply 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,

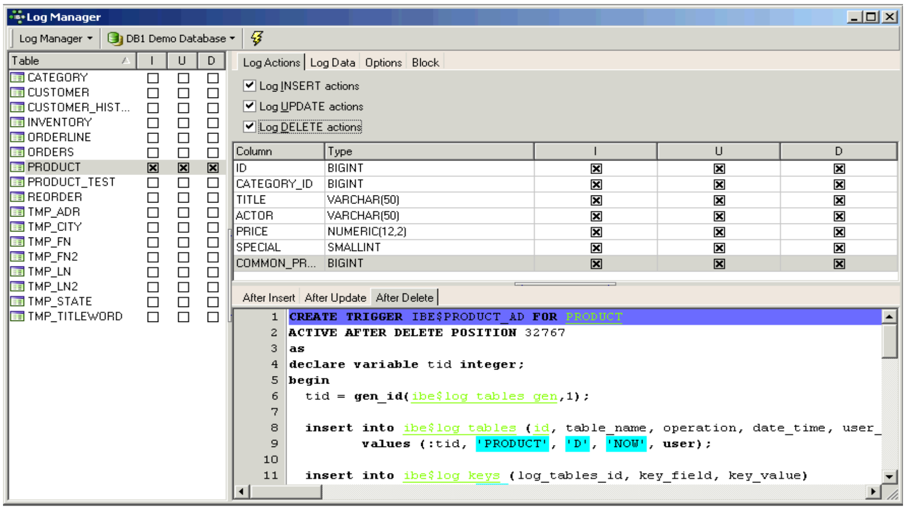


Table	I	U	D
CATEGORY	☐	☐	☐
CUSTOMER	☐	☐	☐
CUSTOMER_HIST...	☐	☐	☐
INVENTORY	☐	☐	☐
ORDERLINE	☐	☐	☐
ORDERS	☐	☐	☐
PRODUCT	☒	☒	☒
PRODUCT_TEST	☐	☐	☐
REORDER	☐	☐	☐
TMP_ADR	☐	☐	☐
TMP_CITY	☐	☐	☐
TMP_FN	☐	☐	☐
TMP_FN2	☐	☐	☐
TMP_LN	☐	☐	☐
TMP_LN2	☐	☐	☐
TMP_STATE	☐	☐	☐
TMP_TITLEWORD	☐	☐	☐

✓ Log INSERT actions
✓ Log UPDATE actions
✓ Log DELETE actions

Column	Type	I	U	D
ID	BIGINT	☒	☒	☒
CATEGORY_ID	BIGINT	☒	☒	☒
TITLE	VARCHAR(50)	☒	☒	☒
ACTOR	VARCHAR(50)	☒	☒	☒
PRICE	NUMERIC(12,2)	☒	☒	☒
SPECIAL	SMALLINT	☒	☒	☒
COMMON_PR...	BIGINT	☒	☒	☒

```
CREATE TRIGGER IBE$PRODUCT_AD FOR PRODUCT
ACTIVE AFTER DELETE POSITION 32767
as
declare variable tid integer;
begin
  tid = gen_id(ibe$log_tables_gen,1);

  insert into ibe$log_tables (id, table_name, operation, date_time, user_
         values (:tid, 'PRODUCT', 'D', 'NOW', user);

  insert into ibe$log_keys (log_tables_id, key_field, key_value)
```

below which the selected table's fields and field data types are displayed. A log script can be generated for several tables simultaneously by selecting the required tables using the [Ctrl + Shift] keys. The logging options, for example which *INSERT*, *UPDATE* and *DELETE* actions on which tables, can be checked individually or alternatively, the *Log Manager* drop-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.

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. The data logging triggers templates can be altered as wished using the IBExpert *Options* menu item, *General Templates* (*Data Logging Triggers*).

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 opens a new window, 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*.

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).

After compiling and committing, all the specified actions will now be logged:

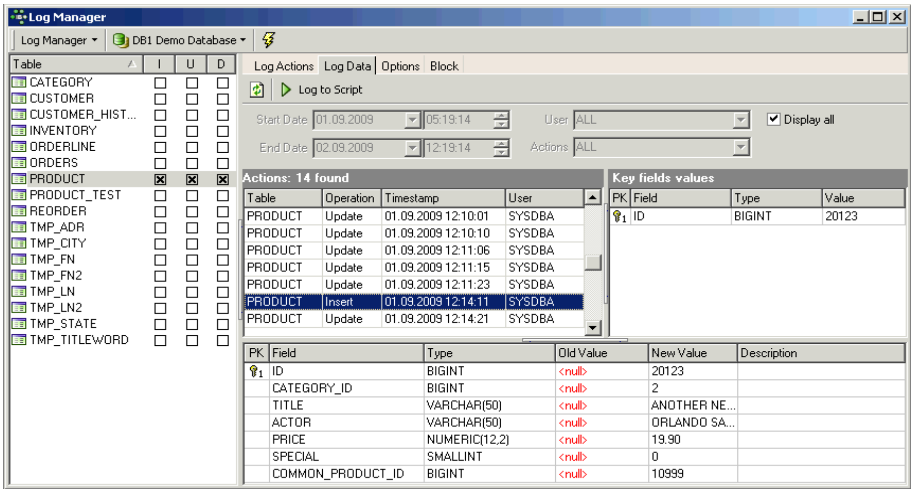


Table	I	U	D
CATEGORY	☐	☐	☐
CUSTOMER	☐	☐	☐
CUSTOMER_HIST...	☐	☐	☐
INVENTORY	☐	☐	☐
ORDERLINE	☐	☐	☐
ORDERS	☐	☐	☐
PRODUCT	☒	☒	☒
PRODUCT_TEST	☐	☐	☐
REORDER	☐	☐	☐
TMP_ADR	☐	☐	☐
TMP_CITY	☐	☐	☐
TMP_FN	☐	☐	☐
TMP_FN2	☐	☐	☐
TMP_LN	☐	☐	☐
TMP_LN2	☐	☐	☐
TMP_STATE	☐	☐	☐
TMP_TITLEWORD	☐	☐	☐

Start Date 01.09.2009 05:19:14 User ALL ✓ Display all
End Date 02.09.2009 12:19:14 Actions ALL

Actions: 14 found

Table	Operation	Timestamp	User
PRODUCT	Update	01.09.2009 12:10:01	SYSDBA
PRODUCT	Update	01.09.2009 12:10:10	SYSDBA
PRODUCT	Update	01.09.2009 12:11:06	SYSDBA
PRODUCT	Update	01.09.2009 12:11:15	SYSDBA
PRODUCT	Update	01.09.2009 12:11:23	SYSDBA
PRODUCT	Insert	01.09.2009 12:14:11	SYSDBA
PRODUCT	Update	01.09.2009 12:14:21	SYSDBA

Key fields values

PK	Field	Type	Value
1	ID	BIGINT	20123

PK	Field	Type	Old Value	New Value	Description
1	ID	BIGINT	<null>	20123	
	CATEGORY_ID	BIGINT	<null>	2	
	TITLE	VARCHAR(50)	<null>	ANOTHER NE...	
	ACTOR	VARCHAR(50)	<null>	ORLANDO SA...	
	PRICE	NUMERIC(12,2)	<null>	19.90	
	SPECIAL	SMALLINT	<null>	0	
	COMMON_PRODUCT_ID	BIGINT	<null>	10999	

When using the *Log Manager* for the first time, it is necessary to confirm the creation of a number of IBExpert system tables.

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.

1. Understanding the log file

Take the time to look at your logs and search for patterns emerging over a period of time, as the source of many problems often go back quite a long time (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: unfortunately this one's pretty terminal, because it's a clear indication that the database is corrupt. It is important to determine which pages are affected, because they may 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!

There are a number of articles concerning database problems and repair at the IBExpert online documentation site.

Chapter 16: Database Backup & Restore

1. Why is a database backup and restore important?

Performing regular backups protects from hardware failures and data corruption, which cannot be fixed by the Firebird/InterBase maintenance tools. It is important to use the Firebird/InterBase 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 Firebird/InterBase backup runs parallel with concurrent database accesses by other users. Firebird/InterBase 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. Also all files in a multifile database are backed up. Firebird/InterBase 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 Firebird/InterBase backup backs up all database files automatically. The different versions of Firebird/InterBase 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 Firebird/InterBase backup utility allows a transportable backup format, so that this file can be restored on any desired Firebird/InterBase platform. 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.

The Firebird/InterBase 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. Note that since 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 a Firebird/InterBase 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).

IBExpert offers two methods for backing up and restoring databases: IBExpert *Services* menu items, *Backup Database* and *Restore Database*, as well as the *IBExpertBackupRestore* service for automating regular backups. Alternatively the Firebird command-line tool, `GBAK` can also be used. Please note that if you run the `GBAK` restore in verbose mode, it can take an awful long time.

2. 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 above.

A database backup may be performed without having to disconnect the database; users may continue their work as Firebird/InterBase 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 drop-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).

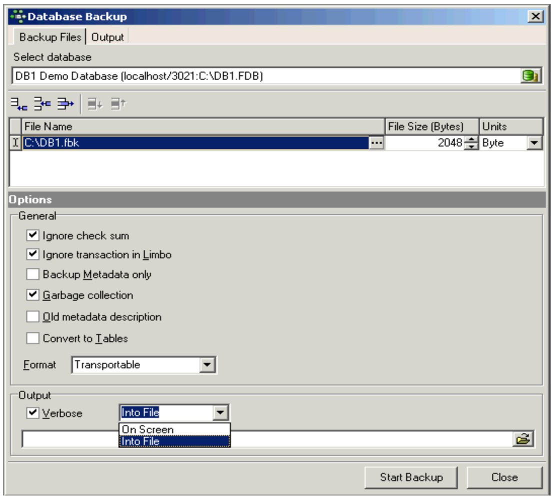
The [...] button to the right of this row allows you to find an existing file or specify the drive, path and backup file name for a new file. Please note that IBExpert will only create a file name on the server, and not locally (as with `GBAK`), because IBExpert uses the Services API. A local backup 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). Firebird/InterBase understands the links that exist with secondary database files and with shadows. Whereas the operating system backup works on a file-by-file basis, Firebird/InterBase always backs up all files in a database.

2.1 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 below 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 Firebird/InterBase 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 Firebird/InterBase 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**: This 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* for the log (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.



Select database: DB1 Demo Database (localhost/3021:C:\DB1.FDB)

File Name	File Size (Bytes)	Units
C:\DB1.fbk	2048	Byte

General

- [✓] Ignore check sum
- [✓] Ignore transaction in Limbo
- [ ] Backup Metadata only
- [✓] Garbage collection
- [ ] Old metadata description
- [ ] Convert to Tables

Format: Transportable

Output

- [✓] Verbose: Into File (On Screen / Into File)

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.

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.

To automate the backup/restore process for your databases, use the HK-Software Service Center's *IBExpertBackupRestore*.

2.2 Garbage collection

Garbage collection is the ongoing cleaning of the database and is performed in the background around the clock. This constantly reorganizes 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 manual maintenance as certain other database environments do. Whether the garbage collector works efficiently or not depends on how the application works. This is illustrated in more detail in *Chapter 27: Database Statistics*.

3. 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 *Chapter 6: 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.
- When you need to alter the database page size.
- A sweep is also automatically performed during a backup, so long as it has not been disabled.

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).

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 IBExpert *Restore Database* dialog requests specification of whether the restored database should overwrite an existing database, or whether a new database should be created. Then the backed up file needs to be selected. The following restore options may be checked/unchecked as wished:

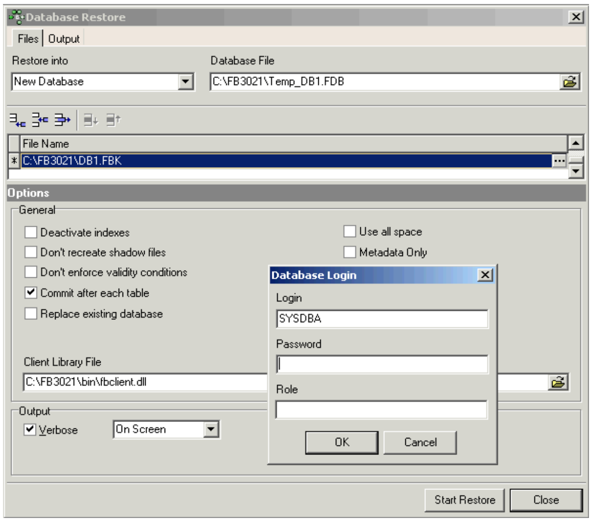
- **Deactivate indexes**: If this option is checked, database indices are deactivated while restoring. This option is used to improve restore performance. If this option is not checked, Firebird/InterBase updates indices after all tables have been populated with the restored rows. This option may also 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.
- **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, Firebird/InterBase 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, work is committed 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, Firebird/InterBase 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.
- **Page size**: Database page size in bytes. This is the only option allowing the page size for an existing database to be altered.
- **Client Library File**: This 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 that specified in the IBExpert *Options* menu item, *Environment Options*.
- **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.

You will be asked to log in before the restore can start:



Please note that Firebird/InterBase does not backup indices. It only backs up the index definition. When the database is restored Firebird/InterBase uses this definition to regenerate the indices.

Using the *Database Registration Info* menu item, *Backup/Restore*, 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.

3.1 Working with shadows

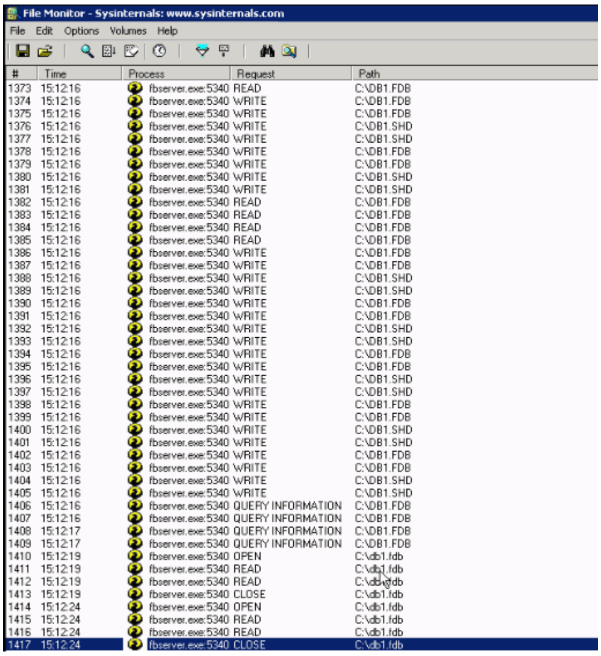
A shadow is a physical copy of the database file. When you need more than the typical level of security, it is possible to add a second hard disk on your machine and make a shadow copy of your database. When you have all data on one computer and it breaks down, you need rapid access to the data. Therefore you should typically create a shadow file on an external USB hard disk. Simply specify on this hard disk:

```
      create shadow 1 'C:\db1.shd'
```

It is possible to create a shadow during runtime. You don't just have to create one shadow, you can create ten if you like, although it is not necessarily a good idea because each writing process must then be done eleven times and not just two!

After committing, the whole database file is copied to the shadow. This is also the fastest way to create a simple database copy on a USB disk, as an alternative to a a backup and restore which takes some time, especially the restore, because in the restore process, the indices are rewritten. You then simply need to use `GFIX` to activate it to turn it into an active database when needed. The `CREATE SHADOW` command makes a physical copy of the database pages from the original file to the shadow file, without thinking about what is written inside.

When viewing a file monitor filtered for operations on the `db1` database, there are a number of operations on both the database file (`.fdb`) and the shadow file (`.shd`):


File Monitor - Sysinternals: www.sysinternals.com
File Edit Options Volumes Help

#	Time	Process	Request	Path
1373	15:12:16	fbserver.exe:5340	READ	C:\DB1.FDB
1374	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1375	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1376	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1377	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1378	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1379	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1380	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1381	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1382	15:12:16	fbserver.exe:5340	READ	C:\DB1.FDB
1383	15:12:16	fbserver.exe:5340	READ	C:\DB1.FDB
1384	15:12:16	fbserver.exe:5340	READ	C:\DB1.FDB
1385	15:12:16	fbserver.exe:5340	READ	C:\DB1.FDB
1386	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1387	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1388	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1389	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1390	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1391	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1392	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1393	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1394	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1395	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1396	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1397	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1398	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1399	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1400	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1401	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1402	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1403	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.FDB
1404	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1405	15:12:16	fbserver.exe:5340	WRITE	C:\DB1.SHD
1406	15:12:16	fbserver.exe:5340	QUERY INFORMATION	C:\DB1.FDB
1407	15:12:16	fbserver.exe:5340	QUERY INFORMATION	C:\DB1.FDB
1408	15:12:17	fbserver.exe:5340	QUERY INFORMATION	C:\DB1.FDB
1409	15:12:17	fbserver.exe:5340	QUERY INFORMATION	C:\DB1.FDB
1410	15:12:19	fbserver.exe:5340	OPEN	C:\db1.fdb
1411	15:12:19	fbserver.exe:5340	READ	C:\db1.fdb
1412	15:12:19	fbserver.exe:5340	READ	C:\db1.fdb
1413	15:12:19	fbserver.exe:5340	CLOSE	C:\db1.fdb
1414	15:12:24	fbserver.exe:5340	OPEN	C:\db1.fdb
1415	15:12:24	fbserver.exe:5340	READ	C:\db1.fdb
1416	15:12:24	fbserver.exe:5340	READ	C:\db1.fdb
1417	15:12:24	fbserver.exe:5340	CLOSE	C:\db1.fdb

In a production environment, when a select is performed for example, all the read statements are done on the `fdb` file, the shadow file is only used for writing. In spite of this it is still recommend you use a very fast external hard drive for an active shadow, as you may notice a drop in performance with the double amount of write operations now being made to both databases.

If you encounter the problem that the shadow file is now the only file that you have, because your main computer has broken down, to turn the shadow file into a valid database, you need to use the Firebird/InterBase tool, `gfix`. On the command line type:

```
        C:\> gfix localhost/3021:C:\db1.shd –activate
```

and that's all you need to enter to get a valid database!

So for example, you need a copy of your main database which is 5 GB large; you just create a shadow on a USB hard disk, commit the shadow, and after everything is committed, pull it out. *Forced Writes* handles both databases in the same way. When something is written in the TIP, it is written in both files. When it is written on a page at the end of the file, it is written at the end of both files. So you always have the possibility to make a high speed backup of your database, and you can activate this backup with a simple batch command to turn it into a valid database.

In a shadow you have the same problems as in your main database. For example, when you have deleted all the records in your main database and you have already committed it, you will have the same problem in your shadow, because it is a physical copy of your main database file, nothing else. So if you need an "undo" of your transactions, you need another concept, for example, a transaction log.

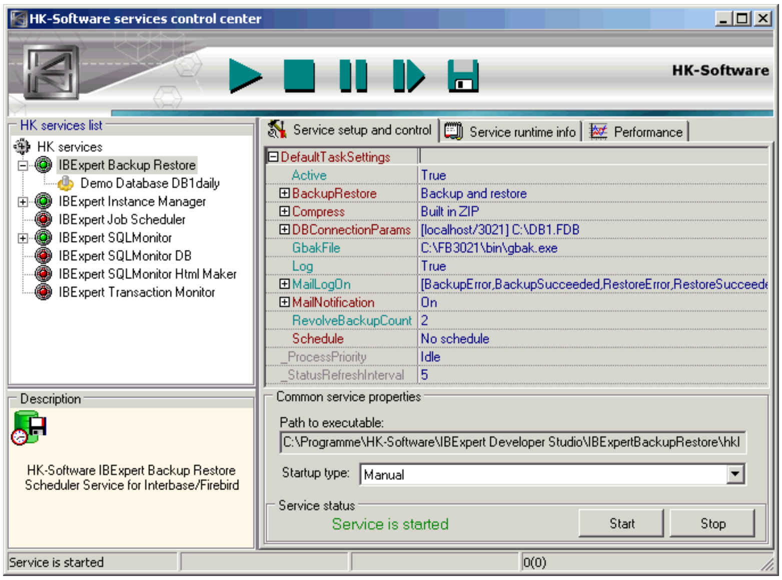
If the shadow is no longer available, the main database ignores the shadow (default setting). This is, for example, the reason why you can put a shadow on the USB hard disk, and directly remove it. The database server with the typical settings does not need to stop its work when the shadow is no longer available. In order to reactivate the shadow you will need to specify the `CREATE SHADOW` command again. For security reasons it is possible to specify that no new transactions can be started when the shadow is no longer available. However this option is seldom used, because the moment one of the hard disks stops working, no one can work at all. These options can be specified in the `CREATE SHADOW` statement.

4. Automating the database backup and restore

It is possible to automate the database backup in a batch file in the *Windows Scheduled Tasks*. Although a great tool for automating your backups and restores is the IBExpert Server Tool, *IBExpertBackupRestore*. *IBExpertBackupRestore* is a 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.

4.1 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.


HK-Software services control center
HK-Software
HK services list
HK services
IBExpert Backup Restore
Demo Database DB1daily
IBExpert Instance Manager
IBExpert Job Scheduler
IBExpert SQLMonitor
IBExpert SQLMonitor DB
IBExpert SQLMonitor Html Maker
IBExpert Transaction Monitor
Service setup and control
Service runtime info
Performance
Description
HK-Software IBExpert Backup Restore Scheduler Service for Interbase/Firebird

DefaultTaskSettings	
Active	True
BackupRestore	Backup and restore
Compress	Built in ZIP
DBConnectionParams	[localhost/3021] C:\DB1.FDB
GbakFile	C:\FB3021\bin\gbak.exe
Log	True
MailLogOn	[BackupError,BackupSucceeded,RestoreError,RestoreSucceede
MailNotification	On
RevolveBackupCount	2
Schedule	No schedule
_ProcessPriority	Idle
_StatusRefreshInterval	5

Common service properties

Path to executable: C:\Programme\HK-Software\IBExpert Developer Studio\IBExpertBackupRestore\hkl

Startup type: Manual

Service status: Service is started

Start Stop

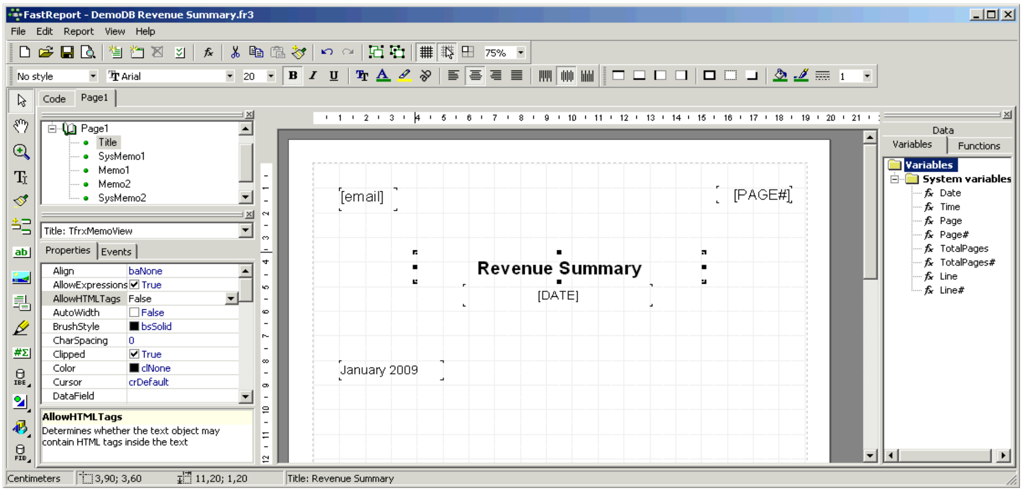
Service is started 0(0)

Here you can see the HK-*Software SCC* with the *IBExpertBackupRestore* configuration loaded. In the HK Services list tree view you can actually see the service item with one task below it. Each task is a database backup/restore schedule configuration.

Chapter 17: Reporting

The integrated IBExpert *Report Manager*, found in the IBExpert *Tools* menu, can be used to create automated reports.

With a right click on the alias name in the *Report Manager* you can create a new report inside your database. When starting the *Report Manager* for the first time, you will be asked to confirm creation of two IBExpert system tables. Following confirmation, you will see the *Report Manager* desktop:



Align	baNone
AllowExpressions	✓ True
AllowHTMLTags	False
AutoWidth	False
BrushStyle	■ bsSolid
CharSpacing	0
Clipped	✓ True
Color	■ clNone
Cursor	crDefault
DataField	

AllowHTMLTags
Determines whether the text object may contain HTML tags inside the text

In this example we have simply added two text areas with different font settings and a variable inside the text. Variables are always enclosed in `[ ]`. Later we have also added an image on the upper right with the company logo.

When you save the report and close the *Report Designer* and then double-click the report in the *Report Explorer*, you will see a preview, but since the variables are not set, there is no text where the variables were used. These variables can be set in an IBEBlock script using the IBExpert *SQL Editor*, which can also create the report and export it to different formats.

```
EXECUTE ibeblock
execute ibeblock
as
begin
    CRLF = ibec_CRLF();

    SELECT IBE$REPORT_SOURCE FROM ibe$reports
    where ibe$report_id = 1
    into :RepSrc;

    Params['email'] = 'hklemt@h-k.de';
    Report = ibec_CreateReport(RepSrc, Params, null);
```

```
     Res = ibec_ExportReport(Report, 'c:\r.html', __erHTML, 'Export-
Pictures=FALSE;');
     Res = ibec_ExportReport(Report, 'c:\r.pdf', __erPDF, 'Export-
Pictures=TRUE;');
     sMessage = '';
     sMessage=ibec_LoadFromFile('c:\r.html');
     ibec_smtp_SendMail('smtp.1und1.de',
                        '25',
                        'comm@ibexpert.biz',
                        'pwd',
                        '"IBExpert KG" <comm@ibexpert.biz>',
                        'hklemt@h-k.de',
                        '',
                        '',
                        'Test',
                        :sMessage,
                        'c:\r.pdf',
                        '',
                        'encoding="UTF-8";ContentType=text/html;
Priority=Highest','');
 end
```

The report is saved in the database in the `IBE$Reports` table. Using a `SELECT`, it is copied to a variable `RepSrc`. All parameters used in the report can be set in a `Params` array and transferred as the reference to the function `ibec_CreateReport`.

1. ibec_CreateReport

`ibec_CreateReport` prepares a report from a specified source (*FastReport*) and returns prepared report data.

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`.

Syntax

```
function ibec_CreateReport(ReportSource : string; Params : array of
variant; Options : string) : variant;
```

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
```

2. ibec_ExportReport

The `ibec_ExportReport` function is called twice using different formats.

Description

`ibec_ExportReport` exports report, created with the IBExpert *Report Manager* and prepared using the `ibec_CreateReport` function, into a specified format.

Syntax

```
 function ibec_ExportReport(PreparedReport : variant; FileName :
string; ExportType : integer; Options : string) : boolean;
```

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`.
`PrintOp-timized=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.
ExportPic-tures=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

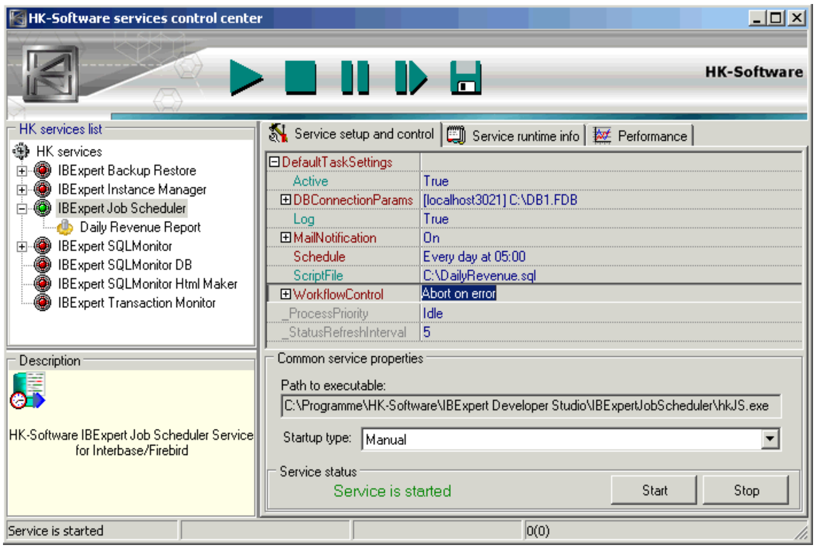
```
execute ibeblock
as
begin
  Params['HeaderMemo'] = '';
  Params['MEMO2'] = 2;
```

```
        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
```

After this is done, the HTML version is loaded from file into a variable and a new e-mail is sent using the `ibec_smtp_SendMail` function, where the PDF file is added as an attachment.

3. Job Automation with the IBExpertJobScheduler

IBExpertJobScheduler is one of the modules in the *HK-Software Services Control Center* found in the IBExpert *Services* menu. Use it to schedule regular jobs to run automatically, for example, if you wish a certain IBEBlock to be executed on a daily basis or every two hours. Specify mail notification of successful completion or only if an error has occurred.



HK services
- IBExpert Backup Restore
- IBExpert Instance Manager
- IBExpert Job Scheduler
  - Daily Revenue Report
- IBExpert SQLMonitor
- IBExpert SQLMonitor DB
- IBExpert SQLMonitor Html Maker
- IBExpert Transaction Monitor

DefaultTaskSettings	
Active	True
DBConnectionParams	[localhost3021] C:\DB1.FDB
Log	True
MailNotification	On
Schedule	Every day at 05:00
ScriptFile	C:\DailyRevenue.sql
WorkflowControl	Abort on error
_ProcessPriority	Idle
_StatusRefreshInterval	5

HK-Software IBExpert Job Scheduler Service for Interbase/Firebird

Common service properties

Path to executable: C:\Programme\HK-Software\IBExpert Developer Studio\IBExpertJobScheduler\hkJS.exe

Startup type: Manual

Service status: Service is started

Service is started | 0(0)

Chapter 18: Data Analysis

The IBExpert Tools menu item, *Data Analysis*, is an ideal OLAP and data warehouse component for quickly and easily analyzing data in the database. 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. In fact, IBExpert's *Data Analysis* offers innumerable possibilities to define reports quickly and easily, or to simply collate the data material.

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.

The functionalities and options available in the Pivot Cube will be illustrated using the following `SELECT` command, executed in the *SQL Editor*:

```
select distinct
    orderline.orderdate,
    orders.totalamount,
    category.txt,
    orders.netamount
from orders
   inner join orderline on (orders.id = orderline.orders_id)
   inner join product on (orderline.product_id = product.id)
   inner join category on (product.category_id = category.id)
```

By clicking the *Data Analysis* icon on the *SQL Editor / Results* page, the *PivotCubeForm* is opened.

1. Cube Structure

The first page has three main areas:

- **All Fields**: This automatically displays all data set fields displayed on the SQL Editor's *Results* page.
- **Dimensions**: what is to be analyzed and displayed. The field order is at this stage irrelevant.
- **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, `ORDERDATE`, `TXT` and `NETAMOUNT`, the `ORDER_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 drop-down lists. Multiple field selection/deselection is also possible.

The NETAMOUNT field can be dragged 'n' dropped from the *All Fields* panel into the *Measures* area. Select *Calculation Type* from the options offered in the drop-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] and displayed on the *Cube Page.*

2. Cube

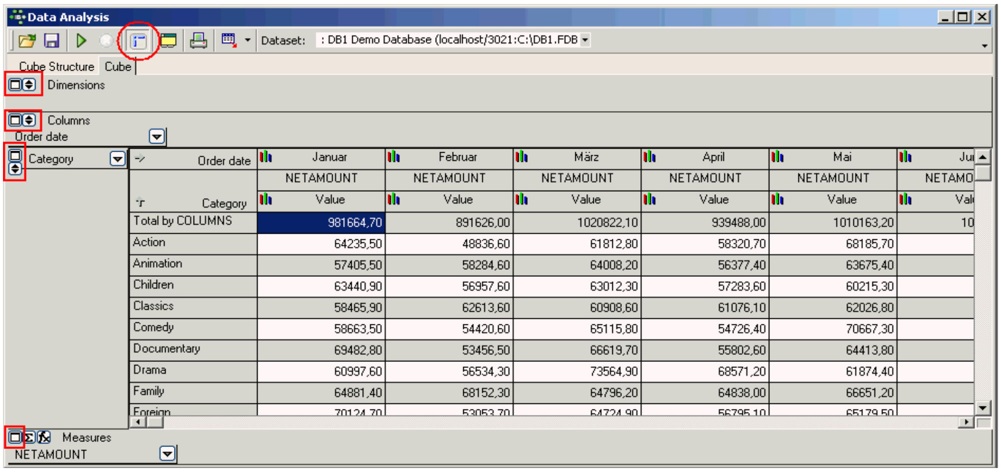
The second page in the *PivotCubeForm* displays the cube itself in the third of four areas, so-called toolbars:

- **Dimensions**
- **Columns**
- **Main display area**
- **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.

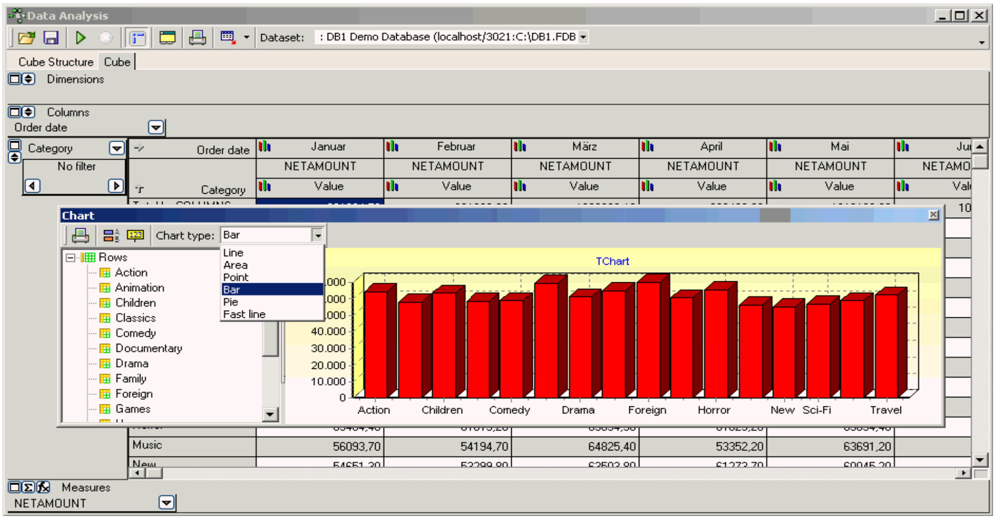
The toggle toolbars on/off icon (see circled icon below) can be used to remove these areas completely leaving just the main blue display area, or blending them in again.

It is now possible to generate a summary, for example, which film categories have generated which revenues.



Order date	Januar	Februar	März	April	Mai	Ju
	NETAMOUNT	NETAMOUNT	NETAMOUNT	NETAMOUNT	NETAMOUNT	NETAMO
Category	Value	Value	Value	Value	Value	Val
Total by COLUMNS	981664,70	891626,00	1020822,10	939488,00	1010163,20	10
Action	64235,50	48836,60	61812,80	58320,70	68185,70	
Animation	57405,50	58284,60	64008,20	56377,40	63675,40	
Children	63440,90	56957,60	63012,30	57283,60	60215,30	
Classics	58465,90	62613,60	60908,60	61076,10	62026,80	
Comedy	58663,50	54420,60	65115,80	54726,40	70667,30	
Documentary	69482,80	53456,50	66619,70	55802,60	64413,80	
Drama	60997,60	56534,30	73564,90	68571,20	61874,40	
Family	64881,40	68152,30	64796,20	64838,00	66651,20	
Foreign	70124,70	53053,70	64724,90	56795,10	65179,50	

The data can be displayed graphically by clicking on the graphics icon to the left of the *Measures* (here: *Order date* (monthly) or *Category*):

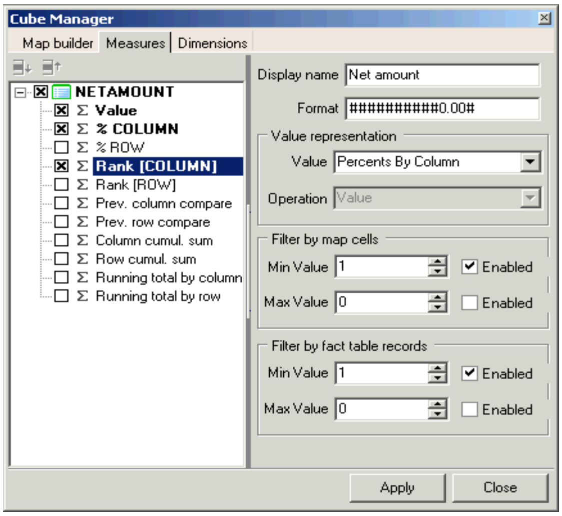


The *Graphics* window has its own mini toolbar, allowing the graph type to be altered, the legend and notes to be blended in or out, and the graph to be printed. There are numerous options to add functional values and formulae. Refer to *Data Analysis Cube Manager* and *Data Analysis Calculated Measures Manager* below for further information.

The generated data and analyses 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.

3. 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:


Cube Manager

Map builder | Measures | Dimensions

- [✓] NETAMOUNT
  - [✓] Σ Value
  - [✓] Σ % COLUMN
  - [ ] Σ % ROW
  - [✓] Σ Rank [COLUMN]
  - [ ] Σ Rank [ROW]
  - [ ] Σ Prev. column compare
  - [ ] Σ Prev. row compare
  - [ ] Σ Column cumul. sum
  - [ ] Σ Row cumul. sum
  - [ ] Σ Running total by column
  - [ ] Σ Running total by row

Display name: Net amount
Format: ###########0.00#

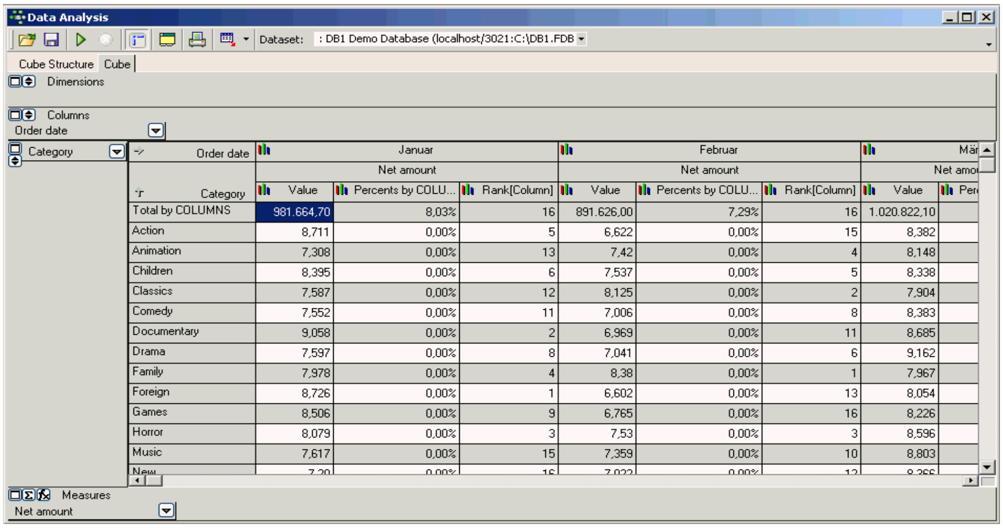
Value representation
Value: Percents By Column
Operation: Value

Filter by map cells
Min Value: 1 [✓] Enabled
Max Value: 0 [ ] Enabled

Filter by fact table records
Min Value: 1 [✓] Enabled
Max Value: 0 [ ] Enabled

Apply | Close

Click the *Apply* icon to view the results:



Data Analysis

Dataset: : DB1 Demo Database (localhost/3021:C:\DB1.FDB

Cube Structure | Cube

Dimensions

Columns: Order date

Rows: Category

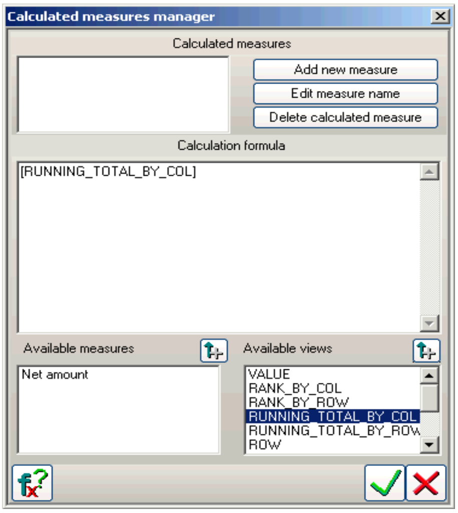
Measures: Net amount

Order date	Januar			Februar			Mär
	Net amount			Net amount			Net amount
Category	Value	Percents by COLU...	Rank[Column]	Value	Percents by COLU...	Rank[Column]	Value
Total by COLUMNS	981.664,70	8,03%	16	891.626,00	7,29%	16	1.020.822,10
Action	8,711	0,00%	5	6,622	0,00%	15	8,382
Animation	7,308	0,00%	13	7,42	0,00%	4	8,148
Children	8,395	0,00%	6	7,537	0,00%	5	8,338
Classics	7,587	0,00%	12	8,125	0,00%	2	7,904
Comedy	7,552	0,00%	11	7,006	0,00%	8	8,383
Documentary	9,058	0,00%	2	6,969	0,00%	11	8,685
Drama	7,597	0,00%	8	7,041	0,00%	6	9,162
Family	7,978	0,00%	4	8,38	0,00%	1	7,967
Foreign	8,726	0,00%	1	6,602	0,00%	13	8,054
Games	8,506	0,00%	9	6,765	0,00%	16	8,226
Horror	8,079	0,00%	3	7,53	0,00%	3	8,596
Music	7,617	0,00%	15	7,359	0,00%	10	8,803
New	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Depending on what you wish to see, it is possible to specify an ascending or descending order by simply clicking on the column headers.

4. 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*.



Calculated measures manager

Calculated measures

Calculation formula: [RUNNING_TOTAL_BY_COL]

Available measures: Net amount

Available views: VALUE, RANK_BY_COL, RANK_BY_ROW, RUNNING_TOTAL_BY_COL (selected), RUNNING_TOTAL_BY_ROW, ROW

New measures can be added and edited or existing measures deleted.

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 all specifications have been made satisfactorily, click the *Confirmation* (green tick) 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.

Chapter 19: Data Export & Import

IBExpert offer a number of methods for exporting and importing database objects and data. Please also refer to *Chapter 20: Data & Metadata Manipulation* for transferring metadata and data between Firebird and InterBase databases.

1. Export

Data can be exported from the *Data* page in the *Table Editor* and *View Editor*, the *Results* page in the *SQL Editor* and from the *ODBC Viewer*. The example below illustrates the export of data from the `db1 CUSTOMER` table into Excel format:

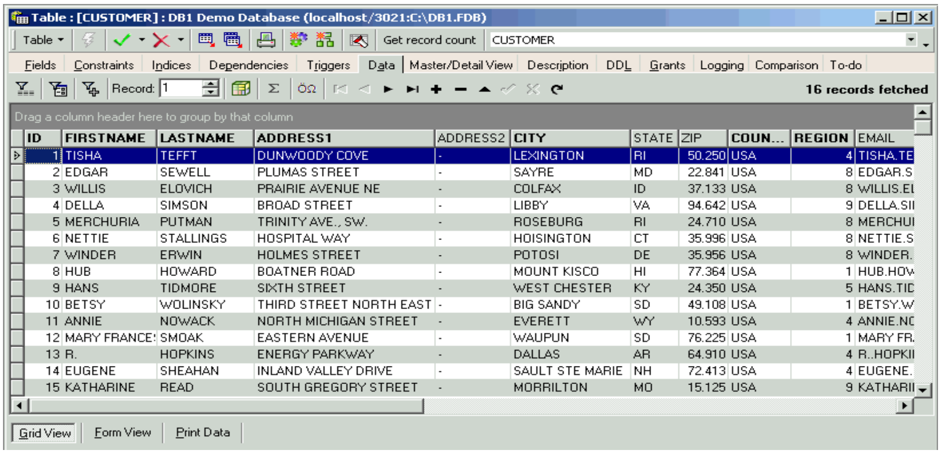


Table : [CUSTOMER] : DB1 Demo Database (localhost/3021:C:\DB1.FDB)

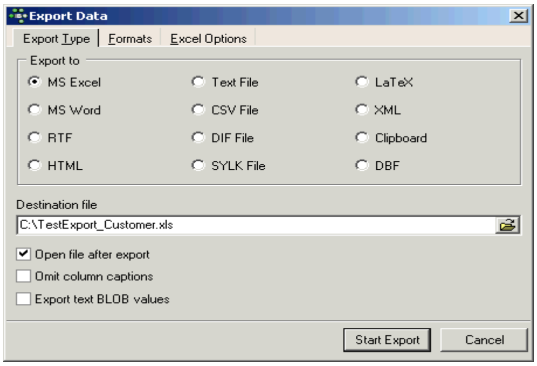
Record: 1 — 16 records fetched

ID	FIRSTNAME	LASTNAME	ADDRESS1	ADDRESS2	CITY	STATE	ZIP	COUN...	REGION	EMAIL
1	TISHA	TEFFT	DUNWOODY COVE	-	LEXINGTON	RI	50.250	USA	4	TISHA.TE
2	EDGAR	SEWELL	PLUMAS STREET	-	SAYRE	MD	22.841	USA	8	EDGAR.S
3	WILLIS	ELOVICH	PRAIRIE AVENUE NE	-	COLFAX	ID	37.133	USA	8	WILLIS.EL
4	DELLA	SIMSON	BROAD STREET	-	LIBBY	VA	94.642	USA	9	DELLA.SII
5	MERCHURIA	PUTMAN	TRINITY AVE., SW.	-	ROSEBURG	RI	24.710	USA	8	MERCHUI
6	NETTIE	STALLINGS	HOSPITAL WAY	-	HOISINGTON	CT	35.996	USA	8	NETTIE.S
7	WINDER	ERWIN	HOLMES STREET	-	POTOSI	DE	35.956	USA	8	WINDER.
8	HUB	HOWARD	BOATNER ROAD	-	MOUNT KISCO	HI	77.364	USA	1	HUB.HOV
9	HANS	TIDMORE	SIXTH STREET	-	WEST CHESTER	KY	24.350	USA	5	HANS.TIC
10	BETSY	WOLINSKY	THIRD STREET NORTH EAST	-	BIG SANDY	SD	49.108	USA	1	BETSY.W
11	ANNIE	NOWACK	NORTH MICHIGAN STREET	-	EVERETT	WY	10.593	USA	4	ANNIE.NC
12	MARY FRANCE!	SMOAK	EASTERN AVENUE	-	WAUPUN	SD	76.225	USA	1	MARY FR.
13	R.	HOPKINS	ENERGY PARKWAY	-	DALLAS	AR	64.910	USA	4	R..HOPKII
14	EUGENE	SHEAHAN	INLAND VALLEY DRIVE	-	SAULT STE MARIE	NH	72.413	USA	4	EUGENE.
15	KATHARINE	READ	SOUTH GREGORY STREET	-	MORRILTON	MO	15.125	USA	9	KATHARII

Double-click on the `CUSTOMER` table in the Database Explorer to open the *Table Editor / Data* page. The same data can also be exported from the *SQL Editor / Results* page by typing the SQL

```
select * from customer
```

Start the data export by clicking the *Export Data* icon or using the key combination [Ctrl + E] to open the *Data Export* window.


Export Data

Export to: [✓ MS Excel] MS Word, RTF, HTML, Text File, CSV File, DIF File, SYLK File, LaTeX, XML, Clipboard, DBF

Destination file: C:\TestExport_Customer.xls

[✓] Open file after export
[ ] Omit column captions
[ ] Export text BLOB values

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).

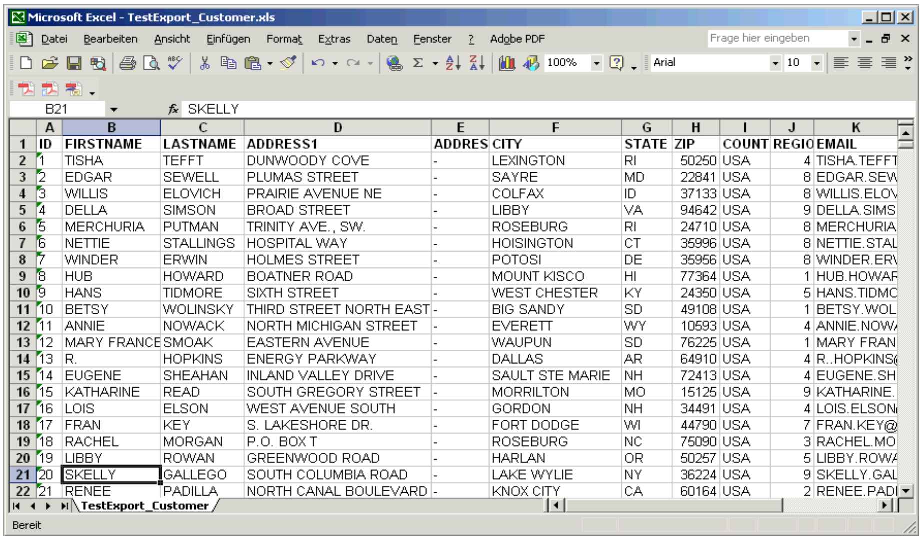
The destination file name must also be specified, and check options allow you to define whether the resulting export file should be opened following the data export or not, and - for certain export formats - whether column headings should be omitted or not, and whether text blob values should also be 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 under the IBExpert menu item *Options / Environment Options*.

Depending on the format, further options can be specified on the second and third pages, *Formats* and *Options*, specific to the export type. Here it is possible to specify a range of numerical formats, including currency, float, integer, date, time or date and time, as well as the decimal separator. 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. Open your data file in its new format, and you will see how easy it is to export data!



B21 SKELLY

	A	B	C	D	E	F	G	H	I	J	K
1	ID	FIRSTNAME	LASTNAME	ADDRESS1	ADDRES	CITY	STATE	ZIP	COUNT	REGIO	EMAIL
2	1	TISHA	TEFFT	DUNWOODY COVE	-	LEXINGTON	RI	50250	USA	4	TISHA.TEFFT
3	2	EDGAR	SEWELL	PLUMAS STREET	-	SAYRE	MD	22841	USA	8	EDGAR.SEW
4	3	WILLIS	ELOVICH	PRAIRIE AVENUE NE	-	COLFAX	ID	37133	USA	8	WILLIS.ELOV
5	4	DELLA	SIMSON	BROAD STREET	-	LIBBY	VA	94642	USA	9	DELLA.SIMS
6	5	MERCHURIA	PUTMAN	TRINITY AVE., SW.	-	ROSEBURG	RI	24710	USA	8	MERCHURIA
7	6	NETTIE	STALLINGS	HOSPITAL WAY	-	HOISINGTON	CT	35996	USA	8	NETTIE.STAL
8	7	WINDER	ERWIN	HOLMES STREET	-	POTOSI	DE	35956	USA	8	WINDER.ERV
9	8	HUB	HOWARD	BOATNER ROAD	-	MOUNT KISCO	HI	77364	USA	1	HUB.HOWAR
10	9	HANS	TIDMORE	SIXTH STREET	-	WEST CHESTER	KY	24350	USA	5	HANS.TIDMC
11	10	BETSY	WOLINSKY	THIRD STREET NORTH EAST	-	BIG SANDY	SD	49108	USA	1	BETSY.WOL
12	11	ANNIE	NOWACK	NORTH MICHIGAN STREET	-	EVERETT	WY	10593	USA	4	ANNIE.NOWA
13	12	MARY FRANCE	SMOAK	EASTERN AVENUE	-	WAUPUN	SD	76225	USA	1	MARY FRAN
14	13	R.	HOPKINS	ENERGY PARKWAY	-	DALLAS	AR	64910	USA	4	R..HOPKINS(
15	14	EUGENE	SHEAHAN	INLAND VALLEY DRIVE	-	SAULT STE MARIE	NH	72413	USA	4	EUGENE.SH
16	15	KATHARINE	READ	SOUTH GREGORY STREET	-	MORRILTON	MO	15125	USA	9	KATHARINE.
17	16	LOIS	ELSON	WEST AVENUE SOUTH	-	GORDON	NH	34491	USA	4	LOIS.ELSON
18	17	FRAN	KEY	S. LAKESHORE DR.	-	FORT DODGE	WI	44790	USA	7	FRAN.KEY@
19	18	RACHEL	MORGAN	P.O. BOX T	-	ROSEBURG	NC	75090	USA	3	RACHEL.MO
20	19	LIBBY	ROWAN	GREENWOOD ROAD	-	HARLAN	OR	50257	USA	5	LIBBY.ROWA
21	20	SKELLY	GALLEGO	SOUTH COLUMBIA ROAD	-	LAKE WYLIE	NY	36224	USA	9	SKELLY.GAL
22	21	RENEE	PADILLA	NORTH CANAL BOULEVARD	-	KNOX CITY	CA	60164	USA	2	RENEE.PADI

1.1 Export data into script

Data can also be exported into a script. The *Export Data into Script* dialog can be started using the respective icon on the *Data* page in the *Table Editor* and *View Editor*, the *Results* page in the SQL Editor or from the *ODBC Viewer*.

The following options may be selected before starting the export:

- Export into: *File*, *Clipboard* or *Script Executive*.
- Export as: `INSERT` statements, `UPDATE` statements, `UPDATE OR INSERT` statements or as a set of `EXECUTE PROCEDURE` statements.

2. Import

IBExpert offers two methods for importing data: the IBExpert *ODBC Viewer*, found in the *Tools* menu, or using IBExpert's `INSERTEX` function.

2.1 ODBC Viewer

The *ODBC Viewer* allows you to browse data from any ODBC source available on your PC and also import data from an ODBC source into an SQL script or directly into a Firebird/InterBase database.

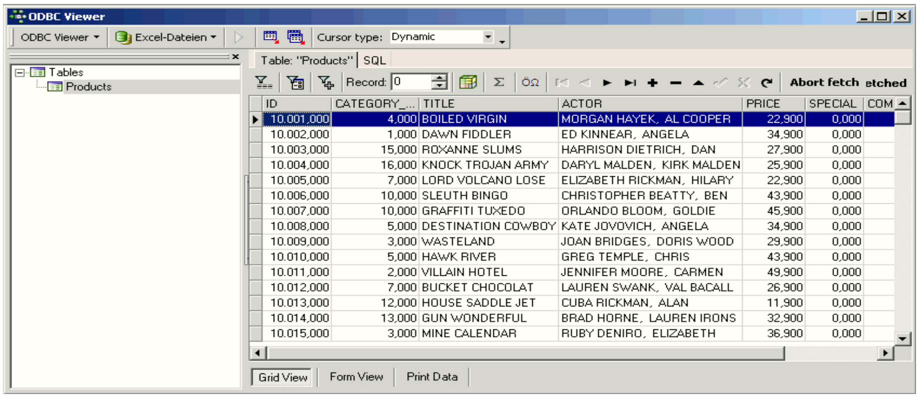
Setting up and testing the ODBC driver

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. You can find the correct connection string for the ODBC driver you are using here: http://www.connectionstrings.com/. Data from other data sources can be imported using the ODBC Viewer.

Importing data using the ODBC Viewer

Simply select the database from the selection of formats: dBASE or Excel files, or Microsoft Access databases, to load the database tables.

Double-click on a table in 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 *Print*.



Cursor type: Dynamic

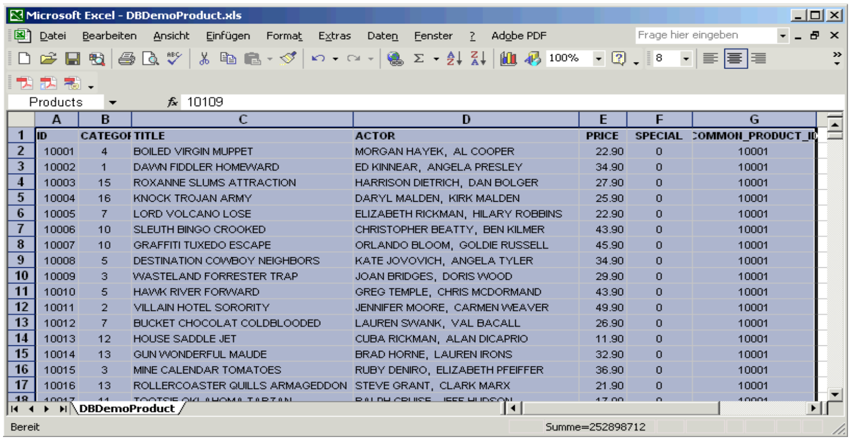
ID	CATEGORY_...	TITLE	ACTOR	PRICE	SPECIAL	COM
10.001,000	4,000	BOILED VIRGIN	MORGAN HAYEK, AL COOPER	22,900	0,000	
10.002,000	1,000	DAWN FIDDLER	ED KINNEAR, ANGELA	34,900	0,000	
10.003,000	15,000	ROXANNE SLUMS	HARRISON DIETRICH, DAN	27,900	0,000	
10.004,000	16,000	KNOCK TROJAN ARMY	DARYL MALDEN, KIRK MALDEN	25,900	0,000	
10.005,000	7,000	LORD VOLCANO LOSE	ELIZABETH RICKMAN, HILARY	22,900	0,000	
10.006,000	10,000	SLEUTH BINGO	CHRISTOPHER BEATTY, BEN	43,900	0,000	
10.007,000	10,000	GRAFFITI TUXEDO	ORLANDO BLOOM, GOLDIE	45,900	0,000	
10.008,000	5,000	DESTINATION COWBOY	KATE JOVOVICH, ANGELA	34,900	0,000	
10.009,000	3,000	WASTELAND	JOAN BRIDGES, DORIS WOOD	29,900	0,000	
10.010,000	5,000	HAWK RIVER	GREG TEMPLE, CHRIS	43,900	0,000	
10.011,000	2,000	VILLAIN HOTEL	JENNIFER MOORE, CARMEN	49,900	0,000	
10.012,000	7,000	BUCKET CHOCOLAT	LAUREN SWANK, VAL BACALL	26,900	0,000	
10.013,000	12,000	HOUSE SADDLE JET	CUBA RICKMAN, ALAN	11,900	0,000	
10.014,000	13,000	GUN WONDERFUL	BRAD HORNE, LAUREN IRONS	32,900	0,000	
10.015,000	3,000	MINE CALENDAR	RUBY DENIRO, ELIZABETH	36,900	0,000	

You can even query the table contents and view, print or export the results.

The *Export data* and *Export data into script* features described above are also available here, along with the additional option to *Export data into table*.

Importing Excel files and spread sheets

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 (our example has been defined in Excel as *Products*):


Microsoft Excel - DBDemoProduct.xls
Datei Bearbeiten Ansicht Einfügen Format Extras Daten Fenster ? Adobe PDF
Frage hier eingeben
100%
8
Products
10109
DBDemoProduct
Bereit
Summe=252898712

	A	B	C	D	E	F	G
1	ID	CATEGOI	TITLE	ACTOR	PRICE	SPECIAL	:OMMON_PRODUCT_ID
2	10001	4	BOILED VIRGIN MUPPET	MORGAN HAYEK, AL COOPER	22.90	0	10001
3	10002	1	DAWN FIDDLER HOMEWARD	ED KINNEAR, ANGELA PRESLEY	34.90	0	10001
4	10003	15	ROXANNE SLUMS ATTRACTION	HARRISON DIETRICH, DAN BOLGER	27.90	0	10001
5	10004	16	KNOCK TROJAN ARMY	DARYL MALDEN, KIRK MALDEN	25.90	0	10001
6	10005	7	LORD VOLCANO LOSE	ELIZABETH RICKMAN, HILARY ROBBINS	22.90	0	10001
7	10006	10	SLEUTH BINGO CROOKED	CHRISTOPHER BEATTY, BEN KILMER	43.90	0	10001
8	10007	10	GRAFFITI TUXEDO ESCAPE	ORLANDO BLOOM, GOLDIE RUSSELL	45.90	0	10001
9	10008	5	DESTINATION COWBOY NEIGHBORS	KATE JOVOVICH, ANGELA TYLER	34.90	0	10001
10	10009	3	WASTELAND FORRESTER TRAP	JOAN BRIDGES, DORIS WOOD	29.90	0	10001
11	10010	5	HAWK RIVER FORWARD	GREG TEMPLE, CHRIS MCDORMAND	43.90	0	10001
12	10011	2	VILLAIN HOTEL SORORITY	JENNIFER MOORE, CARMEN WEAVER	49.90	0	10001
13	10012	7	BUCKET CHOCOLAT COLDBLOODED	LAUREN SWANK, VAL BACALL	26.90	0	10001
14	10013	12	HOUSE SADDLE JET	CUBA RICKMAN, ALAN DICAPRIO	11.90	0	10001
15	10014	13	GUN WONDERFUL MAUDE	BRAD HORNE, LAUREN IRONS	32.90	0	10001
16	10015	3	MINE CALENDAR TOMATOES	RUBY DENIRO, ELIZABETH PFEIFFER	36.90	0	10001
17	10016	13	ROLLERCOASTER QUILLS ARMAGEDDON	STEVE GRANT, CLARK MARX	21.90	0	10001
18	10017	11	TOOTSIE OKLAHOMA TARZAN	RALPH CRUISE, JEFF HUDSON	17.90	0	10001

This defined data can then be used as a table in the ODBC Viewer.

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; in our example, *DBDemoProduct*). The first line is used always used for the column names.

2.2 ODBC access with IBEBlock

The following describes a simple way to access data via ODBC using IBExpert's IBEBlock. Download *IBEBlockScriptSamples.zip* from http://www.ibexpert.com/download/other_files/. Copy the `Demo.mdb` and `ODBCAcc.ibeblock` files (found in the `Blocks/ODBC Access` directory) into a separate directory. Copy `ODBCAcc.ibeblock` (copy of the script below) into the *SQL Editor*.

You can find the correct connection string for the ODBC driver you are using here: http://www.connectionstrings.com/ . Then simply modify the path to `Demo.mdb`, and 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
```

More about the many possibilities and functions of IBEBlock can be foundin *Chapter 23 IBEBlock*.

2.3 INSERTEX

IBExpert offers a range of further import and export options, one of them being INSERTEX, a function for importing CSV files. INSERTEX is one of IBExpert's script language extensions, which offer the developer a number of additional language options. The INSERTEX command can be used in IBExpert's *Script Executive* or *SQL Editor* (both found in the *Tools* menu).

INSERTEX imports data from a CSV-file into a database table. It has the following 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.

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.

Further import and export options are offered by IBExpert's *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.

3. 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.

Chapter 20: Data & Metadata Manipulation

IBExpert offers a number of options for manipulating data and 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 Firebird/InterBase database. 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.

The current metadata definitions can be viewed on the DDL page in the individual IBExpert object editors.

1. Extract Metadata

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 array fields' data (as blob data into a `LOB` file). The resulting script can be used to create a new empty database.

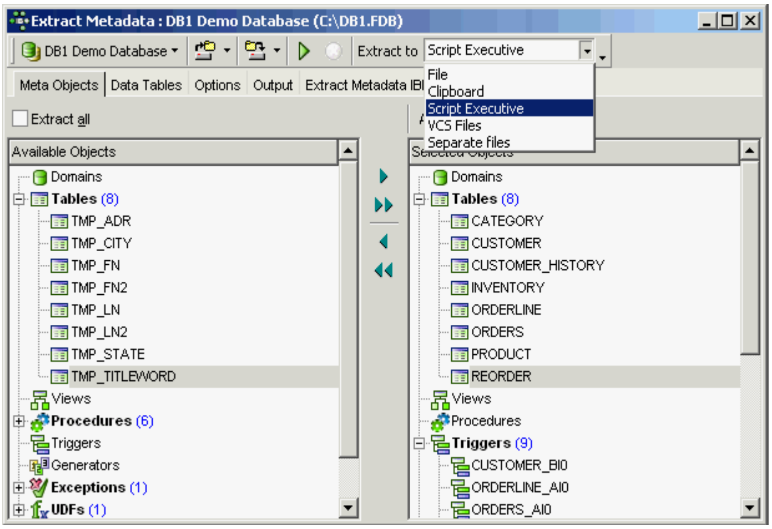
Table data can be extracted into separate files (`TABLE_1.sql`, `TABLE_2.sql`, `TABLE_3.sql` etc.) - the maximum file size can be specified on the *Options* page; once this size is reached, a new file is automatically generated by IBExpert, a particularly useful option when working with extremely large scripts, as problems are often encountered when executing scripts larger than 2 GB.

To begin preparing your metadata extraction, first select a database from the toolbar's drop-down list of all registered databases. The toolbar's *Extract to* options include: *File*, *Clipboard*, *Script Executive* (default), *VCS Files* and *Separate Files*.

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 selected, it is of course necessary to specify a file path and name (`*.sql` or *Metadata Extract Configuration* `*.mec`).

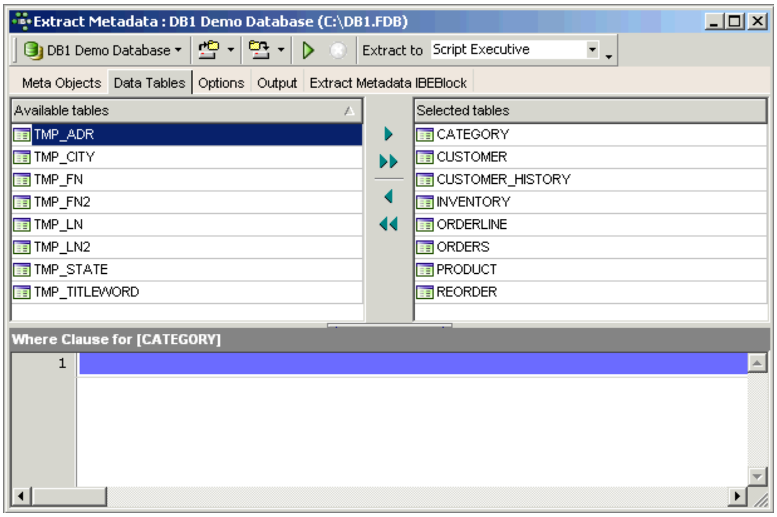
The first page, *Meta Objects*,displays all available objects for the selected database in the usual IBExpert tree form. The *Select Objects Tree* feature offers the user the choice whether to extract all database objects (check option), or specify individual objects, (using the **<** or **>** buttons, 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 simply clicking the respective button above the *Selected Objects* window.

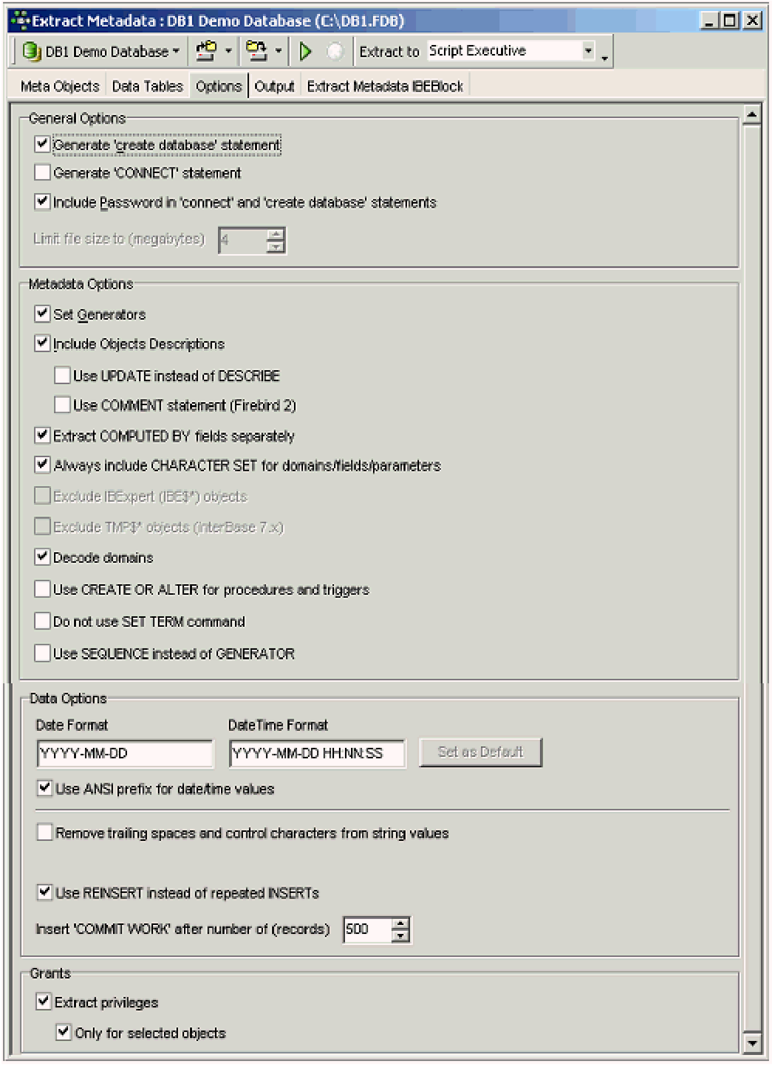
It is also possible to drag objects from the object dependencies trees (found on the *Dependencies* page in the object editors) and the field dependencies list (found in the *Field Dependencies* window at the bottom of the *Fields* page in the *Table/View editors*) into the *Selected Objects* tree.

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:



By selecting one of the tables in the *Selected Tables* list on the right-hand side, it is possible to add a `WHERE` clause, if wished.

The *Extract Metadata Options* page offers a wide range of further check options:



Extract Metadata : DB1 Demo Database (C:\DB1.FDB)

DB1 Demo Database · Extract to: Script Executive

Meta Objects | Data Tables | Options | Output | Extract Metadata IBEBlock

General Options
- [✓] Generate 'create database' statement
- [ ] Generate 'CONNECT' statement
- [✓] Include Password in 'connect' and 'create database' statements
- Limit file size to (megabytes): 4

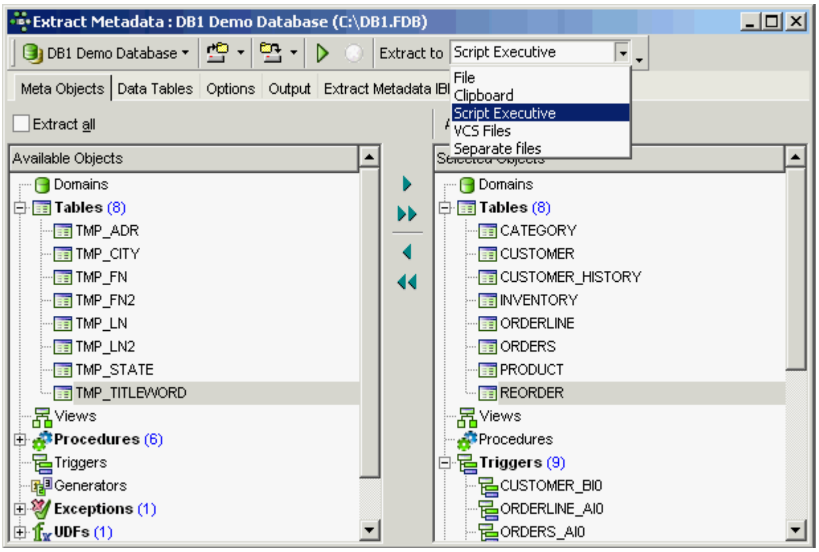
Metadata Options
- [✓] Set Generators
- [✓] Include Objects Descriptions
  - [ ] Use UPDATE instead of DESCRIBE
  - [ ] Use COMMENT statement (Firebird 2)
- [✓] Extract COMPUTED BY fields separately
- [✓] Always include CHARACTER SET for domains/fields/parameters
- [ ] Exclude IBExpert (IBE$*) objects
- [ ] Exclude TMP$* objects (InterBase 7.x)
- [✓] Decode domains
- [ ] Use CREATE OR ALTER for procedures and triggers
- [ ] Do not use SET TERM command
- [ ] Use SEQUENCE instead of GENERATOR

Data Options
- Date Format: YYYY-MM-DD
- DateTime Format: YYYY-MM-DD HH:NN:SS
- Set as Default
- [✓] Use ANSI prefix for date/time values
- [ ] Remove trailing spaces and control characters from string values
- [✓] Use REINSERT instead of repeated INSERTs
- Insert 'COMMIT WORK' after number of (records): 500

Grants
- [✓] Extract privileges
  - [✓] Only for selected objects

including *General Options* – adding the `CREATE DATABASE` or `CONNECT` statement, including password and limiting the file size; *Metadata Options*, *Data Options* and *Grants*.

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.



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 ORDERS table every evening.

Any number of configurations may be saved in various formats: *Metadata extract configuration* (*.mec) allows you to quickly and simply load a specified configuration in the *Extract Metadata* dialog. *IBEBlock* (*.ibeblock) 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. Alternatively select a file format of your choice.

Simply specify the directory and file name you wish to extract to, and then customize the *Extract Metadata block* on the *IBEBlock* page as required and file. This function offers a quick and simple solution for a number of otherwise cumbersome tasks, such as generating foreign language versions of your database, subsequent alteration of the character set, alternative backup and restore or incremental backups.

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. Then use the IBExpert *Tools* menu item, *Extract Metadata*. Connect to your database and select all tables for metadata and data. *Extract to* - select *separate files* from the drop-down list. Extract all objects and data from all tables.

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`. Then generate your script (green arrow icon or [F9], deleting the original database file.

If required, add the missing data as far as possible from an older extract file or backup copy of the database.

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.

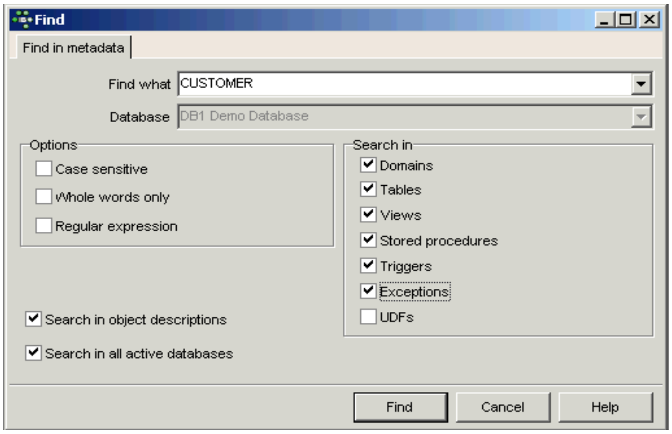
`IBE$Start` runs the operations such as creating the database and metadata. Tables are generated, without any primary keys, foreign keys, constraints, triggers, etc. This is followed by a series of insert commands, using the IBEBlock function, `REINSERT`. `IBE$Finish` then inserts all primary keys, foreign keys etc.

You can, of course carry all this out at script level, using `ibec_ExtractMetadata`.

This method can 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.

2. Search Metadata

This option is useful for finding individual words/digits or word/digit strings in metadata and in object descriptions. It even searches for and displays field names, as opposed to the Database Explorer filter, which only searches for object names. It can be found in the IBExpert *Tools* menu or using the *Edit* menu's *Find* option - *Find in Metadata* page. The *Find Metadata* dialog offers a number of options:



Find what: CUSTOMER
Database: DB1 Demo Database

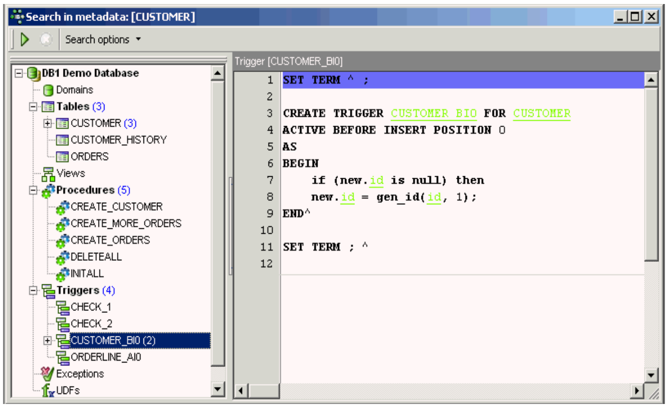
Options
- [ ] Case sensitive
- [ ] Whole words only
- [ ] Regular expression

Search in
- [✓] Domains
- [✓] Tables
- [✓] Views
- [✓] Stored procedures
- [✓] Triggers
- [✓] Exceptions
- [ ] UDFs

- [✓] Search in object descriptions
- [✓] Search in all active databases

Select previous search criteria listed in the drop-down list. A single active database may be selected from the second drop-down list; alternatively the *Search in all Active Databases* option in the bottom left-hand corner of the dialog can be activated.

Further *Search* options include *Case sensitive, Whole words only, Regular Expression* (recognizes regular expressions in the search string) and *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:



```
SET TERM ^ ;

CREATE TRIGGER CUSTOMER_BIO FOR CUSTOMER
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
    if (new.id is null) then
    new.id = gen_id(id, 1);
END^

SET TERM ; ^
```

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 summary 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.

3. SQL Editor 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.

3.1 Creating a table from query results

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);
```

3.2 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`.

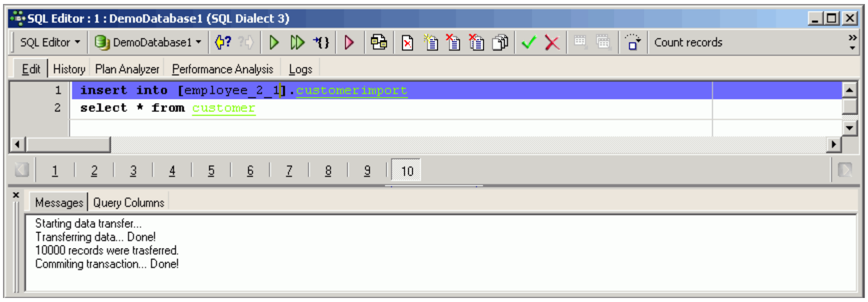
For example: to transfer data from the `db1` database to the Firebird/InterBase sample database, `EMPLOYEE.FDB`, open the SQL Editor for the `db1` database, and

```
select * from customer
```

To transfer this data to `EMPLOYEE.FDB` (with the IBExpert alias name, `EMPLOYEE_2_1`) write the following SQL when connected to the source (`db1`) database:

```
insert into [employee_2_1].customerimport
select * from customer
```

If the table doesn't already exist, IBExpert will ask if you wish to create it:



```
insert into [employee_2_1].customerimport
select * from customer
```

```
Starting data transfer...
Transferring data... Done!
10000 records were trasferred.
Commiting transaction... Done!
```

and 10,000 customer records have just been copied into another database without having to worry about first creating a new table of the same structure or performing some complicated export routine!

4. Copy Database Object

Copy Database Object is available as a 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 *New object name* field; this can of course be altered if wished.

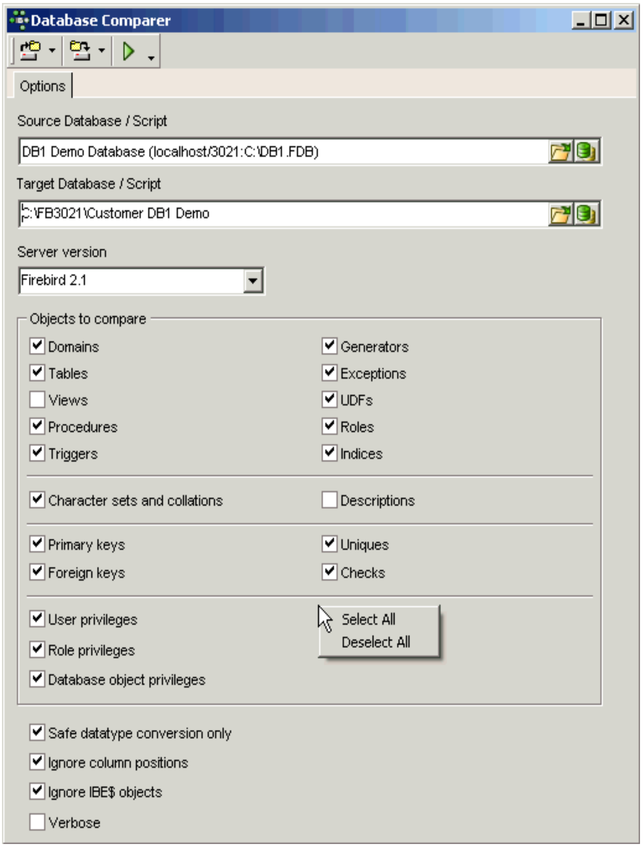
Depending upon the object selected, a number of check box *Copy* options are offered, including options for exactly which contents should be copied, and how IBExpert is to proceed should the object already exist.

Start the copy process by clicking the green arrow icon or using [F9]. After the *Output* script has been generated, the default IBEBlock is displayed on the *Blocks* page. 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.

5. Database Comparer & Table Data Comparer

The IBExpert *Database Comparer* allows developers to compare database versions or database SQL scripts. This is particularly useful, for example, before installing an updated client application



Source Database / Script: DB1 Demo Database (localhost/3021:C:\DB1.FDB)

Target Database / Script: C:\FB3021\Customer DB1 Demo

Server version: Firebird 2.1

Objects to compare

- [✓] Domains
- [✓] Tables
- [ ] Views
- [✓] Procedures
- [✓] Triggers
- [✓] Generators
- [✓] Exceptions
- [✓] UDFs
- [✓] Roles
- [✓] Indices
- [✓] Character sets and collations
- [ ] Descriptions
- [✓] Primary keys
- [✓] Foreign keys
- [✓] Uniques
- [✓] Checks
- [✓] User privileges
- [✓] Role privileges
- [✓] Database object privileges

- [✓] Safe datatype conversion only
- [✓] Ignore column positions
- [✓] Ignore IBE$ objects
- [ ] Verbose

which contains new tables, procedures, exceptions, 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. IBExpert has already implemented support of many known Firebird 2.5 features, including `TYPE OF COLUMN`, `IN AUTONOMOUS TRANSACTION` and extended syntax of `EXECUTE STATEMENT`.

The *Database Comparer* can be started from the IBExpert *Tools* menu. 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 Database Explorer into the respective fields in the *Database Comparer* dialog.

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 `IBEScript.exe` (IBExpert command-line tool). When settings are saved into an `INI` file, IBExpert also saves the server version.

Server version 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.

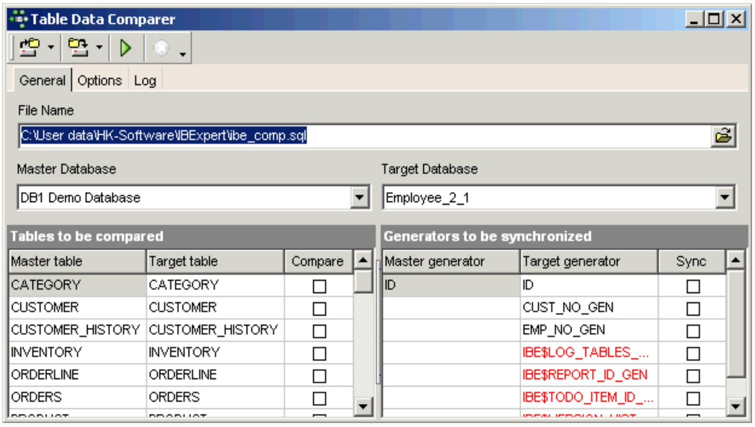
There are a number of options, which can be checked if they should be included in the comparison. All options can be selected or deselected simply and quickly using the right-click context-sensitive menu.

After selecting all features to be (or not to be) compared, click the *Compare* icon to start the comparison.

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. On the *Statements* page it is easy to unselect or select individual statements using point and click. 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 data type 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.

The *Table Data Comparer* is similar to the *Database Comparer*. 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.



File Name: C:\User data\HK-Software\IBExpert\ibe_comp.sql

Master Database: DB1 Demo Database

Target Database: Employee_2_1

Tables to be compared

Master table	Target table	Compare
CATEGORY	CATEGORY	☐
CUSTOMER	CUSTOMER	☐
CUSTOMER_HISTORY	CUSTOMER_HISTORY	☐
INVENTORY	INVENTORY	☐
ORDERLINE	ORDERLINE	☐
ORDERS	ORDERS	☐

Generators to be synchronized

Master generator	Target generator	Sync
ID	ID	☐
	CUST_NO_GEN	☐
	EMP_NO_GEN	☐
	IBE$LOG_TABLES_...	☐
	IBE$REPORT_ID_GEN	☐
	IBE$TODO_ITEM_ID_...	☐

The *General* page displays the default file path and name for the resulting comparison script. This can of course be altered as wished. As with the *Database Comparer*, first select the *Master* or *Reference Database* from the drop-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 required 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. Note that system tables are not selected when using this function.

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 relevant toolbar icons. Refine your specification on the *Options* page which offers a number of check box options.

To start the table comparison simply click the *Compare* button (green arrow) or [F9].

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.

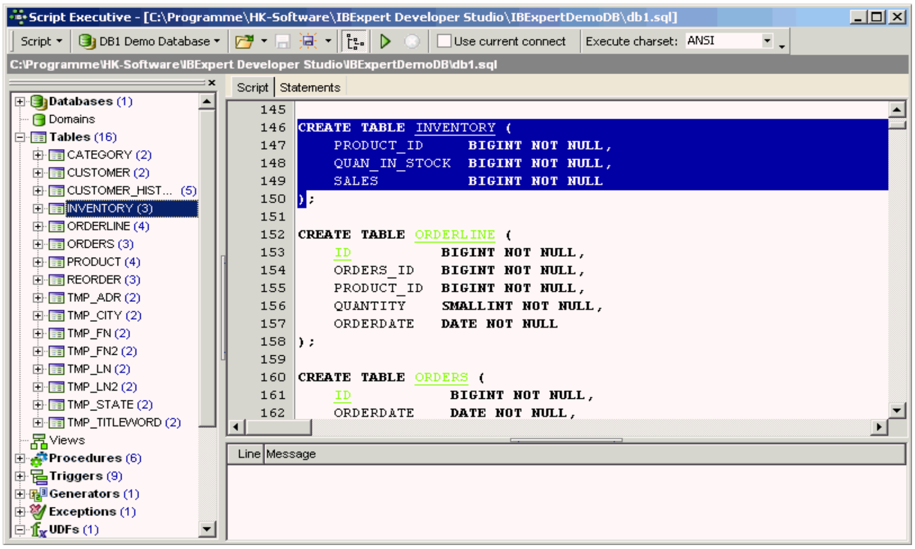
Chapter 21: Script Executive & Script Language Extensions

IBExpert's *Script Executive* has already been briefly presented in previous chapters. This versatile IBExpert feature can be used to view, edit and execute SQL scripts.

Although Firebird/InterBase can also process such procedure definitions in the *SQL Editor*, it is recommended using 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.

It can be started from the IBExpert *Tools* menu, using the respective icon in the *Tools* toolbar or [Ctrl + F12]. It is used for SQLs covering several rows.

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.



```
CREATE TABLE INVENTORY (
    PRODUCT_ID      BIGINT NOT NULL,
    QUAN_IN_STOCK   BIGINT NOT NULL,
    SALES           BIGINT NOT NULL
);

CREATE TABLE ORDERLINE (
    ID          BIGINT NOT NULL,
    ORDERS_ID   BIGINT NOT NULL,
    PRODUCT_ID  BIGINT NOT NULL,
    QUANTITY    SMALLINT NOT NULL,
    ORDERDATE   DATE NOT NULL
);

CREATE TABLE ORDERS (
    ID             BIGINT NOT NULL,
    ORDERDATE      DATE NOT NULL,
```

Please note that 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.

IBExpert offers full Unicode support. The internal representation of all texts in the code editors is Windows Unicode (UTF-16LE, two bytes per character). This allows you to use multilingual characters in your procedures, queries, database object descriptions etc., if you use the UTF8 character set when connecting to your database. When you're working with a database using the UTF8 character set IBExpert performs automatic conversion from UTF8 to Windows Unicode (when opening) and back (when you compiling). This applies to Firebird 2.1 and 2.5 databases. For other databases you need to enable this behavior manually (if you really need this!) by flagging the *Do NOT perform conversion from/to UTF8* check box in the *Database Registration Info*. As a rule, IBExpert knows when it must convert strings from Windows Unicode to UTF8 but sometimes it is necessary to specify the conversion type manually. This allows you to specify the necessary character set manually.

The *Script* page includes features such as code completion - 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 respective 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.

1. 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';
```

2. 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=XXXXXX||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 if customers are selected.

3. 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`.

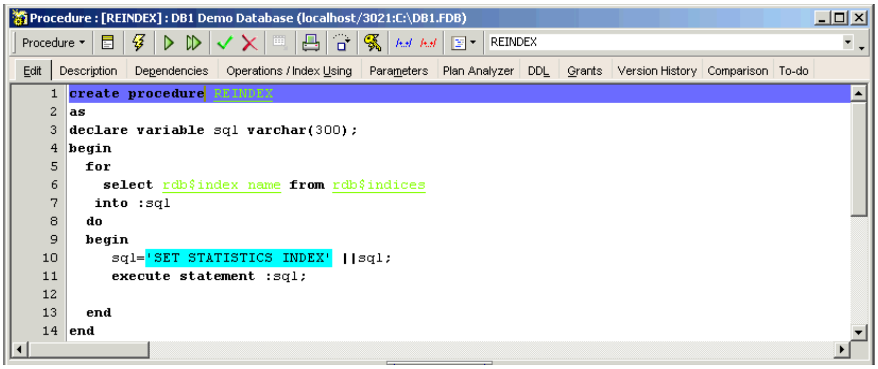
All these are fully documented with examples in the IBExpert online documentation at www.ibexpert.com/doc.

Chapter 22: Firebird 2.0 Blocks

New to Firebird 2.0, Firebird's block implementation enables complex SQL operations in many application areas.

A block is a simple feature, using the new `EXECUTE BLOCK` syntax, which executes a block of PSQL code as if it were a stored procedure, optionally with input and output parameters and variable declarations. This allows the user to perform "on the fly" PSQL within a DSQL context. It performs a block of instructions on the server side, and can in fact be considered a virtual stored procedure.

To illustrate this, let's consider the following situation: you have a procedure, but you don't really want or need to store it in your database.:



```
create procedure REINDEX
as
declare variable sql varchar(300);
begin
  for
    select rdb$index_name from rdb$indices
   into :sql
  do
  begin
     sql='SET STATISTICS INDEX' ||sql;
     execute statement :sql;

  end
end
```

You just want to create such a procedure on the fly and drop it afterward. So make the following simple alterations:

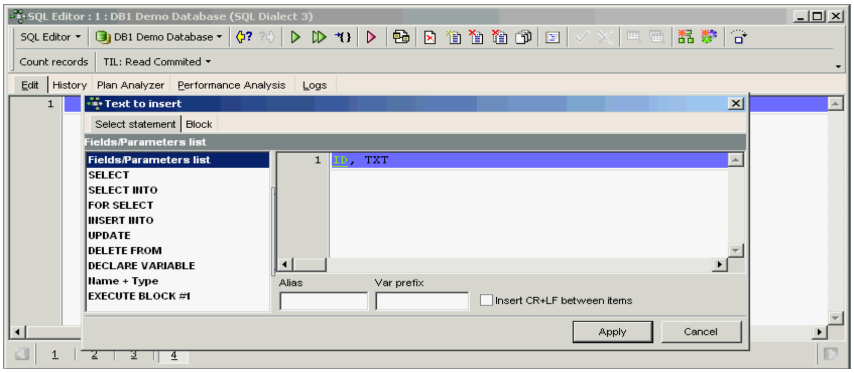
```
execute block
as
declare variable sql varchar(300;
begin
  for
    select rdb$index_name from rdb$indices
    into :sql
 do
  begjn
    sql='SET STATISTICS INDEX' ||sql;
   execute statement :sql;
  end
end
```

and it performs the same task, but as a dynamic block and not as a stored procedure.

The block transfers the source code from the client to the server, and executes it at the same speed as a stored procedure. The block is created and prepared when you start it, and deleted when you commit or roll back. The server will never use it again.

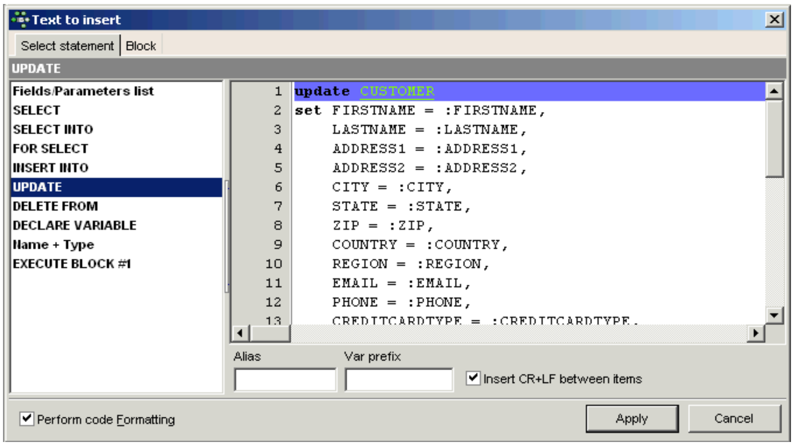
The major advantage of a block is when you are creating a variety of different but similar procedures from your client application, for example you have stored procedures for customer searching; in one stored procedure you are doing the customer search for the sales department, and in the other stored procedure you are doing the customer search for the invoice department. They have slightly different search criteria and want to see different columns in the result sets – this could be an interesting task, as the number of columns can be directly and dynamically created in a block.

`EXECUTE BLOCK` is not only a alternative to stored procedures; there are other uses, particularly for performance tasks. To illustrate this, for example, take a table `TEST1`, drag it from the *Database Explorer* into the *SQL Editor*. The *Text to insert* window opens offering a range of options:



```
ID, TXT
```

To prepare a `SELECT INTO` with carriage return and line feed, simply click on the `SELECT INTO` from the list on the left and check the Insert `CR+LF` between items. IBExpert then inserts the correct syntax. In the case of this small table `TEST1`, this might not appear to be such an advantage, but if you take a look at a table with a larger number of fields (e.g. the `DB1 CUSTOMER` table), you will see how much it helps to have the field names and parameters already inserted into the standard syntax:

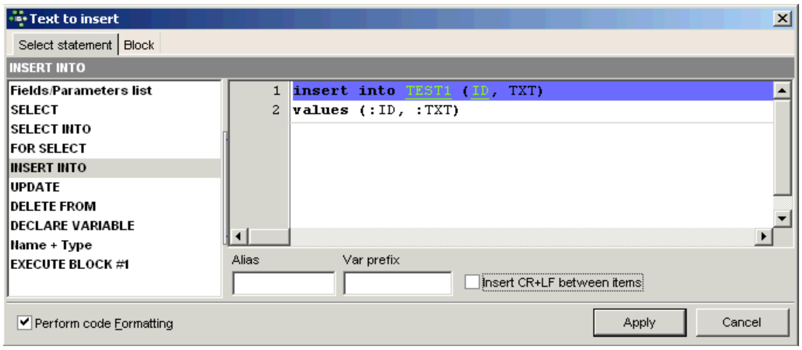


```
update CUSTOMER
set FIRSTNAME = :FIRSTNAME,
    LASTNAME = :LASTNAME,
    ADDRESS1 = :ADDRESS1,
    ADDRESS2 = :ADDRESS2,
    CITY = :CITY,
    STATE = :STATE,
    ZIP = :ZIP,
    COUNTRY = :COUNTRY,
    REGION = :REGION,
    EMAIL = :EMAIL,
    PHONE = :PHONE,
    CREDITCARDTYPE = :CREDITCARDTYPE,
```

To ascertain the data type definitions or to declare variables, simply click on the *Name + Type* in the left-hand list. Variable prefixes can be inserted (for example: v_) in the field *Var prefix* below, to offer you an instant full list of variables for all fields in the table.

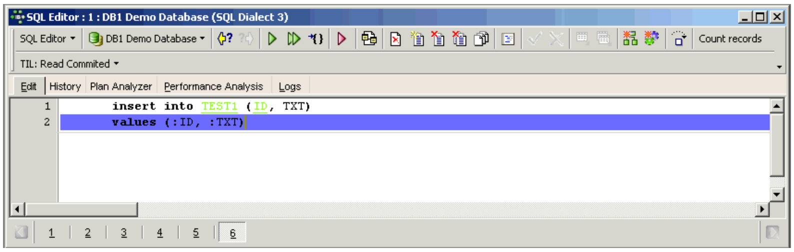
Firebird 2.1 also introduced the possibility to use domains for procedures, procedure parameters and so on. (Please refer to Chapter 12: 14 *Using domains in stored procedures* for details).

To continue with the implementation of the `TEST1` table: an `INSERT INTO` statement is specified without carriage return and line feed or a variable prefix:



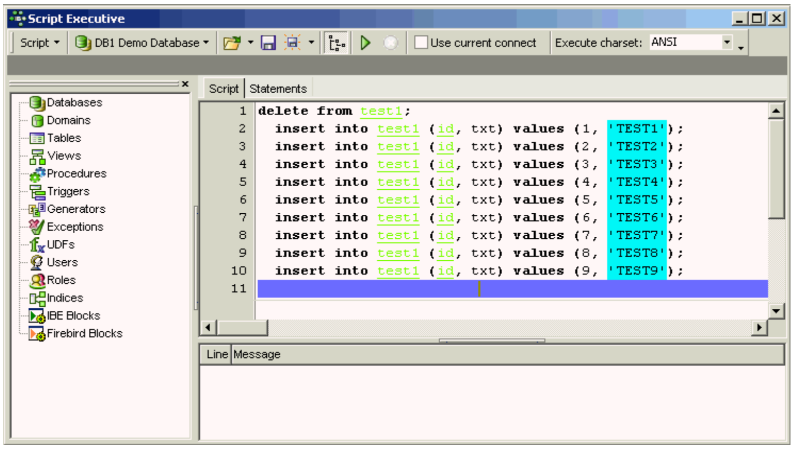
```
insert into TEST1 (ID, TXT)
values (:ID, :TXT)
```

When it's ready simply apply and you the `INSERT INTO` command is already formulated in the *SQL Editor* or *Script Executive*:



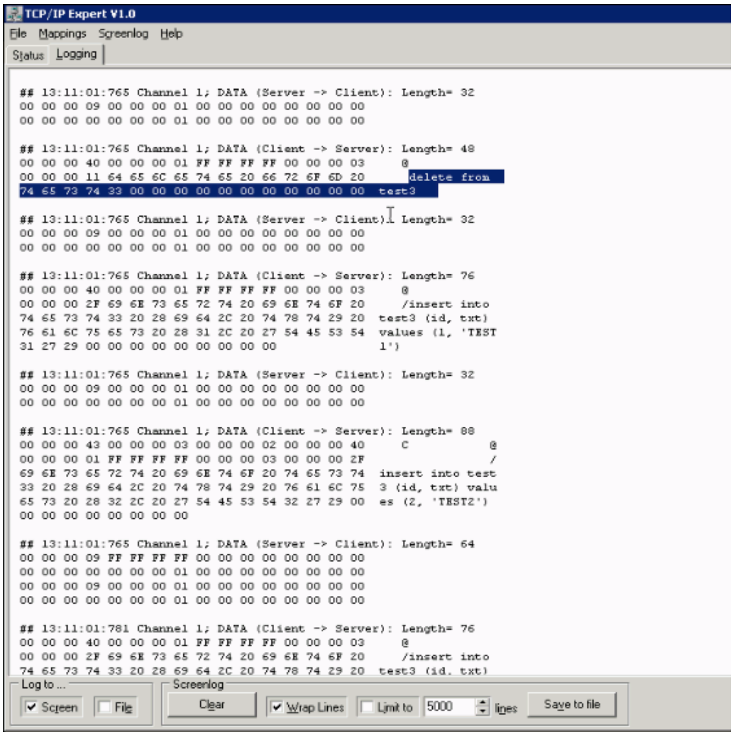
```
        insert into TEST1 (ID, TXT)
        values (:ID, :TXT)
```

Now to illustrate one of the main advantages of Firebird blocks, some operations are added, one by one:



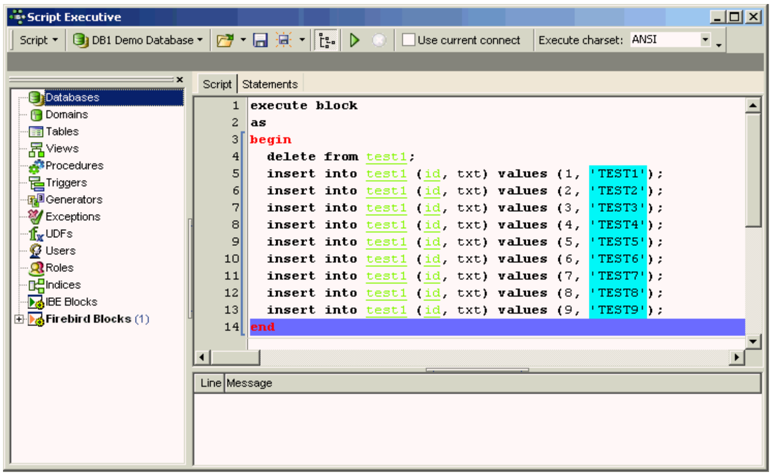
```
delete from test1;
  insert into test1 (id, txt) values (1, 'TEST1');
  insert into test1 (id, txt) values (2, 'TEST2');
  insert into test1 (id, txt) values (3, 'TEST3');
  insert into test1 (id, txt) values (4, 'TEST4');
  insert into test1 (id, txt) values (5, 'TEST5');
  insert into test1 (id, txt) values (6, 'TEST6');
  insert into test1 (id, txt) values (7, 'TEST7');
  insert into test1 (id, txt) values (8, 'TEST8');
  insert into test1 (id, txt) values (9, 'TEST9');
```

When the above script is executed in the IBExpert *Script Executive*, and the *IBExpertNetwork-Monitor* is running in the background, the traffic can be viewed immediately:



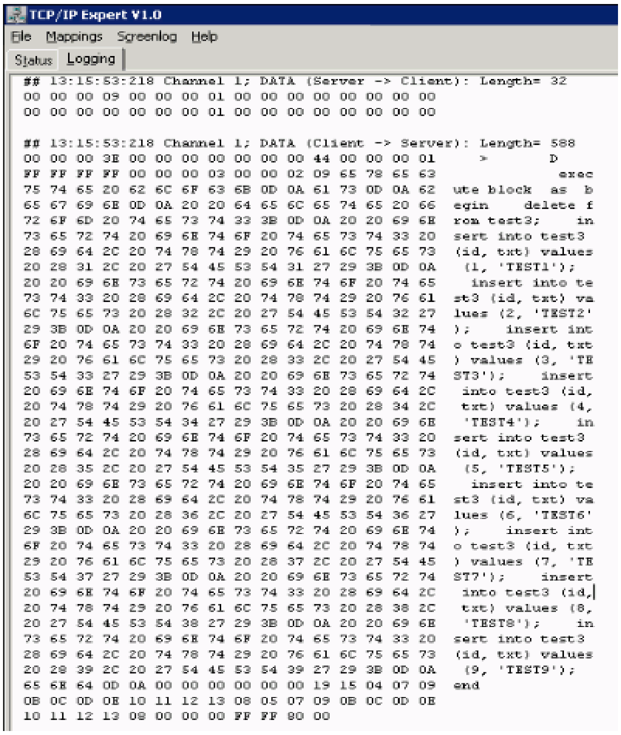
```
## 13:11:01:765 Channel 1; DATA (Server -> Client): Length= 32
00 00 00 09 00 00 00 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00

## 13:11:01:765 Channel 1; DATA (Client -> Server): Length= 48
00 00 00 40 00 00 00 01 FF FF FF FF 00 00 00 03      @
00 00 00 11 64 65 6C 65 74 65 20 66 72 6F 6D 20      delete from
74 65 73 74 33 00 00 00 00 00 00 00 00 00 00 00  test3

## 13:11:01:765 Channel 1; DATA (Server -> Client): Length= 32
00 00 00 09 00 00 00 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00

## 13:11:01:765 Channel 1; DATA (Client -> Server): Length= 76
00 00 00 40 00 00 00 01 FF FF FF FF 00 00 00 03      @
00 00 00 2F 69 6E 73 65 72 74 20 69 6E 74 6F 20     /insert into
74 65 73 74 33 20 28 69 64 2C 20 74 78 74 29 20  test3 (id, txt)
76 61 6C 75 65 73 20 28 31 2C 20 27 54 45 53 54  values (1, 'TEST
31 27 29 00 00 00 00 00 00 00 00 00              1')

## 13:11:01:765 Channel 1; DATA (Server -> Client): Length= 32
00 00 00 09 00 00 00 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00

## 13:11:01:765 Channel 1; DATA (Client -> Server): Length= 88
00 00 00 43 00 00 00 03 00 00 00 02 00 00 00 40      C           @
00 00 00 01 FF FF FF FF 00 00 00 03 00 00 00 2F                  /
69 6E 73 65 72 74 20 69 6E 74 6F 20 74 65 73 74  insert into test
33 20 28 69 64 2C 20 74 78 74 29 20 76 61 6C 75  3 (id, txt) valu
65 73 20 28 32 2C 20 27 54 45 53 54 32 27 29 00  es (2, 'TEST2')
00 00 00 00 00 00 00 00

## 13:11:01:765 Channel 1; DATA (Server -> Client): Length= 64
00 00 00 09 FF FF FF FF 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00
00 00 00 09 00 00 00 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00

## 13:11:01:781 Channel 1; DATA (Client -> Server): Length= 76
00 00 00 40 00 00 00 01 FF FF FF FF 00 00 00 03      @
00 00 00 2F 69 6E 73 65 72 74 20 69 6E 74 6F 20     /insert into
74 65 73 74 33 20 28 69 64 2C 20 74 78 74 29 20  test3 (id, txt)
```

Add the beginning and closing clauses, to turn these statements into a block:



```
execute block
as
begin
  delete from test1;
  insert into test1 (id, txt) values (1, 'TEST1');
  insert into test1 (id, txt) values (2, 'TEST2');
  insert into test1 (id, txt) values (3, 'TEST3');
  insert into test1 (id, txt) values (4, 'TEST4');
  insert into test1 (id, txt) values (5, 'TEST5');
  insert into test1 (id, txt) values (6, 'TEST6');
  insert into test1 (id, txt) values (7, 'TEST7');
  insert into test1 (id, txt) values (8, 'TEST8');
  insert into test1 (id, txt) values (9, 'TEST9');
end
```

The Firebird server now processes all operations in one go, and you can see that all operations have been sent as one package to the server.



```
## 13:15:53:218 Channel 1; DATA (Server -> Client): Length= 32
00 00 00 09 00 00 00 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00

## 13:15:53:218 Channel 1; DATA (Client -> Server): Length= 588
00 00 00 3E 00 00 00 00 00 00 00 44 00 00 00 01     >       D
FF FF FF FF 00 00 00 03 00 00 02 09 65 78 65 63               exec
75 74 65 20 62 6C 6F 63 6B 0D 0A 61 73 0D 0A 62  ute block  as  b
65 67 69 6E 0D 0A 20 20 64 65 6C 65 74 65 20 66  egin    delete f
72 6F 6D 20 74 65 73 74 33 3B 0D 0A 20 20 69 6E  rom test3;    in
73 65 72 74 20 69 6E 74 6F 20 74 65 73 74 33 20  sert into test3
28 69 64 2C 20 74 78 74 29 20 76 61 6C 75 65 73  (id, txt) values
20 28 31 2C 20 27 54 45 53 54 31 27 29 3B 0D 0A   (1, 'TEST1');
20 20 69 6E 73 65 72 74 20 69 6E 74 6F 20 74 65    insert into te
73 74 33 20 28 69 64 2C 20 74 78 74 29 20 76 61  st3 (id, txt) va
6C 75 65 73 20 28 32 2C 20 27 54 45 53 54 32 27  lues (2, 'TEST2'
29 3B 0D 0A 20 20 69 6E 73 65 72 74 20 69 6E 74  );    insert int
6F 20 74 65 73 74 33 20 28 69 64 2C 20 74 78 74  o test3 (id, txt
29 20 76 61 6C 75 65 73 20 28 33 2C 20 27 54 45  ) values (3, 'TE
53 54 33 27 29 3B 0D 0A 20 20 69 6E 73 65 72 74  ST3');    insert
20 69 6E 74 6F 20 74 65 73 74 33 20 28 69 64 2C   into test3 (id,
20 74 78 74 29 20 76 61 6C 75 65 73 20 28 34 2C   txt) values (4,
20 27 54 45 53 54 34 27 29 3B 0D 0A 20 20 69 6E   'TEST4');    in
73 65 72 74 20 69 6E 74 6F 20 74 65 73 74 33 20  sert into test3
28 69 64 2C 20 74 78 74 29 20 76 61 6C 75 65 73  (id, txt) values
20 28 35 2C 20 27 54 45 53 54 35 27 29 3B 0D 0A   (5, 'TEST5');
20 20 69 6E 73 65 72 74 20 69 6E 74 6F 20 74 65    insert into te
73 74 33 20 28 69 64 2C 20 74 78 74 29 20 76 61  st3 (id, txt) va
6C 75 65 73 20 28 36 2C 20 27 54 45 53 54 36 27  lues (6, 'TEST6'
29 3B 0D 0A 20 20 69 6E 73 65 72 74 20 69 6E 74  );    insert int
6F 20 74 65 73 74 33 20 28 69 64 2C 20 74 78 74  o test3 (id, txt
29 20 76 61 6C 75 65 73 20 28 37 2C 20 27 54 45  ) values (7, 'TE
53 54 37 27 29 3B 0D 0A 20 20 69 6E 73 65 72 74  ST7');    insert
20 69 6E 74 6F 20 74 65 73 74 33 20 28 69 64 2C   into test3 (id,
20 74 78 74 29 20 76 61 6C 75 65 73 20 28 38 2C   txt) values (8,
20 27 54 45 53 54 38 27 29 3B 0D 0A 20 20 69 6E   'TEST8');    in
73 65 72 74 20 69 6E 74 6F 20 74 65 73 74 33 20  sert into test3
28 69 64 2C 20 74 78 74 29 20 76 61 6C 75 65 73  (id, txt) values
20 28 39 2C 20 27 54 45 53 54 39 27 29 3B 0D 0A   (9, 'TEST9');
65 6E 64 0D 0A 00 00 00 00 00 00 19 15 04 07 09  end
0B 0C 0D 0E 10 11 12 13 08 05 07 09 0B 0C 0D 0E
10 11 12 13 08 00 00 00 FF FF 80 00
```

Especially when you need to insert or update a large amount of data, you can write your application in such a way as this, storing all the insert/update statements as a `TString` list or similar, writing `EXECUTE BLOCK` in front of it, concluding with an `END`, and executing it as a single statement.

Firebird 2.0 blocks can also be debugged directly in the *SQL Editor* (or alternatively in the *Block Editor*) using the *Block Debugger*.

There is a limit to the amount of source code that can be transferred in a single package, it may not be larger than 32 Kb. In the case of larger data packets, it is necessary to split them into multiple packages, but this is usually still more efficient that sending each command individually.

Transactions cannot be controlled from inside a block because the block is always a part of your client transaction.

Blocks were implemented in Firebird 2.0. InterBase 2007 introduced something similar but it does not have all the functionalities that Firebird has.

When you are working with IBExpert, you can use IBEBlocks. Simply write `IBEBLOCK` instead of `BLOCK` and it still works!

Chapter 23: 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`, independent of the database server version.

With `EXECUTE IBEBLOCK` you will be able to:

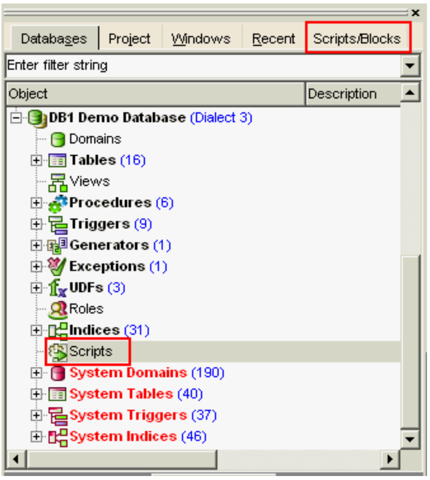
- Work with different connections within a single `IBEBLOCK` at the same time.
- Move (copy) data from one database to another.
- Join tables from different databases.
- Compare data from different databases and synchronize them.
- Populate a table with test data using random values or values from other tables or even from other databases.
- Access external databases via ODBC drivers.
- Transaction control across multiple databases.
- Integration of many IBExpert functions in batch files.
- Dispatch and receive e-mails.

to name but a few of the many functions IBEBlock has to offer.

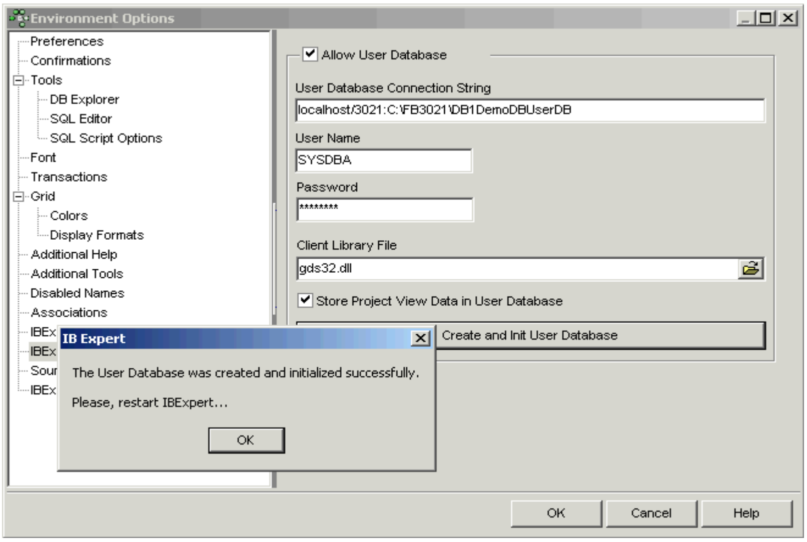
The syntax of `IBEBLOCK` is similar to that of stored procedures but there are many important extensions. For example:

- 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.
- *Code Insight* also supports IBEBlock constants and functions.

Single IBEBlocks can be executed in the *SQL Editor*. They can be debugged in the *SQL Editor* too, 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`.



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*. It is strongly recommended to use the IBExpert *User Database* as the main storage for IBExpert for security reasons (all your IBEBlocks are then stored in a Firebird database). This can be activated using the IBExpert *Options* menu item, *Environment Options / User Database*.



[✓] Allow User Database

User Database Connection String: localhost/3021:C:\FB3021\DB1DemoDBUserDB

User Name: SYSDBA

Password: ********

Client Library File: gds32.dll

[✓] Store Project View Data in User Database

IB Expert

The User Database was created and initialized successfully.

Please, restart IBExpert...

OK

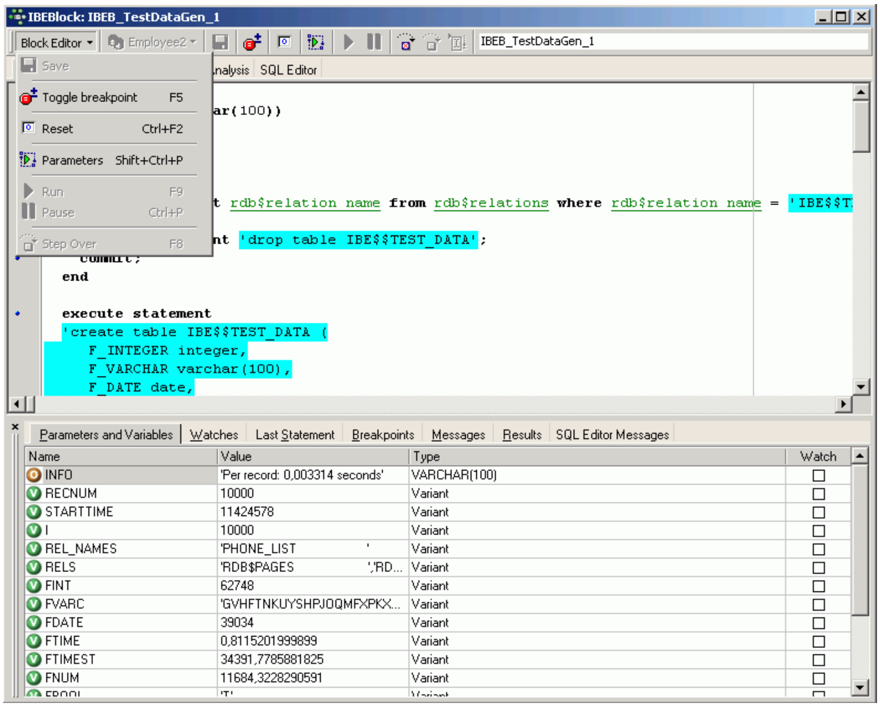
The Database Explorer's *Scripts/Blocks* page displays all existing `IBEScripts` and IBEBlocks saved locally in the database. The Database Explorer's *Database* page also has a node, *Scripts*, displayed in all registered, connected databases.

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 open the *Block Editor*. The IBExpert *Block Editor* can be used to edit and execute IBEBlocks and `IBEScripts`. 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 node 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.

It is even possible to execute Firebird 2.0 blocks stored in registered databases or in the IBExpert *User Database* directly from the Database Explorer. Simply use the Database Explorer right-click context-sensitive menu or open the script in the *Block Editor* and execute using [F9].

When writing new IBEBlocks, do not forget to save the block by clicking on the *Save* icon, in order to commit it, before running it.



```
ar(100))

t rdb$relation name from rdb$relations where rdb$relation name = 'IBE$$T

nt 'drop table IBE$$TEST_DATA';
    commit;
  end

  execute statement
  'create table IBE$$TEST_DATA (
     F_INTEGER integer,
     F_VARCHAR varchar(100),
     F_DATE date,
```

Name	Value	Type	Watch
INFO	'Per record: 0,003314 seconds'	VARCHAR(100)	☐
RECNUM	10000	Variant	☐
STARTTIME	11424578	Variant	☐
I	10000	Variant	☐
REL_NAMES	'PHONE_LIST '	Variant	☐
RELS	'RDB$PAGES ','RD...	Variant	☐
FINT	62748	Variant	☐
FVARC	'GVHFTNKUYSHPJOQMFXPKX...	Variant	☐
FDATE	39034	Variant	☐
FTIME	0,8115201999899	Variant	☐
FTIMEST	34391,7785881825	Variant	☐
FNUM	11684,3228290591	Variant	☐
FBOOL	'T'	Variant	☐

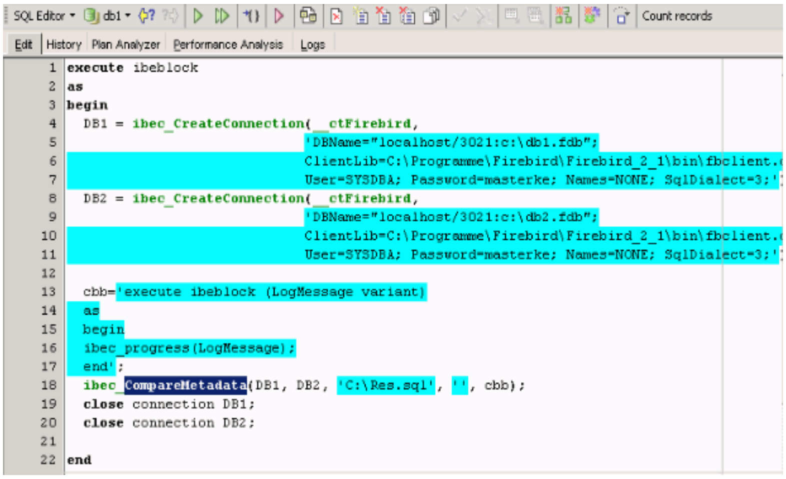
Input parameters can be specified by clicking on the *Parameters* icon (or using [Shift + Ctrl + P]), and the block run in the usual IBExpert way by using [F9] or the green arrow icon.

Similar to the *Procedure and Trigger Debugger*, the *Block Editor* allows you to debug your script or block. The IBExpert *Debugger* is described in detail in *Chapter 12: 9.1 Stored procedure and trigger debugger.*

The range of IBEBlock language includes procedural extensions as well as a huge range of functions. These are all fully documented at www.ibexpert.com/doc.

To illustrate the possible deployment of IBEBlocks, we would like to show an example using the IBExpert *Tools* menu item, *Database Comparer*, and *ibec_CompareMetadata*.

If you've installed your own software application at a customer's you probably won't want to give him his own IBExpert, and let him play around and manipulate the database structure on his own. For such a situation we have created inside the IBEBlock language, for example, `ibec_CompareMetadata`, where you can create your own database connection to two databases, `db1` and `db2`, compare their metadata, and run the resulting script to update the customer's database. (Refer to the IBEBlock online documentation chapters, *ibec_CreateConnection* and *ibec_CompareMetadata* at www.ibexpert.com/doc, for details of syntax and parameters.)

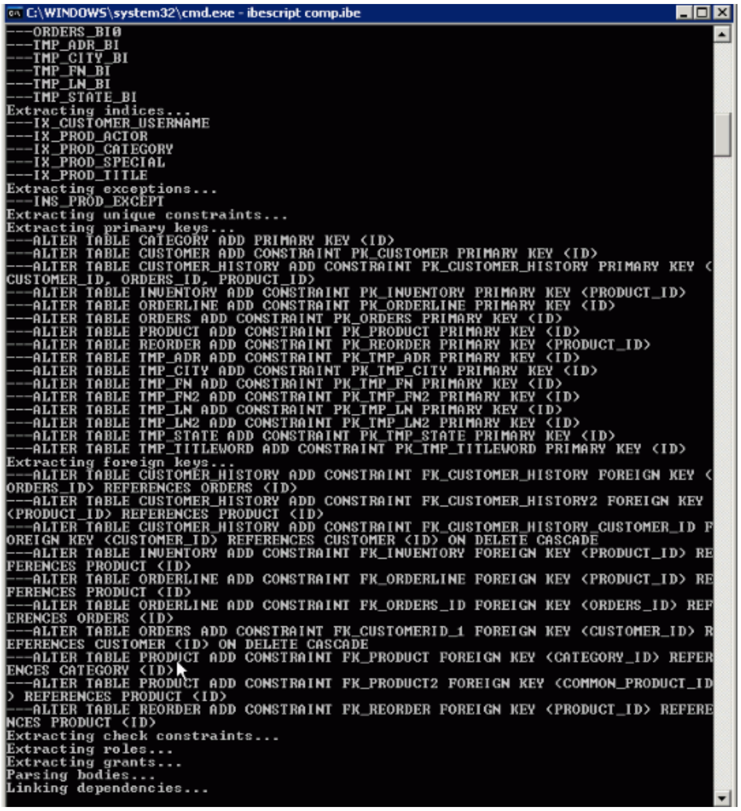


```
execute ibeblock
as
begin
  DB1 = ibec_CreateConnection(__ctFirebird,
                              'DBName="localhost/3021:c:\db1.fdb";
                              ClientLib=C:\Programme\Firebird\Firebird_2_1\bin\fbclient.
                              User=SYSDBA; Password=masterke; Names=NONE; SqlDialect=3;'
  DB2 = ibec_CreateConnection(__ctFirebird,
                              'DBName="localhost/3021:c:\db2.fdb";
                              ClientLib=C:\Programme\Firebird\Firebird_2_1\bin\fbclient.
                              User=SYSDBA; Password=masterke; Names=NONE; SqlDialect=3;'

  cbb='execute ibeblock (LogMessage variant)
  as
  begin
  ibec_progress(LogMessage);
  end';
  ibec_CompareMetadata(DB1, DB2, 'C:\Res.sql', '', cbb);
  close connection DB1;
  close connection DB2;

end
```

This does the same thing as IBExpert's *Database Comparer* but handles it automatically. It uses the reference database, in this case `db1`, and compares it to the customer database, `db2`, followed by the name and directory of the script file. This script file contains an SQL script of all the differences between the two databases.

This IBEBlock can then be stored to a folder, e.g. `IBExpert`, as `comp.ibe`. When `ibescript.exe` is now started on the command line, it starts a command-line version of IBExpert. IBEScript can be found in the main directory of the IBExpert full version.

```
C:\IBExpert\ibescript comp.ibe
```



C:\WINDOWS\system32\cmd.exe - ibescript comp.ibe

```
---ORDERS_BI0
---TMP_ADR_BI
---TMP_CITY_BI
---TMP_FN_BI
---TMP_LN_BI
---TMP_STATE_BI
Extracting indices...
---IX_CUSTOMER_USERNAME
---IX_PROD_ACTOR
---IX_PROD_CATEGORY
---IX_PROD_SPECIAL
---IX_PROD_TITLE
Extracting exceptions...
---INS_PROD_EXCEPT
Extracting unique constraints...
Extracting primary keys...
---ALTER TABLE CATEGORY ADD PRIMARY KEY (ID)
---ALTER TABLE CUSTOMER ADD CONSTRAINT PK_CUSTOMER PRIMARY KEY (ID)
---ALTER TABLE CUSTOMER_HISTORY ADD CONSTRAINT PK_CUSTOMER_HISTORY PRIMARY KEY (
CUSTOMER_ID, ORDERS_ID, PRODUCT_ID)
---ALTER TABLE INVENTORY ADD CONSTRAINT PK_INVENTORY PRIMARY KEY (PRODUCT_ID)
---ALTER TABLE ORDERLINE ADD CONSTRAINT PK_ORDERLINE PRIMARY KEY (ID)
---ALTER TABLE ORDERS ADD CONSTRAINT PK_ORDERS PRIMARY KEY (ID)
---ALTER TABLE PRODUCT ADD CONSTRAINT PK_PRODUCT PRIMARY KEY (ID)
---ALTER TABLE REORDER ADD CONSTRAINT PK_REORDER PRIMARY KEY (PRODUCT_ID)
---ALTER TABLE TMP_ADR ADD CONSTRAINT PK_TMP_ADR PRIMARY KEY (ID)
---ALTER TABLE TMP_CITY ADD CONSTRAINT PK_TMP_CITY PRIMARY KEY (ID)
---ALTER TABLE TMP_FN ADD CONSTRAINT PK_TMP_FN PRIMARY KEY (ID)
---ALTER TABLE TMP_FN2 ADD CONSTRAINT PK_TMP_FN2 PRIMARY KEY (ID)
---ALTER TABLE TMP_LN ADD CONSTRAINT PK_TMP_LN PRIMARY KEY (ID)
---ALTER TABLE TMP_LN2 ADD CONSTRAINT PK_TMP_LN2 PRIMARY KEY (ID)
---ALTER TABLE TMP_STATE ADD CONSTRAINT PK_TMP_STATE PRIMARY KEY (ID)
---ALTER TABLE TMP_TITLEWORD ADD CONSTRAINT PK_TMP_TITLEWORD PRIMARY KEY (ID)
Extracting foreign keys...
---ALTER TABLE CUSTOMER_HISTORY ADD CONSTRAINT FK_CUSTOMER_HISTORY FOREIGN KEY (
ORDERS_ID) REFERENCES ORDERS (ID)
---ALTER TABLE CUSTOMER_HISTORY ADD CONSTRAINT FK_CUSTOMER_HISTORY2 FOREIGN KEY
(PRODUCT_ID) REFERENCES PRODUCT (ID)
---ALTER TABLE CUSTOMER_HISTORY ADD CONSTRAINT FK_CUSTOMER_HISTORY_CUSTOMER_ID F
OREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (ID) ON DELETE CASCADE
---ALTER TABLE INVENTORY ADD CONSTRAINT FK_INVENTORY FOREIGN KEY (PRODUCT_ID) RE
FERENCES PRODUCT (ID)
---ALTER TABLE ORDERLINE ADD CONSTRAINT FK_ORDERLINE FOREIGN KEY (PRODUCT_ID) RE
FERENCES PRODUCT (ID)
---ALTER TABLE ORDERLINE ADD CONSTRAINT FK_ORDERS_ID FOREIGN KEY (ORDERS_ID) REF
ERENCES ORDERS (ID)
---ALTER TABLE ORDERS ADD CONSTRAINT FK_CUSTOMERID_1 FOREIGN KEY (CUSTOMER_ID) R
EFERENCES CUSTOMER (ID) ON DELETE CASCADE
---ALTER TABLE PRODUCT ADD CONSTRAINT FK_PRODUCT FOREIGN KEY (CATEGORY_ID) REFER
ENCES CATEGORY (ID)
---ALTER TABLE PRODUCT ADD CONSTRAINT FK_PRODUCT2 FOREIGN KEY (COMMON_PRODUCT_ID
) REFERENCES PRODUCT (ID)
---ALTER TABLE REORDER ADD CONSTRAINT FK_REORDER FOREIGN KEY (PRODUCT_ID) REFERE
NCES PRODUCT (ID)
Extracting check constraints...
Extracting roles...
Extracting grants...
Parsing bodies...
Linking dependencies...
```

So if you want to update your customer's database, you need `ibescript.exe`, taken from a IBExpert VAR License or IBExpert Server Tools, you need a script file, e.g. `comp.ibe`, a reference database (i.e. the new version of the database), and the customer's database. Then you simply execute the script to create a script listing all differences between the two databases, which can be done in a batch file or using the `ibescript.dll` directly implemented in your application. In the next step, after the SQL differences are created, you can say

```
        C:\IBExpert>ibescript \res.sql
```

and restore in the main directory.

So there are two steps in the command-line window to update any database to a new structure based on a reference database, without any human interaction. If you look at the script closely, the order in which new objects need to be created, and the order in which old objects are deactivated is sometimes not so easy to understand. Let's imagine you have a stored procedure that uses another stored procedure which in turn uses a third stored procedure. You want to change the third procedure. So you need to deactivate your first and second procedure so that the third procedure is free for changes. This is all handled automatically by IBExpert and its script tools.

The source code of your procedures and IBEBlocks may contain sensitive information. So there is the possibility to use the `-e` parameter:

```
C:\IBExpert> ibescript comp.ibe –e
```

This encrypts the script into binary encrypted code. The file is converted into an ESQL file (`comp.ibe.esql`). If the password is used, it can only be encrypted with the password. This particular file cannot be decrypted, but IBExpert can still execute

```
C:\IBExpert> ibescript comp.ibe.esql
```

without any decryption. And if you ignore the callback (`ibec_Progress`) functions:

```
cbb='execute ibeblock (LogMessage variant)
as
begin
  ibec_Progress(LogMessage);
end';
```

no messages will appear on the screen during execution. This callback function `cbb` is just a string that represents another IBEBlock, `ibec_Progress`, with this function inside.

Another example of what you can do with such an IBEBlock is ODBC access:

```
execute ibeblock
as
begin
  FBX = ibec_CreateConnection(__ctFirebird,
  'DBName="localhost/3021:C:\db1.fdb; Clientlib=gds32.dll;
  User=SYSDBA; Password=masterke; Names=NONE; SQLDialect=3;')
  OBX = ibec_CreateConnection(__ctODBC,
  'DBQ=C:\demo.mdb;DRIVER=Microsoft Access Driver (*.mdb)');
  use OBX;
  for
    select CustNo, Company, Addr1
    from customer
    order by company
    into :CustNo, :Company, :Addr1
    do
    begin
      use FBX;
      insert into "customerx"
      ("CustNo", "Company", "Addr")
      values
      (:CustNo, :Company, :Addr1);
    end
  use FBX;
  commit;
  ibec_CloseConnection(OBX);
  ibec_CloseConnection(FBX);
end
```

Here the Firebird connection and the the ODBC connection has been created, and then a `SELECT FOR` statement is executed on the ODBC connection.

The returning values are put into the variables defined. If you do not want to declare your variables, IBExpert doesn't require it. The `FOR SELECT` statement then switches to the Firebird database (`FBX`). The data is then inserted into the Firebird database's `CUSTOMERX` table. Finally the Firebird connection needs to be committed and then both connections closed. There is also the possibility to do some exception handling, and for example the `ibec_CopyData` is able to copy data to a local Firebird table from any source.

If you don't know what tables already exist in your ODBC partner, it is sometimes hard to write your own `CREATE TABLE` statements compatible to the one that you need in your Firebird database. There is a useful tool in IBExpert, the *ODBC Viewer (Chapter 19)*. This can be used easily to open the same database.

In the Windows Control Center / Data Sources (ODBC) there is a `demo.mdb`, which is based on an MSAccess ODBC driver. Double-clicking on the database name in IBExpert's *ODBC Viewer* directly opens the ODBC driver and displays the `CUSTOMER` table and its data. To select a table whose data is required in the Firebird database, use the menu item *Export to script/table*, select the *Export into a table* page, give the new table a name, and simply export.

This way you very quickly have a new database table full of the data that was in the original Access database. If the table data is removed, so that only the metadata structure remains, you can see how the IBEBlock used earlier transfers the data automatically. This functionality does not depend on Firebird/InterBase. It can be used between any databases with ODBC connectivity.

Further IBEBlock examples can be viewed in *Chapter 17: Reporting*.

Chapter 24: Database Documentation

Database documentation is unfortunately an often unpopular but necessary part of any developer's (and administrator's) work. With a little effort though, IBExpert provides the necessary tools to generate up-to-date documentation and the click of a mouse.

1. Object and field descriptions

All IBExpert object editors offer *Description* fields for all objects and fields.

The *Description page* should be used to describe all database object; the *Description field* for describing the fields. Although this may seem like a tedious task, it is only necessary to write a brief description for each object once; this is then displayed in all automatically-generated documentation.

2. Template short cuts

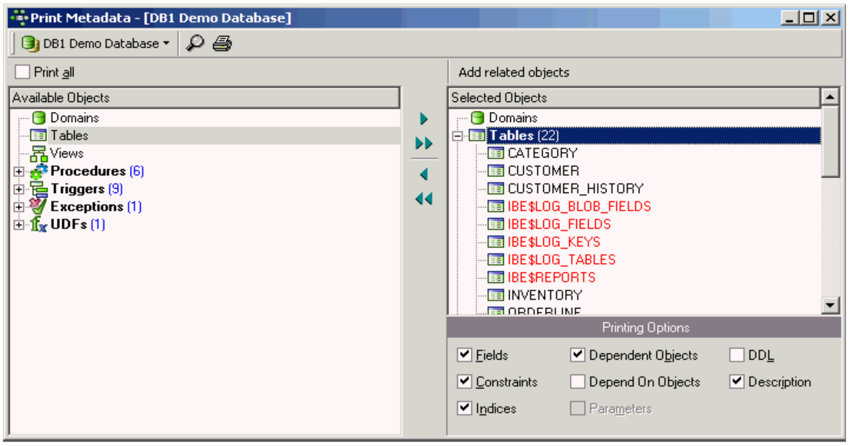
To document code quickly and efficiently, use the IBExpert *Options* menu item, *Keyboard templates* to create your own shortcut: use to insert author, date and time fields automatically and rapidly, with a simple button click. For example, the abbreviation `ME` can be specified with the expansion `/* #author #date */` (click the *Author* and *Date* buttons to insert the fields, add the comment symbols, done!). This results in a simple documentation comment at the beginning of all SQLs listing author and date (i.e. `/* SYSDBA 08/07/2009 */`) simply by typing `ME`!

3. Print metadata

Print Metadata prints the database metadata, along with dependencies, descriptions, 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.

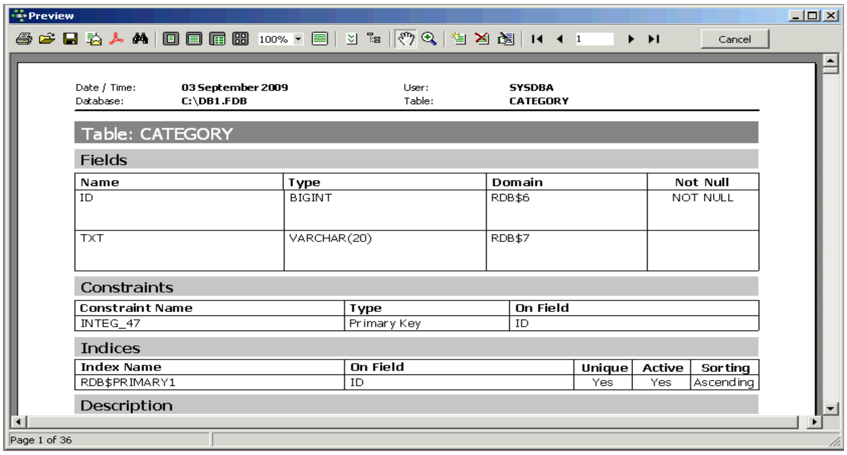
When one of the selected database objects or object groups is highlighted, a number of check options appear in the lower right panel. These include:



- [ ] Print all
- [✓] Fields
- [✓] Constraints
- [✓] Indices
- [✓] Dependent Objects
- [ ] Depend On Objects
- [ ] Parameters
- [ ] DDL
- [✓] Description

- 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!



Date / Time:	03 September 2009	User:	SYSDBA
Database:	C:\DB1.FDB	Table:	CATEGORY

Table: CATEGORY

Fields

Name	Type	Domain	Not Null
ID	BIGINT	RDB$6	NOT NULL
TXT	VARCHAR(20)	RDB$7	

Constraints

Constraint Name	Type	On Field
INTEG_47	Primary Key	ID

Indices

Index Name	On Field	Unique	Active	Sorting
RDB$PRIMARY1	ID	Yes	Yes	Ascending

Description

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.

4. Generate HTML documentation

Using this IBExpert *Tools* menu item, *Generate HTML documentation*, complete or partial database documentation can be generated for a named, connected database. This option is an excellent feature for software documentation, particularly if the the description fields (available in all IBExpert object editors) were always used as objects were created in the database. This feature fully supports UTF8.

The toolbar displays the selected connected database. The drop-down lists offers a choice of all registered databases. The default output directory can be overwritten if wished.

The *Generate HTML Documentation* Editor is similar to the *Extract Metadata* Editor with its *Select Objects Tree* window. 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 offers a drop-down list of character sets which can be selected for the documentation, and lists a series of check box options 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* (cascaded style sheets) page displays the code for the HTML page template. With knowledge of HTML these style sheets can be adapted as wished. The *Output* page displays the code used to generate the HTML documentation.

An IBExpert *Help* window is automatically opened following successful generation of the HTML documentation. This begins with a list of contents followed by detailed documentation of the individual objects.

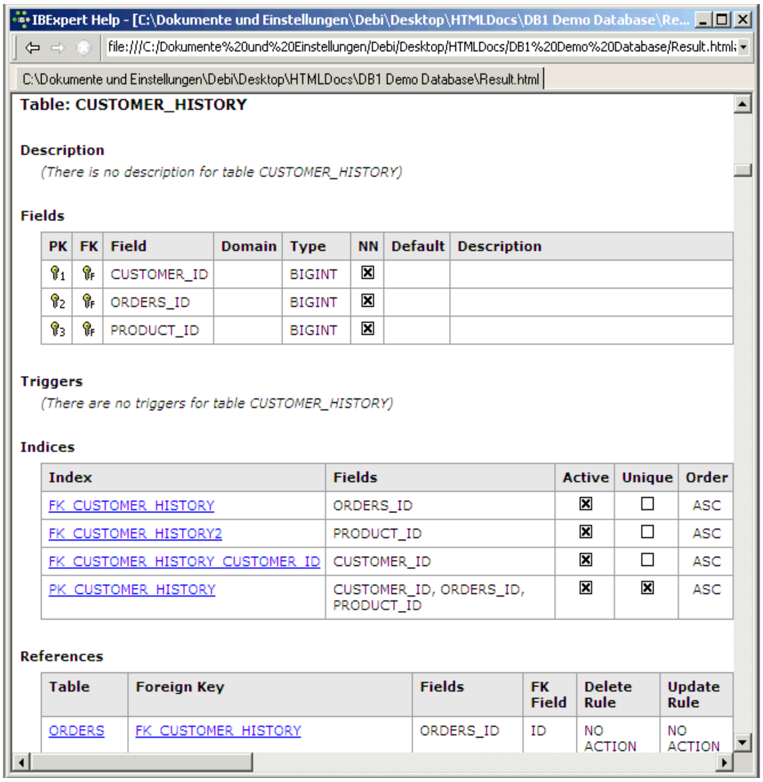

IBExpert Help - [C:\Dokumente und Einstellungen\Debi\Desktop\HTMLDocs\DB1 Demo Database\Re...
file:///C:/Dokumente%20und%20Einstellungen/Debi/Desktop/HTMLDocs/DB1%20Demo%20Database/Result.html
C:\Dokumente und Einstellungen\Debi\Desktop\HTMLDocs\DB1 Demo Database\Result.html

Table: CUSTOMER_HISTORY

Description

(There is no description for table CUSTOMER_HISTORY)

Fields

PK	FK	Field	Domain	Type	NN	Default	Description
1	F	CUSTOMER_ID		BIGINT	☒		
2	F	ORDERS_ID		BIGINT	☒		
3	F	PRODUCT_ID		BIGINT	☒		

Triggers

(There are no triggers for table CUSTOMER_HISTORY)

Indices

Index	Fields	Active	Unique	Order
FK_CUSTOMER_HISTORY	ORDERS_ID	☒	☐	ASC
FK_CUSTOMER_HISTORY2	PRODUCT_ID	☒	☐	ASC
FK_CUSTOMER_HISTORY_CUSTOMER_ID	CUSTOMER_ID	☒	☐	ASC
PK_CUSTOMER_HISTORY	CUSTOMER_ID, ORDERS_ID, PRODUCT_ID	☒	☒	ASC

References

Table	Foreign Key	Fields	FK Field	Delete Rule	Update Rule
ORDERS	FK_CUSTOMER_HISTORY	ORDERS_ID	ID	NO ACTION	NO ACTION

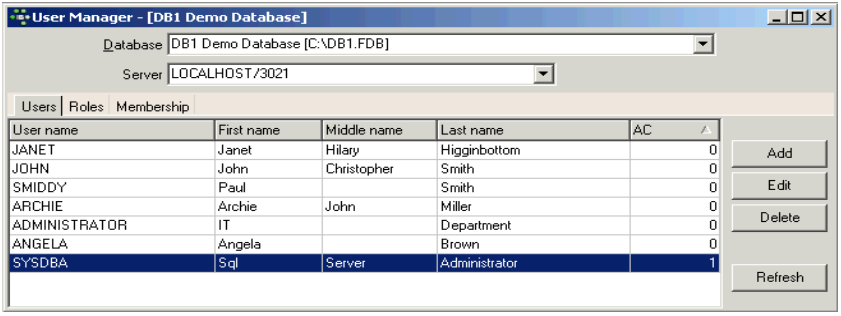
The `results.html` can of course be opened in any popular browser.

By clicking on one of the object subjects, such as tables, a table of all such objects (i.e. all tables) for this database appear. Clicking on the individual objects then automatically displays the description (if existent) and the definition.

Chapter 25: 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.

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 a list of all registered databases (drop-down list). The server connection may be altered using the drop-down list.



Database: DB1 Demo Database [C:\DB1.FDB]

Server: LOCALHOST/3021

User name	First name	Middle name	Last name	AC
JANET	Janet	Hilary	Higginbottom	0
JOHN	John	Christopher	Smith	0
SMIDDY	Paul		Smith	0
ARCHIE	Archie	John	Miller	0
ADMINISTRATOR	IT		Department	0
ANGELA	Angela		Brown	0
SYSDBA	Sql	Server	Administrator	1

Select the database and server (local or remote) to administrate.

If the registered database is using Firebird version 2.1 or higher and the *Trusted authentication* option has been specified in the *Database Registration Info*, then Windows "Trusted User" security is also supported here.

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 Firebird/InterBase `GRANT` and `REVOKE` commands (see Chapter 26: Grant Manager for details), or the front-end program.

Please note that to create, edit and delete users and roles you should have the rights of server administrator.

1. 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. The *AC* (Active Users) column shows how many active connections a user has to the specified database. This works only with active databases. The *Refresh* button has been added to refreshes the 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 the database owner, as users are created for all databases on the server). Simply click the *Add* button, and complete the *New User* form. 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 necessary, for example, if a user marries.

This list contains all current 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.

2. 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 (using IBExpert's *Grant Manager*), 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.

3. 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.

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 (please refer to *Membership page* below). Role users and rights can then 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*.

4. Membership page

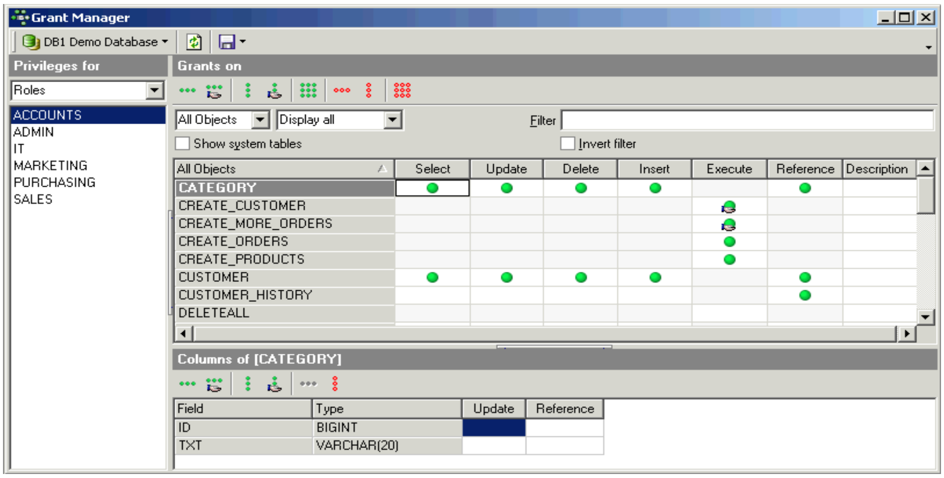
The *Membership* page shows which users have been granted rights to which roles.

The abbreviations *G* stands for *Granted*, *M* for *Member* of selected role and *AO* for w*ith ADMIN option*. Users can be assigned roles simply by selecting the user, and checking either the *Grant/Member* of selected role 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.

Chapter 26: 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. It is possible to grant rights for database objects on the *Grants* page in the object editors.

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 Database Explorer. Alternatively use the Database Explorer's right mouse-click menu item *Edit Role* or key combination [Ctrl + O].



ACCOUNTS
ADMIN
IT
MARKETING
PURCHASING
SALES

All Objects	Select	Update	Delete	Insert	Execute	Reference	Description
CATEGORY	●	●	●	●		●	
CREATE_CUSTOMER					●		
CREATE_MORE_ORDERS					●		
CREATE_ORDERS					●		
CREATE_PRODUCTS					●		
CUSTOMER	●	●	●	●		●	
CUSTOMER_HISTORY						●	
DELETEALL							

Field	Type	Update	Reference
ID	BIGINT		
TXT	VARCHAR(20)		

The *Grant Manager* Editor offers options to select the database, and select a group for which privileges are to be assigned. Once a database object has been selected, a full list of such users/objects in this database is displayed in the panel directly below.

The *Grants* toolbar enables the user to quickly assign or revoke rights to one or more objects, or for one or more operations. These same operations can also be specified using the right-click pop-up menu.

It is possible to filter the grants 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 drop-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.

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.

A further menu option here is *Show Column Privileges* (check box option). This blends the lower window in and out, 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, when creating or altering a database object! They allow the developer to check existing permissions 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.

1. 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 `CREATE_MORE_ORDERS`, use the following:

```
GRANT EXECUTE
ON PROCEDURE CREATE_MORE_ORDERS
TO Janet, John;
```

Firebird/InterBase 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>`).

2. 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. Firebird/InterBase 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. 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.

Chapter 27: Database Statistics

The IBExpert *Services* menu item, *Database Statistics*, reveals a wealth of information about your database. When approaching the *Database Statistics* analysis, it is important to know what information is available, which information is important and how to interpret and use it to solve performance problems.

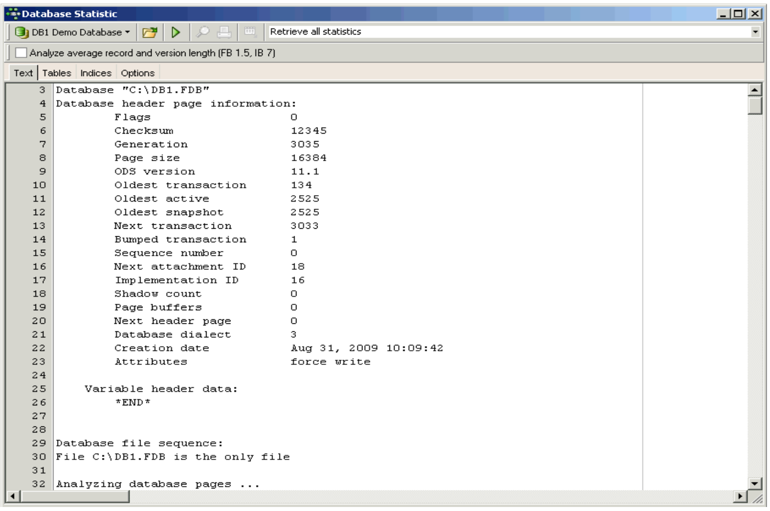
A common performance problem is that the database gradually becomes slower and slower. This is usually due to an open transaction somewhere in the database. Look at the number of record versions (total record versions). These exist because Firebird still needs to store the old data for old open transactions. This is handled internally by a transaction number.

In a production database with multiple users you will often see record versions, but if there are no old open transactions the database will delete these older record versions automatically when they are no longer needed, i.e. following a commit or rollback. The garbage collector cannot work if there are open transactions anywhere.

The oldest and newest transaction numbers can be found in the summary at the top of the log found in the *Text* page. The larger the difference between the *Oldest active transaction (OAT)* and the *Next transaction*, the bigger performance problems you will encounter. The Firebird server does not just administrate record versions for the database object which still has an open transaction, but for the entire database. In repeatable read mode a snapshot is made of the whole database, as soon as a transaction is started. When the transaction is completed (i.e. committed or rolled back) the garbage collector will then delete all old record versions that are no longer needed.

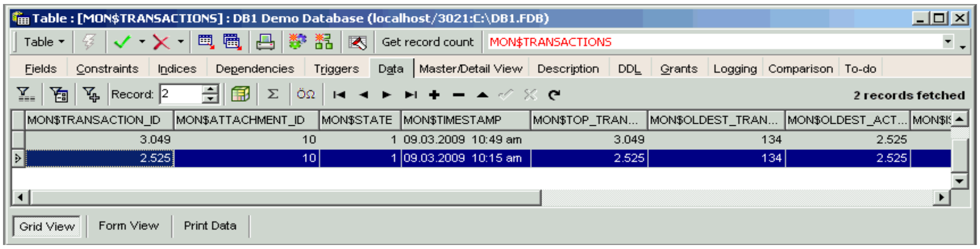
The log file and the *Tables* page show the statistics for all tables: here you can ascertain which tables have large amount of record versions being held by the server. The max number of versions means there is one record that has this amount of different versions. This indicates that there is still one active transaction in the database so that the old record versions cannot be deleted.

To find out what or who is causing such a problem, look at the server while the database is in use.



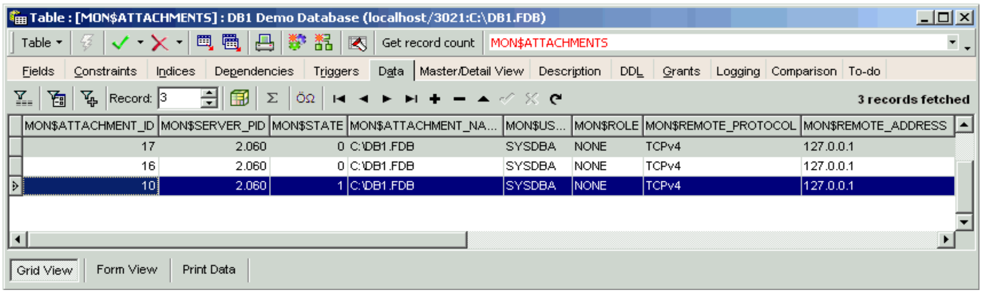
```
Database "C:\DB1.FDB"
Database header page information:
         Flags                      0
         Checksum                   12345
         Generation                 3035
         Page size                  16384
         ODS version                11.1
         Oldest transaction         134
         Oldest active              2525
         Oldest snapshot            2525
         Next transaction           3033
         Bumped transaction         1
         Sequence number            0
         Next attachment ID         18
         Implementation ID          16
         Shadow count               0
         Page buffers               0
         Next header page           0
         Database dialect           3
         Creation date              Aug 31, 2009 10:09:42
         Attributes                 force write

    Variable header data:
         *END*


Database file sequence:
File C:\DB1.FDB is the only file

Analyzing database pages ...
```

The above summary shows us that the next transaction is number 3033, and the oldest active transactions number is 2525. If system tables are activated in the IBExpert *Database Explorer* (check the options using the IBExpert *Database* menu item, *Database Registration Info / Database Explorer* page), you can view and open the Firebird 2.1 `MON$TRANSACTIONS` table. On the *Data* page there is an entry in this example for transaction 2525:



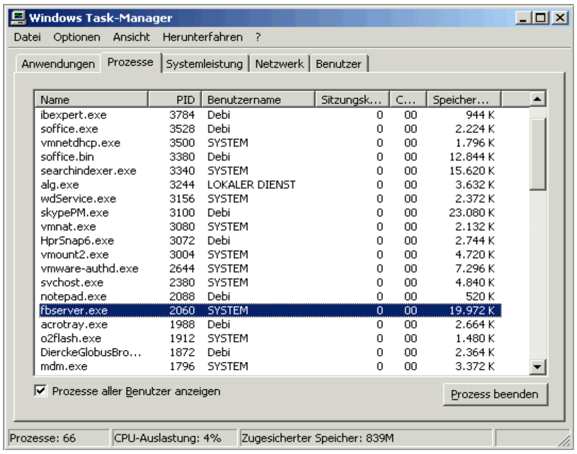
MON$TRANSACTION_ID	MON$ATTACHMENT_ID	MON$STATE	MON$TIMESTAMP	MON$TOP_TRAN...	MON$OLDEST_TRAN...	MON$OLDEST_ACT...	MON$I
3.049	10	1	09.03.2009 10:49 am	3.049	134	2.525	
2.525	10	1	09.03.2009 10:15 am	2.525	134	2.525	

This transaction has an attachment ID number 10. It was started at 10:15 and has been active for over 30 minutes. A typical transaction would not be active for that length of time. More information concerning this attachment ID 10 can be found in the `MON$ATTACHMENTS` table:



MON$ATTACHMENT_ID	MON$SERVER_PID	MON$STATE	MON$ATTACHMENT_NA...	MON$US...	MON$ROLE	MON$REMOTE_PROTOCOL	MON$REMOTE_ADDRESS
17	2.060	0	C:\DB1.FDB	SYSDBA	NONE	TCPv4	127.0.0.1
16	2.060	0	C:\DB1.FDB	SYSDBA	NONE	TCPv4	127.0.0.1
10	2.060	1	C:\DB1.FDB	SYSDBA	NONE	TCPv4	127.0.0.1

Here the `MON$SERVER_PID` is displayed. If you go to the Windows Task Manager's *Processes* page, you will see the process ID numbers (you may first need to select the column for display using the *View* menu item, *Select columns ...*, and check the *PID (Process ID)* column). You can then trace the number of the Firebird instance that is used by the server. Furthermore this table also displays the user and role name, the remote address and, if you use the new Firebird clients, you will also see the remote PID.

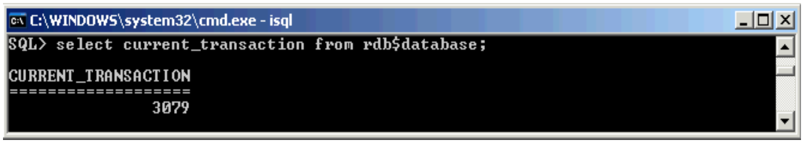


Name	PID	Benutzername	Sitzungsk...	C...	Speicher...
ibexpert.exe	3784	Debi	0	00	944 K
soffice.exe	3528	Debi	0	00	2.224 K
vmnetdhcp.exe	3500	SYSTEM	0	00	1.796 K
soffice.bin	3380	Debi	0	00	12.844 K
searchindexer.exe	3340	SYSTEM	0	00	15.620 K
alg.exe	3244	LOKALER DIENST	0	00	3.632 K
wdService.exe	3156	SYSTEM	0	00	2.372 K
skypePM.exe	3100	Debi	0	00	23.080 K
vmnat.exe	3080	SYSTEM	0	00	2.132 K
HprSnap6.exe	3072	Debi	0	00	2.744 K
vmount2.exe	3004	SYSTEM	0	00	4.720 K
vmware-authd.exe	2644	SYSTEM	0	00	7.296 K
svchost.exe	2380	SYSTEM	0	00	4.840 K
notepad.exe	2088	Debi	0	00	520 K
fbserver.exe	2060	SYSTEM	0	00	19.972 K
acrotray.exe	1988	Debi	0	00	2.664 K
o2flash.exe	1912	SYSTEM	0	00	1.480 K
DierckeGlobusBro...	1872	Debi	0	00	2.364 K
mdm.exe	1796	SYSTEM	0	00	3.372 K

In the above example the Windows Task Manager shows me that the PID 2060 has started this transaction.

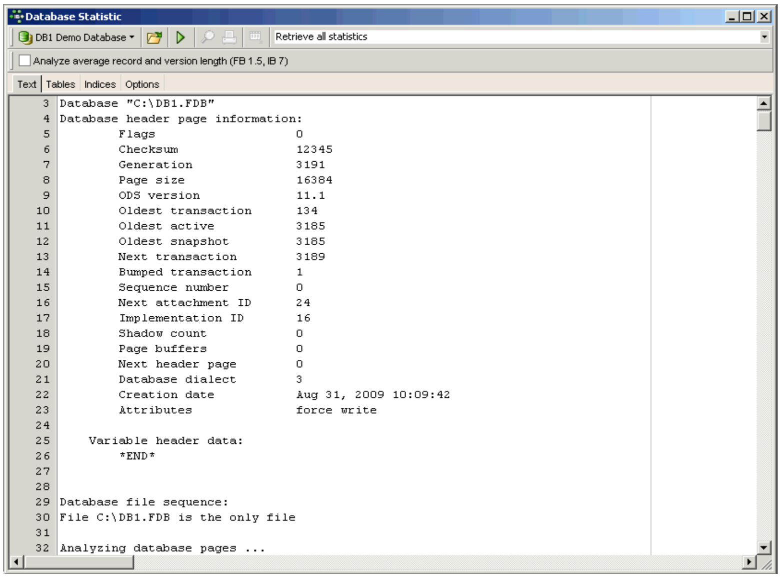
Now you only need to find out who/what is using the Firebird server with the transaction number 2525. Connect via `isql` or using IBExpert's *SQL Editor* to find out your own current transaction number using:

```
SELECT CURRENT_TRANSACTION FROM RDB$DATABASE;
```



```
C:\WINDOWS\system32\cmd.exe - isql
SQL> select current_transaction from rdb$database;

CURRENT_TRANSACTION
===================
               3079
```

Once the initiator of the oldest transaction is found it can be committed or rolled back. If we now go back to the `MON$ATTACHMENTS` table the oldest record is no longer 2525, and if we go back to the *Database Statistics* and run it again, we see the *Oldest active transaction* is now 3185:



```
Database "C:\DB1.FDB"
Database header page information:
        Flags                   0
        Checksum                12345
        Generation              3191
        Page size               16384
        ODS version             11.1
        Oldest transaction      134
        Oldest active           3185
        Oldest snapshot         3185
        Next transaction        3189
        Bumped transaction      1
        Sequence number         0
        Next attachment ID      24
        Implementation ID       16
        Shadow count            0
        Page buffers            0
        Next header page        0
        Database dialect        3
        Creation date           Aug 31, 2009 10:09:42
        Attributes              force write

    Variable header data:
        *END*


Database file sequence:
File C:\DB1.FDB is the only file

Analyzing database pages ...
```

The *Oldest snapshot* transaction number 3185 shows where the garbage collector will start its work.

The IBExpert *Database Statistics* are a vital tool for solving performance problems and discerning areas for fine-tuning. They are also useful, for example, for determining the largest table, are there any empty tables, average record length (could you increase performance by splitting, for example, a large table into several smaller ones?), analyzing indices (comparing their actual selectivity with the real selectivity - do you need to recompute the selectivity of all indices?, which indices are unused or useless, analyze their depth, etc. etc.).

Chapter 28: Optimizing Database Performance

The aim of all developers is to develop a trouble-free efficient database. However as time goes on and more and more data is added, altered and deleted, unforeseen performance problems are often encountered. It is important to recognize any impending problems as early as possible. IBExpert offers a number of tools for this.

Some of these features, such as *Indices*, *Performance Analysis*, *SP/Triggers/Views Analyzer*, *Logging*, *Database Backup & Restore*, and *Database Statistics* have already been mentioned in previous chapters. 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.

1. Operating systems

Certainly the most 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 individual 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.

2. 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.

3. 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.

4. 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.

5. 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 memory, 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*.

6. 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 discernible 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, particularly 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.

7. Firebird benchmarks tests

The IBExpert demo database, DB1, 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.

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!

8. Optimizing the database

1. Split complex tables into several smaller ones (Database normalization).

   For reasons of compatibility 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.

7. If you are upgrading from an older Firebird version to the new 2.1 version, it is also important that you upgrade all your clients accordingly. The Firebird 2.1 client can communicate much more effectively with the Firebird 2.1 server, which can mean performance improvements of up to 40%!

9. 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 *Developing a data model* in chapter 7 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 and index selectivity*, *Fehler: Referenz nicht gefunden*, *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.

10. The Firebird Optimizer and index statistics

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.

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 million) 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.

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.

11. Automating the recalculation of index statistics

A common problem is that when an application is delivered to a customer, an "empty" database is supplied, i.e. it contains only the metadata and no customer data. As different customers enter different amounts of data, with time some may complain that their application is too slow in certain areas. This is most often due to the indices' statistics not having been calculated up to date (or not having been calculated at all!), which means that the Optimizer cannot use the indices efficiently to process queries.

If you want to have your software working at its most efficient, always use up-to-date statistic values to maximize performance (if one customer has many orders for few products all serviced by two employees and another few orders for many products, serviced by 100 employees, the index statistics and hence selectivity, will obviously develop differently). Without updating the index statistics regularly as more and more data is added you will incur performance problems (eg. all males living in Little Bigton). The command for this is:

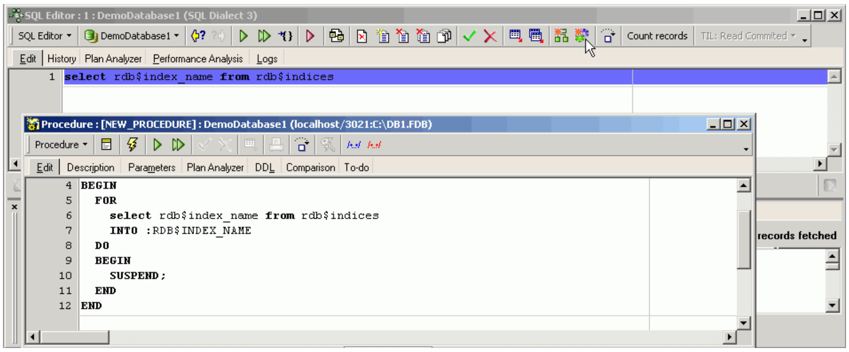
```
SET STATISTICS INDEX
```

The index names can be found in a system table called `RDB$INDICES`. This table also displays the index value of each index in the `RDB$STATISTICS` column.

Use:

```
SELECT RDB$INDEX_NAME FROM RDB$INDICES
```

to obtain list of all index names. A procedure can then be created directly from this selecting into *Local variables*.



SQL Editor : 1 : DemoDatabase1 (SQL Dialect 3)

```
select rdb$index_name from rdb$indices
```

Procedure : [NEW_PROCEDURE] : DemoDatabase1 (localhost/3021:C:\DB1.FDB)

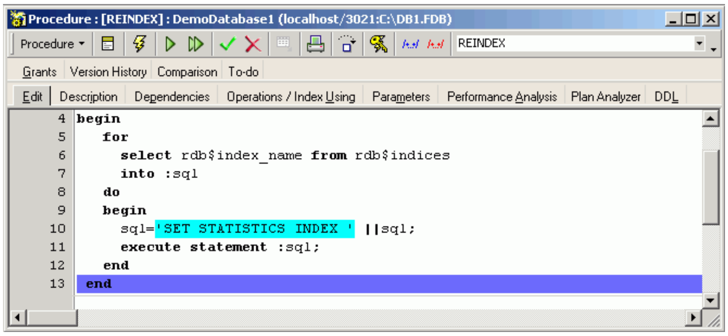
```
BEGIN
  FOR
    select rdb$index_name from rdb$indices
    INTO :RDB$INDEX_NAME
  DO
  BEGIN
    SUSPEND;
  END
END
```

(This and the following illustration show the *Procedure Editor* with deactivated *Lazy Mode*.)

Simply rename the procedure to `REINDEX`, alter the variable to declare variable `sql carchar(300)`; and also into `:sql`. After the index name has been put into the variable, it should say:

```
BEGIN
  SQL='SET STATISTICS INDEX ' ||SQL;
  EXECUTE STATEMENT :SQL;
```

Here the `SET STATISTICS INDEX` statement has been combined with the `sql` variable. And inside a Firebird stored procedure it is possible to use this SQL statement, which is inside a variable, and execute it directly from the procedure.



```
begin
   for
     select rdb$index_name from rdb$indices
     into :sql
   do
   begin
     sql='SET STATISTICS INDEX ' ||sql;
     execute statement :sql;
   end
 end
```

To run simply type:

```
EXECUTE PROCEDURE REINDEX
```

You do not even need to shut down the database to recompute the selectivity of indices.

Do this regularly and the Optimizer will be able to use indices efficiently.

Chapter 29: Avoiding Server Problems

1. Typical causes of server problems

1.1 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 mechanism 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.

1.2 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 they're running out of space. When the hard drive begins to become full, Windows stores data pages anywhere it can find space. Which of course degrades performance when searching for and uploading the data on these pages. Please refer to *Chapter 28: 4. Temporary files* for further information.

Hardware defects can happen at any time and can have disastrous effects, if you can't react quickly. The best defense against such a problem is to run a database shadow on another server or external hard drive. Please refer to *Chapter 16: 3.1. Working with shadows* for further information.

1.3 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.

- **Windows system restore**: On Windows *My Computer / System Properties* the automatic *System restore* can be disabled. This also prevents Windows copying all manner of files into the `Win/System32/dllcache` directory (it not been unknown to discover files of 5GB and more in this directory!).
- **Automatic Windows update**: the infamous automatic Windows update with its automatic rebooting is the cause of many Firebird server machines suddenly being shut down, because

no one 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.

2. 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 FreeAdhocUDF, and complement it if necessary.

Recommendation:

- Use only robust UDF libraries, such as RFunc.
- Check every UDF you've written yourself not just once, but 10 times!

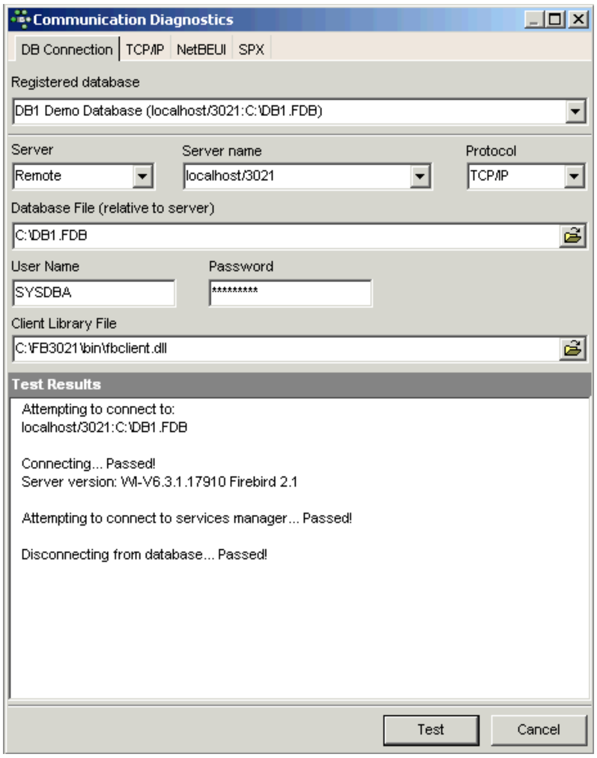
If you're using two difference Firebird/InterBase flavors concurrently, check that the correct `fbclient.dll/gds32.dll` version is installed on the server and all clients. You'd be amazed how often DBAs are surprised by this or that previously undiscovered `dlls` suddenly turning up, because somewhere 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.

3. Communication Diagnostics

IBExpert's *Communication Diagnostics* can be started from the *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:



Registered database: DB1 Demo Database (localhost/3021:C:\DB1.FDB)

Server: Remote

Server name: localhost/3021

Protocol: TCP/IP

Database File (relative to server): C:\DB1.FDB

User Name: SYSDBA

Password: *********

Client Library File: C:\FB3021\bin\fbclient.dll

Test Results

```
Attempting to connect to:
localhost/3021:C:\DB1.FDB

Connecting... Passed!
Server version: WI-V6.3.1.17910 Firebird 2.1

Attempting to connect to services manager... Passed!

Disconnecting from database... Passed!
```

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.

The following protocols are supported:

- **TCP/IP** (worldwide standard)
- **SPX**: which used to be used by Novell; now even Novell supports TCP/IP.
- **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.

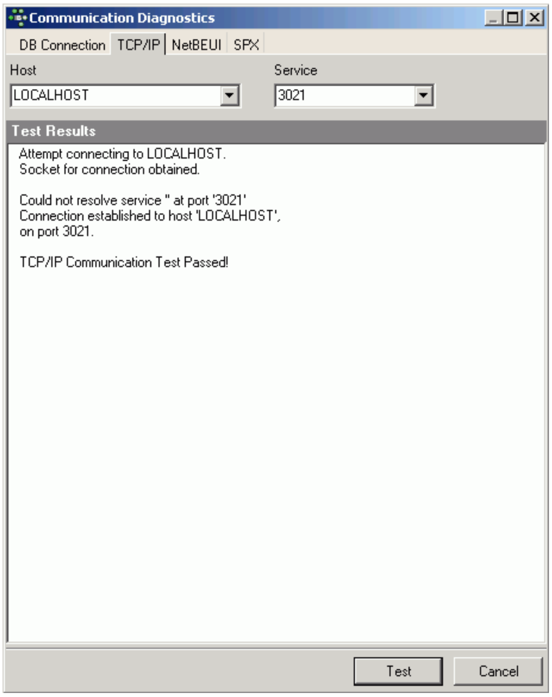
Should problems occur, switch to the relevant protocol page and test again.

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 Firebird and InterBase. 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 Firebird/InterBase 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 that 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.



Host: LOCALHOST
Service: 3021

```
Attempt connecting to LOCALHOST.
Socket for connection obtained.

Could not resolve service '' at port '3021'
Connection established to host 'LOCALHOST',
on port 3021.

TCP/IP Communication Test Passed!
```

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.

Chapter 30: Command-Line Utilities

1. IBExpert's IBEScript.exe

`IBEScript.exe` can be found in the IBExpert root directory, and needs to be started from DOS.

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).
-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.

WARNING: all options are case-sensitive!

There are two possible ways to encrypt/decrypt scripts and to execute encrypted scripts:

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
```

2. IBEScript.dll

You can use `IBEScript.dll` 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`.

For regulations regarding distribution of any of the IBExpert modules (`ibexpert.exe`, `ibescript.exe`, `ibescript.dll`) together with your application, please refer to www.ibexpert.com.

2.1 IBEScriptDll Readme.txt

`IBEScript.dll` exports the following functions: `ExecScriptFile`, which executes script from file and `ExecScriptText` which executes script from string buffer.

`CONNECT`: connects to the database if there is no `CONNECT` statement in the script.

Refer to the demo application in the `IBEScriptDll` folder for examples of `ExecScriptFile`.

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;
```

3. Firebird/InterBase Command-Line Tools

Several command-line tools are provided with Firebird/InterBase. 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 Firebird/InterBase supports. The command-line tools include the following:

- `fbguard.exe`
- `fbserver.exe`
- `fb_inet_server.exe`
- `fbsvcmgr`
- `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. The individual tools are described in detail in the IBExpert online documentation.

Epilogue

Hopefully you have now gained an insight into the huge potential of IBExpert and Firebird. Unfortunately it is not possible withing the realms of this introductory guide to present all features and functions. Should you wish to deepen/broaden your knowledge we recommend the comprehensive online documentation at www.ibexpert.com/doc where, in addition to a complete and detailed IBExpert documentation, you will find a wealth of database technology articles, language references and Firebird release notes and quick start guides.

And in the rare case that you should not find an answer to your problem there, we recommend the forum at http://www.firebirdexperts.com/.

Hopefully the relationship you have begun with IBExpert and Firebird will last for many years to come!

INDEX

A

ACTIVE
  trigger status **66**
Adding a new table **53**
AFTER
  Trigger position **66**
An Introduction to Stored Procedure 27
Array 59, **78**
Ascending
  Index 68
Ascending index 68
Ascendng
  Index 68
AutoInc 54
AutoInc
  Field **54**
autoincrement 53

B

Backup
  database backup and restore 56
Backup database **107p.**
Basic SQL commands
  alias 33
  ALTER 31, 37
  CONTAINING 33
  CREATE 31, 37
  DELETE 31, 37
  DROP 31, 37
  EXECUTE 31
  EXISTS operator 36
  IN operator 36
  INNER JOIN 34
  INSERT 31, 36
  JOIN 34
    INNER JOIN34
    LEFT OUTER JOIN34
    RIGHT OUTER JOIN34
  ORDER BY33
  1. SELECT31
    Sub-SELECTs34
    UNION SELECT35
  SELECT across multiple tables 34
  SET 31
  SQL 31
  SQL Editor 31
  UPDATE 31, 36
  WHERE clause 33
BEFORE
  Trigger position **66**
BEFORE INSERT 94
BEGIN ... END 81
Benchmarks tests **177**
BIG INTEGER **75**
BIGINT 53
Binary Large OBject **73**
blob **73, 136**
Blob
  Segment size **74**
  Subtype **74**
Blob filter **67**
Blob Viewer/Editor. 73
Blobs **136**
Boolean **79**
BREAK **87**
Breakpoints 89
  Debug procedure 89p.
  Stored procedure 89

C

Careful mode
  Debug procedure **90**
CASCADE **55**
CASCADE
  Referential integrity **55**
change language
  IBExpert interface language 18
CHAR **74**
Character set 139
Charset **48**
CHECK constraint 59
CHECK OPTION **62**
Client Library **47**
CLOSE <cursor_name> 81
Collate **75**
Column 60
Command-line utilities **181**
Complex SELECTs**93**
Compound index **40**
CONNECT 51

Constraint **44,** 54, 59
Constraints 54
Constraints
  Table Editor 54
CONTAINING 91
context-sensitive menu 23
Context-sensitive right-click menu
  Database Designer **50**
Copy Database Object **141**
CREATE DATABASE **46,** 50
Create Database
  Local server **46**
  Remote server **46**
Create multiple CSV files **146**
CREATE PROCEDURE 81
CREATE SHADOW **112**
Create Table **53**
CREATE TRIGGER 81
Create View from Table **61**
Creating your first database 43
CURRENT_DATE 77
CURRENT_TIMESTAMP 77

D

Data Analysis 121
  Calculated Measures Manager **124**
  Columns 122
  Cube **122**
  Cube Structure **121**
  Data Analysis Cube Manager **123**
  Dimensions 122
  Main display area 122
  Measures 122
  OLAP **121**
Data Logging Triggers
  General Templates 103
Data model **43**
Data modeling **45**
Data type **73**
database 49
Database Documentation **163**
Database backup
  Backup Metadata only 108
  Backup Options **108**
  Convert to Tables 109
  Format 109
  Garbage collection 108, **110**
  Ignore transactions in Limbo 108
  Old metadata description 109
  Verbose 109
Database Backup & Restore **107**
database backup and restore 56
Database Comparer **142,** 158
database design 43
Database Designer **45, 49**
Database documentation
  Generate HTML documentation **165**
  Object and field descriptions **163**
  Print metadata **163**
  Template short cuts **163**
Database Explorer 50, 156
Database Explorer, the main navigator, considerably simplifying the work with InterBase/Firebird databases and database objects. 22
database model **49**
database normalization 43
database object **49**
Database objects **53, 59**
Database Properties 39
Database repair **138**
Database shadow **112**
Chapter 27: Database Statistics 171
  Garbage collector **171**
  MON$TRANSACTIONS table **172**
  Tables page 171
  Text page 171
Database trigger **65**
DATE **77**
DB1
  backup **107**
  demo database 21
Debugger 157
Debugging **89**
DECIMAL **76**
DECLARE <cursor_name> CURSOR FOR ... 81
DECLARE VARIABLE 81
DELETE 94, 143
DELETE
  Trigger type **66**
Descending
  Index 68
Descending index 68
Diagrams page
  Database Designer **50**
  DB Explorer **50**
Disable Performance Analysis **39**
DISTINCT 41
Domain **59**
Domain integrity **60**

Domains in stored procedures **95**
DOUBLE PRECISION **76**

E

Excel
- Export 127

Exception **67**
Executable
- Stored procedure **83**

Executable procedure **83**
EXECUTE BLOCK 40, **149**
EXECUTE IBEBLOCK 115, **155**
Execute multiple scripts **146**
EXECUTE PROCEDURE **64,** 81, **83,** 180
EXECUTE STATEMENT **85, 88,** 142, 155
EXIT 83
Export **51, 127**
Export data into script 129
Export data into script
- Export 129

Export formats **128**
Extract blobs **136**
Extract Metadata 133
- Data Tables **134**
- Meta Objects **133**
- Options page **133, 135**

F

Fast mode
- Debug procedure **90**

fb_inet_server.exe 183
fbclient.dll 13
fbguard.exe 183
fbserver.exe 183
fbsvcmgr 183
FETCH <cursor_name> INTO ... 81
Fields
- Description **163**

Find in Metadata **139**
Firebird
- Classic server 11p.
- download
  - Download Firebird,11
- Embedded 11
- Embedded Server 12
- Firebird Guardian 12
- Install
  - client-only install**14**
  - Firebird Installer12
  - Install Firebird**11, 15**
  - Linux**14**
  - Posix platforms**13**
  - Windows**14**
  - Windows platforms**13**
  - ZIP installation**14**
- Superserver 11p.

Firebird 2.0 blocks **149,** 156
Firebird Guardian 13, 15
Firebird Optimizer **178**
Firebird Zip kit **11**
firebird.conf 15, 39, 41, 47, 176
FirebirdServerDefaultInstance **16**
FIRST 85
FLOAT **76**
FOR EXECUTE ... DO ... **85**
FOR SELECT … DO 64
FOR select DO … 81
Foreign key **44,** 50, **54,** 68

G

Garbage collector
- Database Statistics 171

GBAK 183
GDS_LOCK_PRINT 184
GDS32.DLL 13
General Templates **103**
Generate HTML documentation
- CSS **165**
- Options page **165**
- Output page **165**

Generator **53, 66,** 137
GFIX 112, 183
Grant All to All
- Grant 170

Grant All with Grant option
- Grant Manager 170

Chapter 26: Grant Manager 169
- Edit Role **169**
- Grant All to All **170**
- Grant with Grant Authority **169**
- Granting access to stored procedures **170**
- Grants 169
- WITH GRANT OPTION **170**

Grant to All with Grant option
- Grant Manager 170

Grant with Grant Authority
- Grant Manager 169

Granting access to stored procedures
- Grant Manager 170

Grants page

Grant Manager 169
GROUP BY 41
GSEC 184
GSTAT 184

H

HK-Software Services Control Center **15, 119**
hkIM.exe **15**

I

IBEBlock 119, **155**
ibec_CreateReport **116**
ibec_ExportReport **117**
ibec_ExtractMetadata 136, 138
IBEScript 155
IBEScript.dll **182**
IBEScript.exe 143, 159, 181
Decrypt script 181
Encrypt script 181
IBEScriptDll Readme.txt **182**
IBExpert
Customer Version 17
download
Download IBExpert17
Install
Install IBExpert17
Linux
Install IBExpert18
setup_customer.exe 17
Trial Version 17
Windows
Install IBExpert17
IBExpertBackupRestore 114
HK-Software Services Control Center (SCC) **114**
IBExpertInstanceManager **13, 15**
IBExpertJobScheduler **119**
IBExpertNetworkMonitor 152
IBExpetInstanceManager
IpcName 16
RemotePipeName 16
RemoteServiceName 16
RemoteServicePort 16
RootDirectory 16
Service runtime info **16**
IBLOCKPR 184
IBMGR 184
IF condition THEN ... ELSE 81
Iindex statistics **178**
Import **129**
Importing Excel files
Import **130**
IN AUTONOMOUS TRANSACTION 142
Inactive
Trigger status **66**
Index 38, 54, **68,** 97
Index plan 91
Index selectivity **69**
Index statistics **38, 40, 69, 178**
Indices **68**
INSERT 94, 140, 143
INSERT
Trigger type**66**
INSERT ... RETURNING 86
INSERT INTO 151
2.2 INSERTEX 130
Import **130**
Install
Linux
Install Firebird under Linux20
Install IBExpert under Wine20
Wine tools18
Installing multiple Firebird servers **15**
INTEGER **75**
INTO 86
ISC4.GDB **47**
ISQL 184

J

JOIN 38, 40, 60, 93

L

LEAVE <label> 81
LEAVE and BREAK **87**
LIKE 41, 91
LOCALHOST **46**
Log Actions 103
Log Actions page
Log Manager **103**
Log Data
Log Manager 104
Log file **104**
Log Manager **103**
Logging **103**
Logs **40**

M

Membership page
User Manager 168
Memory configuration **176**
Metadata 56, **133,** 139
metadata

253 changes of table left 56
Model Navigator 50
Database Designer **50**
SQL Assistant **50**
Model Options **49p.**
Model Options
Database Designer **49,** 50
MON$ATTACHMENTS **172**
MON$ATTACHMENTS table
MON$SERVER_PID **172**
MON$TRANSACTIONS table
Database Statistics 172
Moving data between databases **141**

N

n:1
relationships **45**
n:m
Create database **46**
Data modelin **45**
Database Designer **45**
Relationships **45**
NBACKUP 183
NBAK 183
NEW and OLD context variables 66
Trigger **66**
New field 54
Field **54**
Next transaction 171
Database Statistics 171
NEXT VALUE FOR <generator> 81
NO ACTION **55**
NO ACTION
Referential integrity **55**
Non-select procedures **65**
NOT NULL 59, **80**
NULL 59, **80**
NUMERIC **76**

O

Obtain current generator values **137**
ODBC driver 155
ODBC Viewer **129**
Oldest active transaction (OAT) 171
Database Statistics **171**
Next transaction 171
online documentation **44**
OPEN <cursor_name> 81
Operating systems **175**
Optimizer 38, **40, 69**
Optimizing database performance 177
Optimizing database performance 175
Operating systems **175**
Optimal hard disk use **175**
Optimizing hardware configuration **176**
Optimizing OS configuration **177**
Optimizing procedures **91**
Optimizing SQL statements **40**
Options page
Log Manager 104
ORDER BY 40, 68

P

Page
Description page **163**
Page Size **47**
Parameters and Variables 90
Debug procedure **90**
Password 47
User Manager 168
Performance Analysis **39,** 41, 91
Performance analysis
Additional **39**
Graphical Summary **39**
PivotCubeForm
Data Analysis **121**
Plan Analysis 92
Plan Analyzer **38,** 41
Port 3050 **46**
Primary key **43, 53,** 62, 68p.
primary key
compound primary key 43
Naming conventions **43**
system object **44**
3. Print 51
Metadata **163**
print Metadata
Print Preview **165**
print previe
Print 51
print preview
Print 51
PROJECT 60
PSQL 62, 65, **81**

R

Read-only view **61**
Recompile all stored procedures and triggers **71,** 97
Recompute Selectivity 38, 70

Recompute selectivity of all indices **38, 70**
Recompute the selectivity of all indices 93
Recursions and modularity **88**
Recursive procedures **88**
Recursive procedures
  Stored procedure **88**
Referential integrity **55**
register a database
  Register 21
Relational database 60
relationship 50
Relationships 44
Report Designer
  ReportManager 115
Report Manager **115**
Reporting **115**
Restore
  database backup and restore 56
3. Restore database 110
  Client Library File 111
  Commit after each table 111
  Deactivate indexes 110
  Don't enforce validity conditions 111
  Metadata only 111
  Page size 111
  Replace existing database 111
  Schadow files
    Don't recreate shadow files 111
  Use all space 111
  Verbose 111
Reverse Engineer **49**
Reverse Engineer ...
  Database Designer **49**
Role **68**
Roles page
  User Manager 168
Row 60
run to cursor 89
Run to cursor
  Debug procedure 89
  Stored procedure 89

S

Script Executive 21, 136, 144
Script Executive & Script Language Extensions **145**
Script language extensions **147**
Search Metadata **139**
Search options **139**
security.fdb 16, **47**
security.fdb or security2.fdb **47**
security2.fdb 16, **47**
SELECT 39, 60, 138
SELECT … INTO : variable_list 81
Select procedures **64**
SELECT statement 61p.
Selectable
  Stored procedure **83**
Selectable procedure **83**
11. Selectable stored procedures 93
  Stored procedure **93**
Sequence **66**
Server-side programming 25
Service description **114**
SET DEFAULT **55**
SET DEFAULT
  Referential integrity **55**
SET NULL **55**
SET NULL
  Referential integrity **55**
SET STATISTICS **70,** 71
SET STATISTICS INDEX **179**
SET TERM **81**
SET TERM
  Stored procedure **81**
SMALL INTEGER **75**
SORT 38
SP/Triggers/Views Analyzer **97**
Special features 140
SQL Dialect **47**
SQL Editor 31, **140,** 155
SQL Editor
  code templates 38
SQL Performance **38**
SQL script 21
STARTING WITH 93
Statements
  Debug procedure 90
step into 89
Step into
  Debug procedure 89
  Stored procedure 89
Stored procedure 25, 39, **62**
Stored Procedure
  INITALL 28
  input parameters 27
  return values 27
Stored procedure and trigger debugger **84, 89**

stored procedure and trigger language **81**
SUSPEND **64, 83,** 93
SYSDBA **47**
system object **44**
System tables 68

T

Table **60**
Table Data Comparer **143**
Table Editor 54
Table trigger **66**
Template short cuts
  Keyboard templates 163
Templates 50
Templates
  Database Designer 50
Temporary files **176**
Test Data Generator **56**
Test Data Generator
  Create test data **56**
TIME **77**
TIMESTAMP **77**
Trace 89
  Debug procedure 89
  Stored procedure 89
Transaction control 155
Trigger **57,** 60, **65,** 67, **94**
Trigger
  Universal triggers 61
Trigger types **66**
TYPE OF COLUMN 142

U

UDF (User-Defined Function) **67**
UNION 60
Universal triggers 90
Universal triggers
  Trigger 90
Updatable view 60, **61**
UPDATE 94, 143
UPDATE
  Trigger type **66**
UPDATE ... RETURNING 86
User Database **156**
Chapter 25: User Manager 167
  Membership page **168**
  Password **168**
  Roles page **168**
  Trusted authentication 167
  Users page **167**
User-Defined Function (UDF) **67**
Users page
  User Manager 167
2. Using the GRANT AUTHORITY option 170
  Grant Manager 170
UTF8 165

V

VARCHAR **74**
Variable = Expression 81
3.1 View 60
  updatable view 60

W

Watches
  Debug procedure 90
WHERE 40, 93, 138
WHILE ... DO **87**
WHILE condition DO 81
Windows Task Manager **172**
WITH CHECK OPTION 61p.
Working with shadows **112**
Writing exceptions 101
  Exception **101**

1:1
  Relationships **44**

/

/* comments */ 81

AN INTRODUCTION TO IBEXPERT & FIREBIRD

[Copy from Foreword]

If you develop SQL databases, professionally or as a hobby, and need an efficient and powerful tool, you cannot go wrong with IBExpert. It 4enables you in just a short space of time to become acquainted with and achieve a command of the open source database, Firebird, as well as its commercial relative, InterBase. There are powerful and yet easy-to-learn editors for all vital functions. Development of database objects, database models, stored procedure and trigger programming, performance tuning -all th8is and much more can be executed simply and quickly using IBExpert.

The Firebird server is an extremely powerful open source database system,which in spite of its very simple installation and administration, offers all essential functions otherwi8se found only in commercial database systems such as Oracle or Informix. However, following installation the Firebird user has but a few command-line tools at his disposal, there is no powerful GUI tool for data definition and administration included in the kit. This, along with the very limited documentation, is the first hurdle that Firebird users need to overcome.

This is where IBExpert comes to the rescue, whether in the form of the free, functionally limited Personal Edition, the gratis Educational Version or the commercial Full Version including all modules, which is also available as a 45-day Trial Version. These resources enable the user the acquire the technical proficiency required for professional applications.

This guide is intended as an introduction for both database developers and administrators, enabling …

IB Expert

IBExpert KG

Im Gewerbepark 8
27798 Hude
Germany

Phone.: +49 (0) 4408 3593492
Fax: +49 (0) 4408 3593499
E-mail: info@ibexpert.com