THE EXPERT'S VOICE® IN OPEN SOURCE

Firebird 2 Supplement for The Firebird Book

A Reference for Database Developers

An essential guide for developers and administrators working with the Firebird open source relational database management system.

Helen Borrie

Supplement published by IBPhoenix Publications

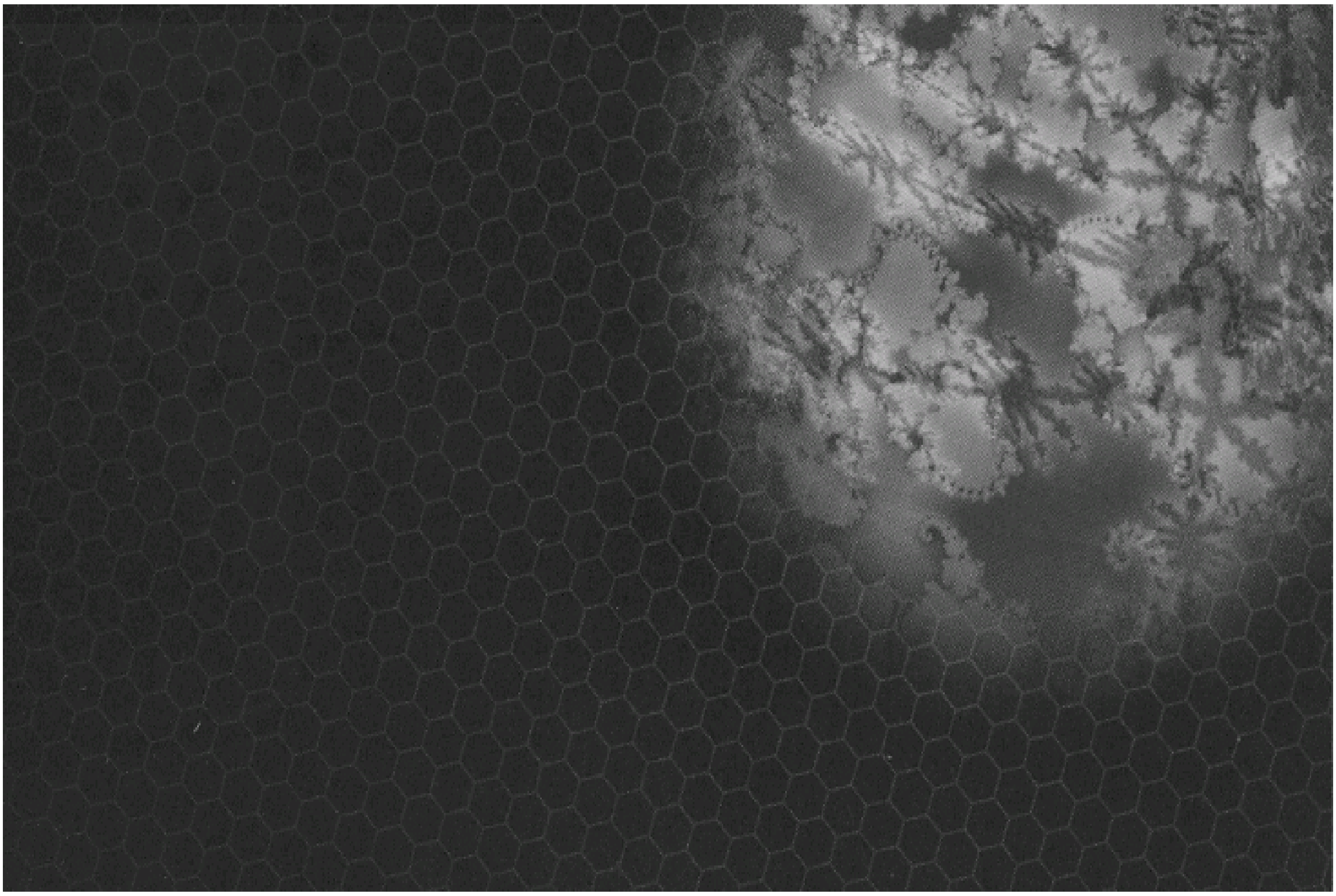
Original volume published by Apress®

Build 2.1.4 Updated September 2009 for Firebird 2.1.3 and 2.0.5

The Supplement will be updated periodically
and announced in the Firebird lists and at the IBPhoenix websites
http://www.ibphoenix.com

Copyright Notice

Table of Contents



Introduction


Read this
first !

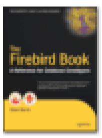
The Firebird 2 Supplement

IN THE YEARS SINCE PUBLICATION OF *THE FIREBIRD BOOK* and the release of Firebird 1.5, the Firebird RDBMS has made great strides. In December, 2006, Firebird 2.0 was released. Before a year was out, its third sub-release, v.2.0.3 was out, shortly followed by the second beta of V.2.1. Before Christmas 2007, a new sub-release, v.1.5.5, extended the support life of the immensely popular predecessor that is the focus of the original book.

While many internal enhancements will greet previous users in Firebird 2.0.x and 2.1, the original *Firebird Book* still holds true. Broader changes will come in Firebird 3, certainly more than enough to warrant a second edition of the book. For Firebird 2.x, this supplement should bring the user up-to-date with the new features and improvements that have been added since publication.

How This Document is Organised

The Supplement is available only in PDF format. Adobe Acrobat Reader is required to read and print it. It is laid out on A5 pages, to enable you to print out pages two-up on A4 paper. The A5 page size will fit between the pages of *The Firebird Book* without overlapping the edges of the book.



Chapter sequence and style follow that of the original book and, where relevant, page references are provided. In some cases, a chapter may be “empty”, as a placeholder for changes that may arise during the evolution of the 2.x sub-releases.

New versions and updates of some Appendices are included: *Error Codes* (App. X) and *Reserved Words* (App. XI). Updates are provided for Apps I (*External Function Summary*), VII (*Firebird Limits*) and VIII (*Character Sets and Collations*).

The Supplement will be updated periodically and announced in the Firebird lists and at the IBPhoenix websites. You may email editors@ibphoenix.com if you wish to get a free update for a copy of the Supplement that you have purchased. The current build version is 2.1.4, dated 13/09/2009.

Any compatibility issues with older Firebird versions are highlighted, with an explanation of any adjustments you will need to do in existing systems to make the feature work as designed.

Errata (mistakes, typos, etc.) that have shown up across several reprintings of *The Firebird Book* for each book chapter at the end of the corresponding Supplement chapter.

Don't be disappointed if some of the Errata described are absent in your copy of the original book. It all depends on when yours was printed!

The Firebird 2.1 series contains much that is new, even if you already migrated your Firebird 1.5 systems and databases to V.2. Wherever possible, this blue star icon will alert you not just to new features but also to changes and improvements affecting existing features.

Highlights of Firebird 2.x

Firebird brings with it a substantial number of enhancements to improve performance, security and support for international languages.

Several limitations are gone, including the old "252 bytes or less" limit on index size and several inhibiting factors in query optimization, especially for complex outer joins and DISTINCT queries. Calculation of index statistics has been revamped to improve the choices the optimizer has available.

This release also sees the completion of porting of the Services API to all models and platforms.

The new on-disk structure (ODS) designator for Firebird 2.0.x is 11 and for V.2.1 it is 11.1. Internally, Firebird 2.x can distinguish a Firebird database of ODS 11 and higher and an InterBase one, by way of a bit setting on database pages.

Although Firebird 2.0 will connect to databases having older ODS versions, most of the new features will not be available to them.

SQL Language

Many new additions have been made to the SQL language, including some useful new extensions in PSQL and, in V.2.1, the implementation of a large number of internal functions that were previously external functions (UDFs). Some important enforcements of standards will affect legacy application code in cases where developers exploited the tolerance of older implementations in InterBase and the Firebird 1 series faulty syntax. Details will be found in Parts Four (DDL), Five (DML) and Seven (PSQL). Language changes and additions to transaction parameters and capabilities are in Chapter 26.

For a summary of the highlights, see Chapter 19, *Firebird's SQL Language* 92.

Administration and Monitoring via SQL

Firebird 2 introduced some new context variables as well as functions and syntax to assist with getting and setting some custom context variables. In V.2.1 comes an entirely new set of facilities for monitoring databases, transactions and statements, along with syntax for extracting the information. The latter enables the long-awaited capability to stop problem transactions.

See Chapter 41, Database Monitoring 229

V.2.1 also brings database-level and transaction-level triggers, opening up effective ways to enhance run-time monitoring, logging, exception handling and troubleshooting.

See Chapter 31, Database Triggers 170

Reserved Words

A number of new reserved keywords were introduced in both V.2.0.x and V.2.1 and restrictions were changed on some existing ones. The full list is available in Firebird's CVS tree in /doc/sql.extentions/README.keywords.

See also see Appendix XI, *Reserved Words* 317, for a completely updated list of keywords.

You must ensure that your DSQL statements and procedure/trigger sources are not using the new keywords as identifiers.

In a Dialect 3 database, such identifiers can be redefined using the same words, as long as the identifiers are enclosed in double-quotes. In a Dialect 1 database there is no way to retain them: they must be redefined with new, legal words.

Security

The former **security.fdb** authentication database has been replaced by **security2.fdb**, which has been refactored to make it difficult for the table containing user accounts and passwords to be accessed directly. The former USERS table is now a view over the new table RDB$USERS. Amongst the benefits comes the ability for users to change their own passwords.

Firebird 2 uses a different, more secure method for password encryption. It also adds some extra, built-in security features to resist hostile attacks from the network.

For details, see Chapter 34, *Server Protection* 185.

Trusted Authentication on Windows

For Firebird 2.1 on Windows server platforms, the security status of operating system users with Administrator privileges (local or domain group) is respected by default and such users will be able to log in with empty Firebird user name and password credentials. Because this introduces a security vulnerability on networks where Windows security is not well controlled, the user authentication mode on Windows is configurable.

For details, see Chapter 34, Trusted Authentication on Windows 188

Tools

This release introduces the incremental backup tools NBak and NBackup 215. The command-line tools have all been tidied up and some have new switches and options available.

---

gbak -R Semantics have changed for the better

An important change has been done to prevent accidental database overwrites as the result of users mistakenly treating "-R" as an abbreviation for "restore". gbak -R was formerly a shortcut for "–REPLACE_DATABASE". Now the -R switch no longer restores a database by overwriting an existing one, but instead reports an error.

If you actually want the former behaviour, you have some alternatives—see Chapter 38, *Database Backup and Restore* 215.

For details of the changes and additions to the command-line tools, see the relevant chapters in Part Nine, *Tools* 206.

Network and System

Many improvements have been made to reduce system limitations and stabilise operations, especially on Windows. Some are highlighted here.

Rebuilt Local Protocol Support on Windows

Firebird 2.0 has replaced the former implementation of the local transport protocol (often referred to as IPC or IPServer) with a new one, named XNET. Besides being more stable, the new implementation can now be used to make local connections to the Classic model of Firebird and for connecting from a local terminal client.

For details, see Chapter 4, *Operating Basics* 26.

New Garbage Collection Options for Superserver

Superserver installations can now be configured to use "cooperative" garbage collection, either in combination with the background GC or instead of it. Unlike background GC, cooperative GC occurs whenever a user transaction starts and reads a table on which a transaction that is no longer "interesting "has obsolete versions present from committed updates or deletes.

For details, see Chapter 15, *Creating and Maintaining a Database* 78.

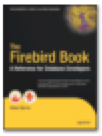
Additional Database Shutdown Modes

A new *state* parameter has been added to the **gfix** utility to enable single-user and full shutdown modes for the **`gfix -shutdown`** and **`gfix -online`** commands.

For details, see Chapter 39, *Housekeeping Tool (gfix)* 222.

Changes to Synchronisation Logic

Several improvements were made to the Lock Manager and elsewhere to reduce lock contention and improve the information provided in Lock Manager memory dumps.



For details of the mysteries of the Lock Manager subsystem, see Chapter 40, *Understanding the Lock Manager*.

64-bit Platform Support

Firebird 2.0 Classic and Superserver for Linux and FreeBSD support the AMD 64-bit platform and should support Intel EM64T also. The Intel IA-64 platform is not supported yet. MacOSX Intel builds on both x64 and x86 platforms are now part of the mainstream release cycle for Firebird 2.1 and higher, along with PPC and x86 release candidates for Firebird 2.0.4. As this supplement was being updated, experimental x64 builds for Solaris were being released for testing.

64-bit Windows platform (AMD64 and Intel EM64T) ports of Classic, Superserver and Embedded models becomes available for the first time with Firebird 2.1..

Some Old Limits Have Gone

Table Size

The previous table size limit of ~ 30 Gb has been removed by the introduction of 40-bit (internally, 64-bit) record enumerators.

For details, see Appendix VII , *Firebird Limits* 273.

Index Size

The old 252-byte limit on the total length of a Firebird index has gone, replaced by a new mechanism that allows indexes of up to 25 per cent of the page size.

For details, see Chapter 18, *Indexes* 84, and Appendix VII , *Firebird Limits* 273.

Debugging Improvements

A number of improvements have been made to assist with debugging the software.

For details, see Appendix XIII , *Miscellaneous* 328.

Registry Search on Win32 Servers is Gone

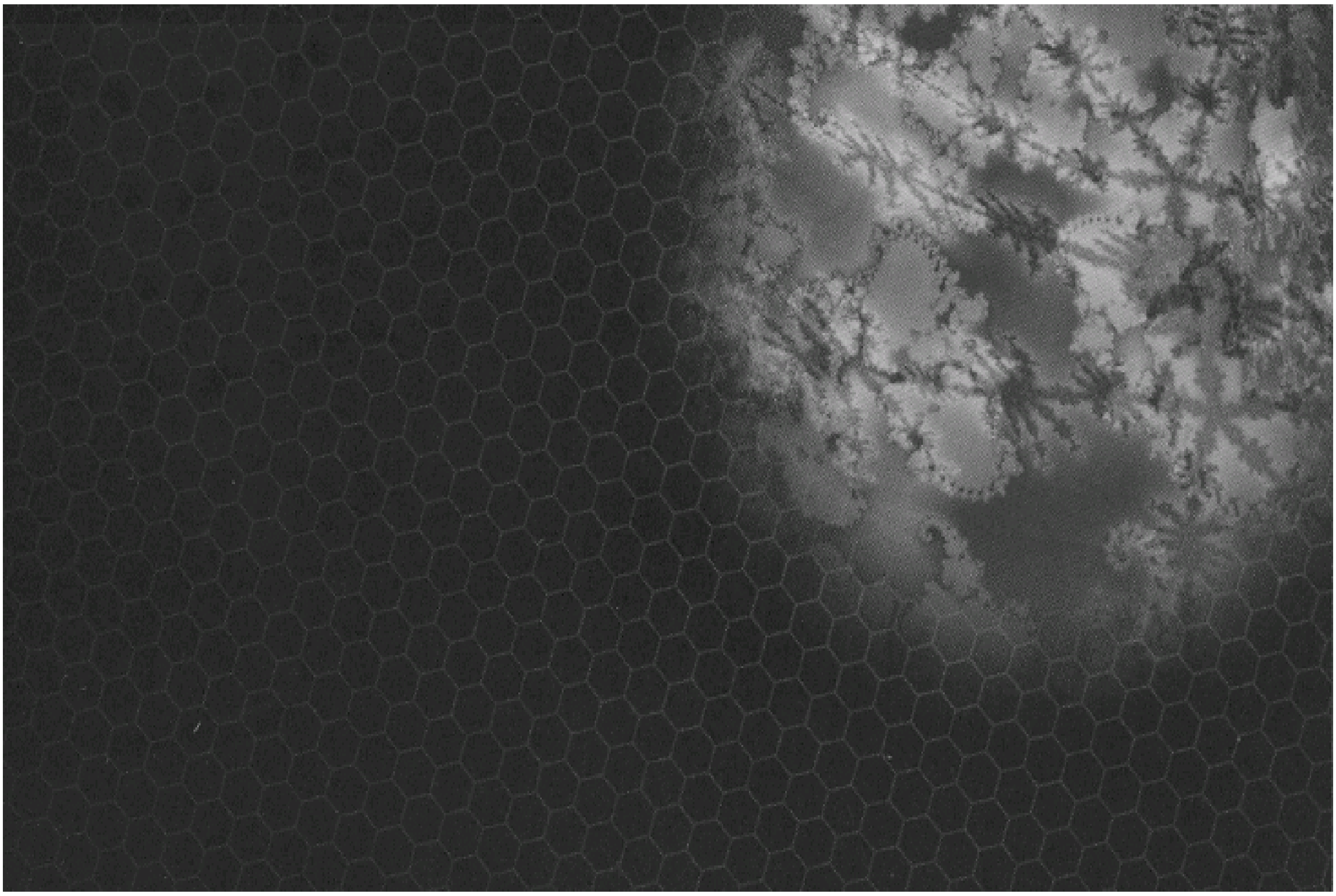
The root directory lookup path has changed so that server processes on Windows no longer use the Registry. However, Registry checking is still used by the command-line utilities.

For details, see Chapter 3 , *Configuring Firebird* 25.

Refer to Page 50, The *Firebird Root Directory*, to understand what this change means.

V.2.0.5 and V.2.1.2 Connection Parameter Restrictions

The V.2.0.5 and V.2.1.2 sub-releases closed a long-standing bug in the API that allowed ordinary users to alter the attribute settings of databases, such as setting Forced Writes off and on or changing the size of the page cache. The fix will affect any existing applications that used non-default values for these parameters. The effects are detailed in Chapter 7, *Firebird Clients* 38.



Part One

Boot Camp

Topics in This Part

Chapter 1
Installation

Treat this chapter as an extension of Chapter 1 in *The Firebird Book*. If you are moving to Firebird 2.0.1 directly from Firebird 1.5.4 or lower versions, it will provide a guide for upgrading your system safely.

Compatibility Issues

Naturally, with so much bug-removal and closing of holes, there are sure to be things that worked before, in your applications and perhaps in stored procedures, that will no longer work under Firebird 2. Scan through this supplement and note the items that are highlighted as compatibility issues. You might well need to attend to some of these issues before you begin.

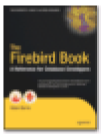
If you already have an earlier version of Firebird or InterBase® on your server and you think you might want to go back to it, set up your fall-back position before you begin.

Back up

When you uninstall Firebird 1.5, certain configuration files in the installation directory will be preserved if you run the installer and OVERWRITTEN if you decompress a zip kit into the default location. The files are

security.fdb	firebird.log	firebird.conf	aliases.conf

Nevertheless, it would be a good idea to keep safe copies of these files if you want to port your existing settings to your new Firebird 2.0 installation.



If it has been a while since you did a Firebird installation, you should review Chapter 1 before you proceed.

If you are upgrading from Firebird 1.0.x, go to Chapter 36, *Configuration and Special Features* , for details of the correlation between settings in *ibconfig* and *firebird.conf*. Study the notes about *firebird.conf* to work out what can be copied directly and what parameters require new syntax.

If you plan to have multiple versions installed, download a copy of the Firebird 1.5.5 release notes from the Firebird website and study Chapter 9, *Configuring the Port Service on Client and Server*. These older notes are also deployed in the /doc directory, along with the Firebird 2.x release notes, once the installation is done.

The on-disk structure (ODS) of the databases created under Firebird has changed. Although the Firebird

2.0 and 2.1 servers will connect to databases having older ODS versions, most of the new features will not be available to them. It is your choice whether you upgrade the databases now or wait until you have run some tests on the new server.

Before you uninstall the older Firebird server and set out to install Firebird 2.0, first make transportable *gbak* backups of your existing databases—including your old **`security.fdb`** or (even older) **`isc4.gdb`** security databases.

Then, before you proceed, restore these backups into a temporary location, using the old *gbak*, and verify that the backups are good by connecting to them with the old *isql* or your favourite admin tool. Place your tested backups in a safe location on the filesystem.

If you don't plan to upgrade your user database[s] immediately, you might like to zip up the originals and move them to offline storage. The freshly restored ones can then be kept for online use after you have installed Firebird 2.

In any case, you will need the transportable *gbak* backup of the security database. After installing Firebird 2, you will use the Firebird 2 *gbak* to restore it as an ODS 11 database, in order to run it through an upgrade script to make it compatible with the new security features.

Restore the security database with a page size of **at least 4096 bytes**! A smaller page size will not work.

Migrating Databases to V.2.1

For the migration to Firebird 2.1, you should plan to do the backup/restore operation, even with your ODS 11 user databases, to bring them up to ODS 11.1 and access the new metadata and administrative capabilities. However, the progressive refactoring of international language handling means that, if you used a non-ASCII character set anywhere in setting up metadata (comments, expressions, check constraints, etc.) you will have more to do with your converted ODS 11.1 user databases before they can be accessed coherently.

For upgrading metadata text, see Chapter 11 65.

Disable the FIREBIRD Variable

Check to make sure that there is no FIREBIRD environment variable defined that is visible to administrator-level users or to the **`localsystem`** user (on Windows) or the **`firebird`** user (on Linux)—see the section entitled *The FIREBIRD Variable* 25 in Chapter 3.

Before you start installing or uninstalling anything, **shut down all databases and stop the server**.

Default Disk Locations

You cannot install Firebird 2.x over the top of an existing v.1.5 installation. However, there are ways to have multiple versions of the server installed on the same host.

Windows

The default location that the executable installer for Windows will use for V.2.0.x Classic or Superserver installs is:

```
c:\Program Files\Firebird\Firebird_2_0\
```

For V.2.1 installs it is:

```
c:\Program Files\Firebird\Firebird_2_1\
```

On Windows, you have three server models to choose from: Superserver, Classic and Embedded Server. This means you have some decisions to make before installing Firebird 2.0. The Superserver and Classic models, as well as server tools-only and client-only, can be installed using the Windows installer application. For a full-release install, it is highly recommended to use the installer if there is one available.

Make sure you are logged in as Administrator (not applicable to Win9x or ME).

If you have a previous version of the Firebird Server that you want to keep available for use, the installer will honor it and not destroy any pieces of the existing server. However, if your server is set up with the client library for the older version (*fbclient.dll* or *gds32.dll*) in the system path, e.g. in the system32 directory, you should rename the DLL before you start installing Firebird 2.x.

Microsoft Runtime Libraries

Both the server and the client components need the C and C++ runtimes to be present in the system path in order to run. If either is not already present on your system, there are copies in Firebird's \bin\ directory (*msvcr71.dll* and *msvcp71.dll* for Firebird 2.0.x, *msvcr80.dll* and *msvcp80.dll* for V.2.1) that you can transfer to the `%system%` folder. Alternatively, you may prefer to download and install the redistributable *vcredist.exe* runtime kit for your platform from the Microsoft site.

Vista

Under the new restrictions about installing dynamic libraries in Vista, merely copying the runtimes into the system folder won't work—you must run the *vcredist.exe* installer in order to register them for use by applications and other libraries.

64-bit Windows

Use the 64-bit installer kit if you are installing Firebird 2.1 on a 64-bit version of Windows. It includes a 32-bit client kit to ensure that any 32-bit applications you are running on that server will work transparently with the 64-bit engine.

The zip kits are platform-specific, so be sure to use the right one. If you need to run 32-bit applications, you will need to take care of installing the 32-bit client library and the Microsoft 32-bit C runtime library yourself.

Linux

For the Linux RPM packages, the default location remains as for previous releases:

```
/opt/firebird/
```

If you plan to retain more than one version of Firebird and the older version was installed using **`rpm -install`** then it is recommended that you uninstall it using **`rpm -e`** and reinstall it later using the tarball kit.

Alternatively, you might prefer to retain the current installation and use the tarball kit to install Firebird 2 or 2.1 into an alternative location.

The official Firebird rpm packages do not support the -U (upgrade existing package) option. Some distribution-specific Firebird packages do, however.

Threading Models for Superserver on 32-bit Linux

The NPTL builds for 32-bit Linux platforms have been available since Firebird 1.5, supporting the "New POSIX Threading Library" that was beginning to be introduced in some late v.2.4 kernel distros. All v.2.6 kernel distros theoretically support NPTL which, as a rule, should make the NPTL build your first choice if you plan to run a Superserver. The "non-NPTL" Superserver build should always be chosen if your distro has the 2.4 kernel.

However, due to ongoing problems with the NPTL implementation in the Linux kernel, it can happen that some distros with the v.2.6 kernel do not implement NPTL. You can test for it as follows.-

```
]$ getconf GNU_LIBPTHREAD_VERSION
```

If it doesn't return something like "NPTL 2.n.n" but something like "linuxthreads-0.nn" then you must use the kit for the old threading model.

Superserver on AMD64 Opteron Systems

There is a recognised threading issue for some AMD Opteron systems where a 64-bit Linux is the operating system platform. Threaded applications exhibit frequent segmentation faults. For Superserver, this translates to unstable attachments and, sometimes, server crashing. If your Linux/AMD Opteron system displays this problem with unacceptable frequency, you should consider testing a different distro, reverting to a v.2.4 version or disabling NPTL and exporting **`LD_ASSUME_KERNEL=2.2.5`**, as described on Page 11 of *The Firebird Book*.

Installing

When all the preparations are complete, install Firebird 2 according to the instructions in *The Firebird Book* or the release notes. On completion, the server will be running with a security database named **`security2.fdb`**.

If you are upgrading, this copy of the security database is temporary. If you try to replace it directly with a pre-v2.0 Firebird security database and then try to connect to the server, you will get the message "Cannot attach to password database". It is not a bug: it is by design. A security database from an earlier Firebird version cannot be used directly in Firebird 2.0 or higher.

On Windows, the temporary SYSDBA password is **`masterke`**.

On Linux, you will find a generated temporary password in the Firebird root directory in a text file named **`SYSDBA.password`**. Type **`cat SYSDBA.password`** to read it.

If you are upgrading from a Firebird 1.5.x or older installation, you have a little more work to do before you can use Firebird 2/2.1 to access your databases using the existing login accounts from **`security.fdb`** or **`isc4.gdb`**.

Upgrading Your Security Database

In order to be able to use an old security database accounts, it is necessary to run the upgrade script **`security_database.sql`**, that is in the **`../upgrade`** sub-directory of your Firebird server installation for convenience.

The script also appears in the *Appendix XIII* 329 of this supplement and in the Appendix to the Firebird 2 release notes.

You will also find a readme file named ***`security_database.txt`*** in the */upgrade* directory beneath the root directory of your installation.

A simple 'cp security.fdb security2.fdb' will make it impossible to attach to the firebird server!

---

1. Make certain that, when you restore the backup of your old database, that you make the *page_size* at least 4 Kb.
2. In pre-2.0 versions of Firebird it was possible to have a user with NULL password. From v.2.0 onward, the RDB$PASSWD field in the security database is constrained as NOT NULL.

   However, to avoid exceptions during the upgrade process, the field is created as nullable by the upgrade script. If you are really sure you have no empty passwords in the security database, you may modify the script yourself. For example, you may edit the line:

```
RDB$PASSWD  RDB$PASSWD,
```

to be

```
RDB$PASSWD RDB$PASSWD NOT NULL,
```

To do the upgrade, follow these steps:

1. Retrieve the transportable backup of your old security database that you made earlier and restore it in some separate directory using the Firebird 2 gbak executable. You can give the restored database any name you like.

   You will now have an ODS 11 version of your old security database, which will enable the script to run successfully over it.

2. Using *isql*, connect to the restored security database as SYSDBA and the temporary password and run the script.
3. Stop the Firebird service.
4. Copy the upgraded database to the Firebird 2 home directory as **`security2.fdb`**.
5. Restart Firebird.

You should now be able to connect to the Firebird 2 server using your old logins and passwords.

You have not quite finished yet.

Update Passwords!

A firebird.conf parameter, ***LegacyHash*** 198, enables initial use of the legacy DES hash passwords to access the new structure and change passwords. By default, ***LegacyHash*** is set to 1. As long as it stays at this value, Firebird's security does not work completely. To set this right, it is necessary to do the following steps:

i. Change the SYSDBA password

ii. Have the users change their passwords (from V.2.0, each user can change his or her own password).

iii. Set LegacyHash back to default value of 0, or comment it out.

iv. Stop and restart Firebird for the configuration change to take effect.

Chapter 1 Errata

Page	Erratum
11	The following text/code is incorrect: 2. You need.... add `LD_ASSUME_KERNEL=2.25` It should read: add `LD_ASSUME_KERNEL=2.2.5`
13	In the sidebar marked CAUTION near the bottom of the page, the phrase "IPX/SX networks "should be "IPX/SPX networks "to be exactly correct.
17	Under the heading "Windows 9x, ME, and XP Home "the first sentence states "Windows 9x, ME, and XP Home do not support services." While true for Windows 9x and ME, it is not true of XP Home. 1. Change the heading to "Windows 9x and ME" 2. Change the text to "Windows 9x and ME do not support services."

Chapter 2
Network Setup

This chapter supplements the material in Chapter 2 of *The Firebird Book*.

Windows Networking

Local Connection Protocol

The transport internals for the so-called "Windows local protocol" have been reimplemented using a subsystem known as XNET instead of the IPServer subsystem of previous Firebird versions.

The Firebird 2.x client libraries *fbclient.dll* and *fbembed.dll* are therefore incompatible with older servers with regard to local protocol and the previous client libraries are incompatible with the Firebird 2 servers in this respect.

If you need to use the local protocol, please ensure your server and client binaries have exactly the same version numbers.

The value of the IPCName parameter has changed from its v.1.5 value (FirebirdIPI). It is now FIREBIRD.

More information about the new XNET implementation can be found in Chapter 4, *Operating Basics* 26.

Server Multi-hop Capability

Historically, InterBase supported remote server redirection, commonly known as "server multi-hop". It was broken in the original code on which Firebird was built and was restored in Firebird during Firebird 2 development.

About Multi-hop

Multi-hop allows you to make a relayed TCP/IP connection to a Firebird server through a chain of one or more other Firebird servers, using a connection string with the hostnames stacked in order, as in the following Windows example:

```
alpha:beta:gamma:delta:C:\privatedata\mydata.fdb
```

The last host in the list (**`delta`** in the example) is the one that opens the database. The other hosts act as intermediate gateways on the gds_db service port (conventionally, port 3050).

Initially, when working, this feature was available unconditionally. Now, it is turned OFF by default and must be explicitly configured in the *firebird.conf* parameter ***Redirection*** 198.

You should not enable server multi-hop unless you really understand its security implications.

For details of why, see the topic *Vulnerabilities* 192 in Chapter 34, *Server Protection*.

Chapter 2 Errata

Page **Erratum**

33 The following text/code is incorrect in the third bullet under "Locating the HOSTS file":

- o On Windows 95/98/ME/XP/Server2003, the HOSTS file is located in C:\Windows

It should read:

- o On Windows XP/Server2003, the HOSTS file is located in C:\Windows**system32**
.

- **On Windows 95/98/ME, look in c:\Windows**.

Chapter 3
Configuring Firebird

In this chapter, as in the corresponding chapter of *The Firebird Book*, only the essentials of configuration are mentioned. Configuration is discussed in depth in Chapter 36, *Configuration and Special Features*.

Chapter 36 196 of this supplement provides details of configuration parameters that have been added, changed or deprecated since Firebird 1.5.

The FIREBIRD Variable

FIREBIRD is an optional environment variable that provides a system-level pointer to the root directory of the Firebird installation. If it exists, it is available everywhere in the scope for which the variable was defined.

The FIREBIRD variable is NOT removed by scripted uninstalls and it is not updated by the installer scripts. If you leave it defined to point to the root directory of a v.1.5.x installation, there will be situations where the Firebird engine, command-line tools, cron scripts, batch files, installers, etc., will not work as expected.

If the Windows installer program finds a value for %FIREBIRD% it will make that path the default location that it offers, instead of c:\Program Files\Firebird\Firebird_2_0 .

Unless you are very clear about the effects of having a wrong value in this variable, you should remove or update it before you begin installing Firebird 2.0. After doing so, you should also check that the old value is no longer visible in the workspace where you are installing Firebird—use the SET FIREBIRD command in a Windows shell or printenv FIREBIRD in a POSIX shell.

For details about setting and changing environment variables, refer to Chapter 3, *Configuring Firebird*, at pages 44-46.

Chapter 4
Operating Basics

Operating basics have changed little in Firebird 2.x. In this chapter are details of the main changes that could have an immediate effect on working with a migrated server.

Windows-based Servers

A number of improvements and changes will affect operational aspects of running Firebird 2.x servers on Windows.

Reimplemented Local Protocol—XNET

Firebird 2.0 has replaced the former implementation of the local transport protocol (often referred to as IPC or IPServer) with a new one, named XNET.

It serves exactly the same goal, to provide an efficient way to connect to server located on the same machine as the connecting client without a remote node name in the connection string. The new implementation is different and addresses the known issues with the old protocol.

Like the old IPServer implementation, the XNET implementation uses shared memory for inter-process communication. However, XNET eliminates the use of window messages to deliver attachment requests and it also implements a different synchronization logic.

Benefits of the XNET Protocol over IPServer

Besides providing a more robust protocol for local clients, the XNET protocol brings some notable benefits:

- it works with Classic Server
- it works for non-interactive services and terminal sessions
- it eliminates lockups when a number of simultaneous connections are attempted

Performance

The XNET implementation should be similar to the old IPServer implementation, although XNET is expected to be slightly faster.

Disadvantages

The one disadvantage is that the XNET and IPServer implementations are not compatible with each other. This makes it essential that your fbclient.dll version should match the version of the server binaries you are using (fbserver.exe or fb_inet_server.exe) exactly. It will not be possible to establish a local connection if this detail is overlooked. (A TCP localhost loopback connection via an ill-matched client will still do the trick, of course).

If you are using the Remote Desktop client to access the server locally from a non-privileged user account, you may find you cannot make a local connection if your client library is earlier than V.2.1.3. To make it possible, open firebird.conf (in your Firebird root directory) and find the parameter **IPCName**. Its default value appears like this:

```
#IpcName = FIREBIRD
```

Delete the comment marker (#) and change the value to the following:

```
IpcName = Global\FIREBIRD
```

Save the file and then stop and restart the server to have the parameter take effect.

Note the change in the IpcName parameter to FIREBIRD—in v.1.5 it was FirebirdIPI.

Change to WNET ("NetBEUI") Protocol

Previously, remote requests via WNET (a.k.a. NetBEUI, Named Pipes) were performed in the context of the *client security token*. Since the server serves every connection according to its client security credentials, it meant that a Firebird client instance running in an NT user domain would acquire the permissions to access the physical database file, UDF libraries, etc., on the server appropriate to that NT user's domain. That situation—known as *client impersonation*—conflicts with what is generally regarded as proper protection for databases and executable server modules in a client-server arrangement.

The WNET protocol no longer performs client impersonation. In Firebird 2.0 and higher, WNET connections are now truly client-server and, like TCP/IP, make no presumptions with regard to the rights of operating system users.

This change will affect applications that took advantage of client impersonation to enable client applications running with NT Administrator permissions to access security and other server-based functions without a SYSDBA login.

Trusted Authentication

For Firebird 2.1 on Windows server platforms, trusted operating system users will be able to log in with empty Firebird user name and password credentials. Because it is enabled by default in the v.2.1 binary kits, deploying Firebird 2.1 blindly to networks where Windows security is not well controlled might not be secure. However, the user authentication mode on Windows is configurable.

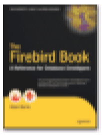
For details, see Chapter 34, Trusted Authentication on Windows 188

Improvements to instsvc.exe

The executable *instsvc.exe* is a utility for installing and uninstalling the Firebird service.

New -i[nteractive] Switch

The optional switch **-i[nteractive]** has been implemented to enable an interactive mode for services running under the domain of the **localsystem** user.



For existing documentation about *instsvc.exe*, see *Using the instsvc Utility* on Page 60 of Chapter 4, *Operating Basics*. For detailed usage instructions, refer to the document **README.instsvc** in the **doc** directory of your Firebird installation.

For v.1.5, interactive mode was required (as *Allow service to interact with desktop*) to run the local IPC protocol, which used a Windows message to connect the server.

The re-implemented local protocol in v.2.0 no longer requires interactive mode and the server itself has no need for it. However, some custom UDFs may use the Windows messaging facilities and this option allows them to work as expected.

Use of *instsvc.exe* is not applicable to Windows platforms that do not support services, viz.,Win9x, WinME.

New -n[ame] Switch

A -n[ame] switch has been added to enable the installation and management of alternatively-named instances of Firebird 2.1. This enables multiple Firebird 2.1 servers, appropriately configured, to be installed and managed on the same host machine.

For example, to install a Firebird 2.1 Superserver service named *FirebirdServer21*:

```
instsvc inst -s -auto -n FirebirdServer21
```

Once it is installed and running, to stop it:

```
instsvc stop -n FirebirdServer21
```

To (re)start it:

```
instsvc start -n FirebirdServer21
```

And to uninstall it, should you need to, first stop it, then:

```
instsvc remove -n FirebirdServer21
```

The instsvc.exe program does not install any registry keys and instreg.exe has not (so far) been made multi-instance-aware. If you have applications that rely on being able to find the Firebird root from the Registry, you can add a new *Instance* value manually to the key

```
HKLM\SOFTWARE\Firebird Project\Firebird Server\Instances
```

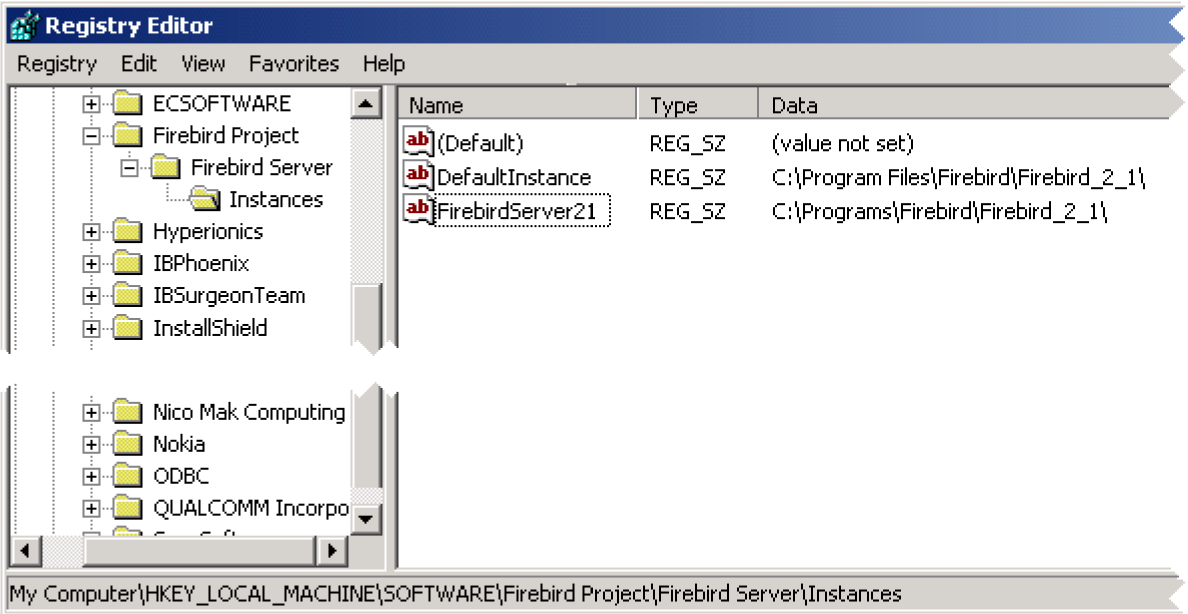
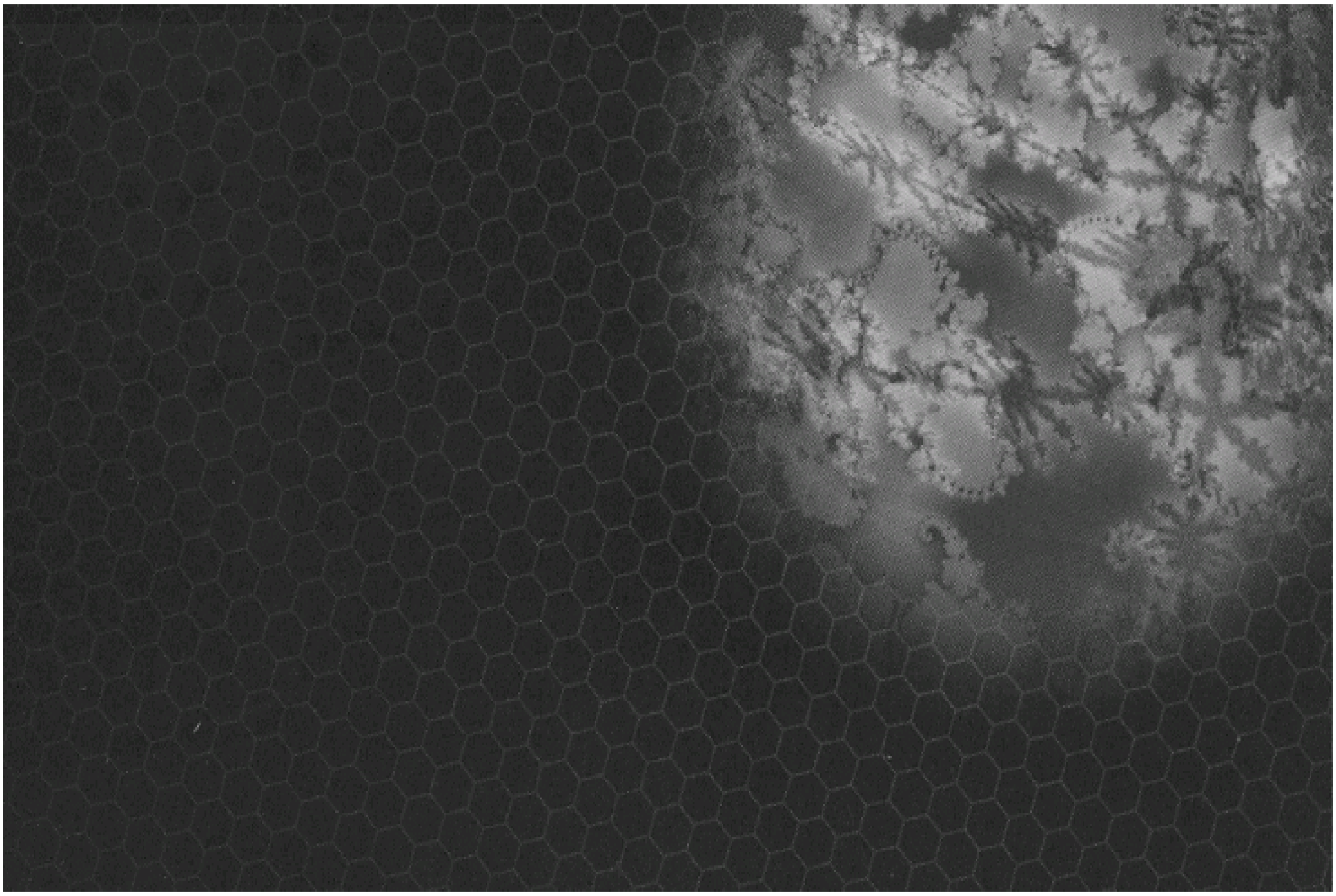


Fig. 4.1 Adding a Custom Service Instance in the Windows Registry



Part Two

Client/Server

Topics in This Part

Chapter 5 Introduction to Client/Server Architecture

No supplementary information.

Chapter 6
Firebird Server

No supplementary information.

Chapter 7
Firebird Clients

The principles of working with Firebird clients have not changed. This chapter of the supplement details changes that might affect the way you develop and deploy your applications programs. It does not provide exhaustive coverage of the subject: it should be taken in along with the many other changes described in other sections.

Miscellaneous Improvements

The remote protocol has been slightly improved to perform better in slow networks, once drivers are updated to utilise the changes. Testing showed that API round trips were reduced by about 50 percent, resulting in about 40 per cent fewer TCP round trips.

Smarter DSQL Error Reporting

Dynamic SQL (DSQL) is the translation layer between the SQL statements as presented by a client through the API and the processing layers beneath. Both data definition (DDL) and data manipulation (DML) statements that pass across the API are DSQL. When procedural (PSQL) modules pass statements to the engine, they act as dynamic clients in this respect and their statements pass through the DSQL layer.

The DSQL parser will now try to report the line and column number of an incomplete statement.

Changes to the Firebird API

Although the Firebird API was not specifically covered in the original *Firebird Book*, Firebird 2 brings some changes to the API that will be important to interface driver developers and direct-to-API programmers.

ibase.h

The API header file, *ibase.h*, has been the subject of a cleanup, with the result that public headers no longer contain private declarations.

Firebird API Version

The macro definition *FB_API_VER* is added to ibase.h to indicate the current API version. The number corresponds to the appropriate Firebird version. Thus, for example, the value of *FB_API_VER* in the v.2.0 builds is is 20 (the two-digit equivalent of 2.0) and for v.2.1 it is 21. Client application code can use this macro to check the version of *ibase.h* the application is being compiled with.

New API Functions and Parameters

The following functions and parameters have been added.

Relation Aliases

isc_dsql_sql_info()

The function call *isc_dsql_sql_info()* has been extended to enable relation aliases to be retrieved, if required.

XSQLVAR Change to sqlsubtype

When the character set of a CHAR or VARCHAR column is anything but NONE or OCTETS and the attachment character set is not NONE, the *sqlsubtype* member of an XSQLVAR pertaining to that column now contains the ID of the attachment (*connection*) character set instead of the ID of the *column's* character set.

New Function for Delivering Error Text

fb_interpret()

The new function *fb_interpret()* replaces the former *isc_interprete()* for extracting the text for a Firebird error message from the error status vector into a client buffer.

isc_interprete() is vulnerable to overruns and is deprecated as unsafe. The new function should be used instead.

Correction to Events Callback Routine

The new prototype for isc_callback reflects the actual callback signature. Formerly, it was:

```
typedef void (* isc_callback) ();
ISC_STATUS isc_que_events(
ISC_STATUS *, isc_db_handle *, ISC_LONG *, short,
char *, isc_callback, void *);
```

isc_event_callback()

In the Firebird 2.0 API it is:

```
typedef void (*ISC_EVENT_CALLBACK)
(void*, ISC_USHORT, const ISC_UCHAR*);
ISC_STATUS isc_que_events(
ISC_STATUS*, isc_db_handle*, ISC_LONG*, short,
const ISC_SCHAR*, ISC_EVENT_CALLBACK, void*);
```

isc_event_callback() may cause a compile-time incompatibility, as older event handling programs cannot be compiled if they use a slightly different signature for a callback routine, e.g., void* instead of const char* as the last parameter.

Lock Timeout for WAIT Transactions

isc_lock_timeout

The new feature extends the WAIT mode by making provision to set a finite time interval to wait for the concurrent transactions. If the timeout has passed, an error (*isc_lock_timeout*) is reported.

isc_tpb_lock_timeout

Timeout intervals can now be specified per transaction, using the new TPB constant *isc_tpb_lock_timeout* in the API.

The DSQL equivalent is implemented via the LOCK TIMEOUT <value> clause of the SET TRANSACTION statement.

Transaction Info

The following items have been added to the *isc_transaction_info()* function call structure:

isc_info_tra_oldest_interesting

Returns the number of the oldest [interesting] transaction when the current transaction started. For snapshot transactions, this is also the number of the oldest transaction in the private copy of the transaction inventory page (TIP).

isc_info_tra_oldest_active

- For a read-committed transaction, returns the number of the transaction that is currently the oldest active one
- For all other transactions, returns the number of the oldest active transaction when the current transaction started.

isc_info_tra_oldest_snapshot

Returns the number of the lowest *tra_oldest_active* of all transactions that were active when the current transaction started.

This value is used as the threshold (“high-water mark”) for garbage collection.

isc_info_tra_isolation

Returns the isolation level of the current transaction. The format of the returned clumplets is:

```
isc_info_tra_isolation,
1, isc_info_tra_consistency | isc_info_tra_concurrency |
2, isc_info_tra_read_committed,
isc_info_tra_no_rec_version | isc_info_tra_rec_version
```

That is, for Read Committed transactions, two items are returned (isolation level and record versioning policy) while, for other transactions, one item is returned (isolation level).

isc_info_tra_access

Returns the access mode (read-only or read-write) of the current transaction. The format of the returned clumplets is:

```
isc_info_tra_access, 1, isc_info_tra_readonly | isc_info_tra_readwrite
```

isc_info_tra_lock_timeout

Returns the lock timeout set for the current transaction.

BLOBs

isc_blob_lookup_desc()

isc_blob_lookup_desc() can now describe BLOBs that are output from stored procedures

Database Info

The following items have been added to the *isc_database_info()* function call structure:

isc_info_active_tran_count

Returns the number of transactions that the client currently has active.

isc_info_creation_date

Returns the date and time when the database was [re]created.

To decode the returned value, call *isc_vax_integer* twice to extract (first) the date and (second) the time portions of the context variable **ISC_TIMESTAMP**. Then, use *isc_decode_timestamp()* as usual.

Services API Improvements

The Services API is now fully implemented for all models of Firebird 2.x. Service Manager task execution has been optimized. Services are now executed as threads rather than processes on some threadable Classic builds (currently 32-bit Windows and Solaris).

New *fbsvcmgr* Command-line Utility

If you have ever been frustrated in the past by not being able to access the Services Manager calls except by getting someone to write a client program for you, then the new *fbsvcmgr* command-line utility may be just the missing piece you have been waiting for. For details, see Chapter 39, Housekeeping Tools 222.

New Services API Items

Two new items that were added to the Services API in Firebird 2.1 are:

- *isc_spb_trusted_auth* applies only to Windows. It forces Firebird to use Windows trusted authentication. For more information about using Windows trusted authentication when accessing Firebird 2.1 and higher servers, see the topic Windows Trusted User Authentication 188 in Chapter 34..
- *isc_spb_dbname* gives the ability to set a database name parameter in all service actions related to accessing the security database from a remote client. It is equivalent to supplying the -*database* switch to the *gsec* utility.

Restrictions for Non-Privileged Clients

Non-SYSDBA access to parts of the Services API that return information about users and database paths has been disabled. A non-privileged user can retrieve information about itself, however.

Database Parameters

and

A long-standing, legacy loophole in the handling of connection parameters enabled ordinary users to make connection settings that could lead to database corruptions or give them access to SYSDBA-only operations. Closing that loophole could affect existing applications, database tools and connectivity layers (drivers, components) that attempt to set them. For example, a Delphi application that included 'RESERVE PAGE SPACE=TRUE' and 'FORCED WRITES=TRUE' in its database Params property will now reject a connection by a non-SYSDBA user with ISC ERROR CODE 335544788, “Unable to perform operation. You must be either SYSDBA or owner of the database.”

The affected parameters are.-

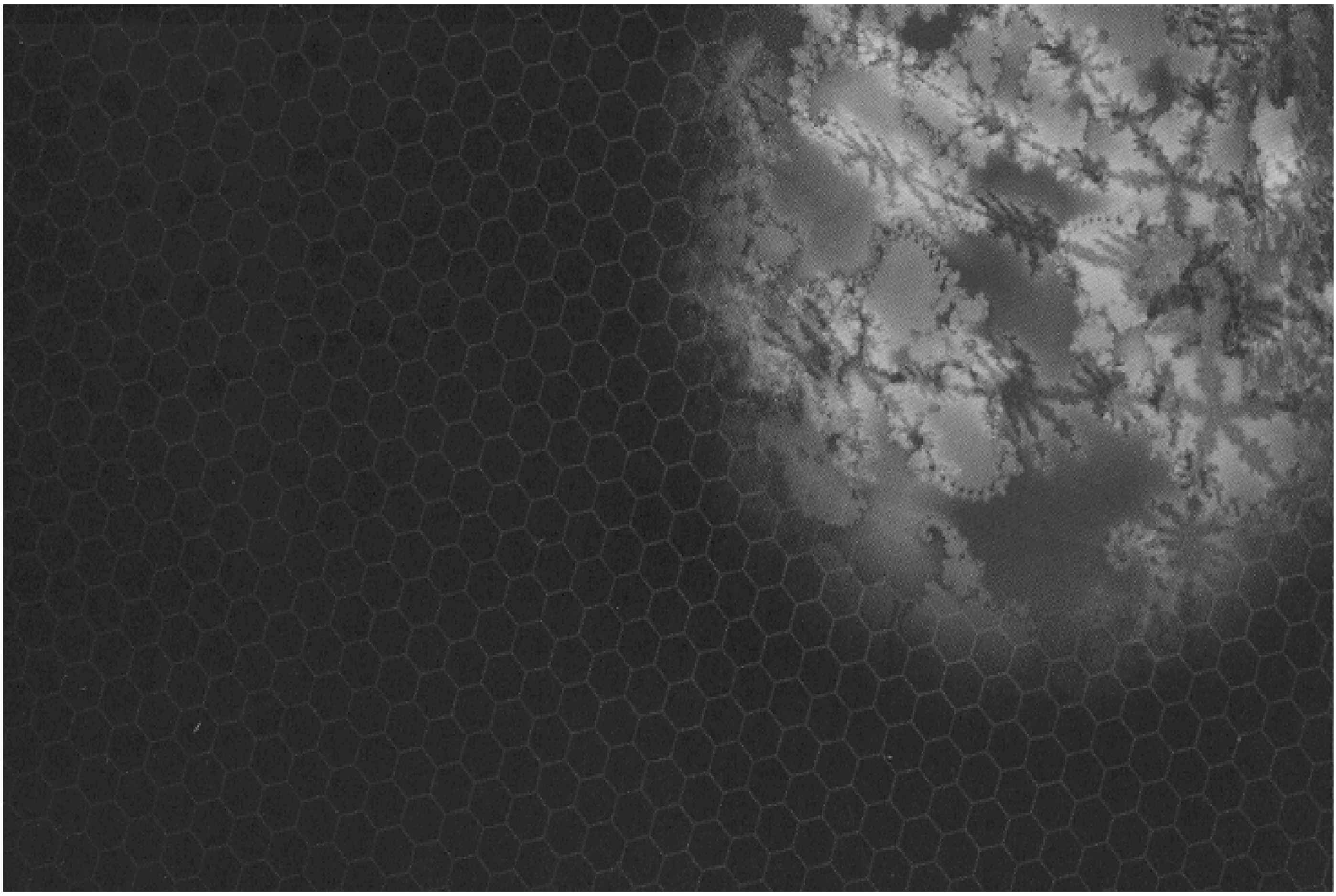
Parameter	***Effect***
isc_dpb_shutdown	Performs a database shutdown
isc_dpb_online	Puts a shut-down database back on-line
isc_dpb_gbak_attach	Allows gbak to attach to the database

Parameter	*Effect*
isc_dpb_gfix_attach	Allows gfix to attach to the database
isc_dpb_gstat_attach	Allows gstat to attach to the database
isc_dpb_verify	Allows gfix to initiate a database validation
isc_dpb_no_db_triggers	Disables database triggers (V.2.1 and higher, ODS 11.1 and higher)
isc_dpb_set_db_sql_dialect	Sets the SQL dialect of the database. Note, this is not the same as setting the SQL dialect of the connection, which is not affected.
isc_dpb_sweep_interval	Sets the sweep interval in the database header
isc_dpb_force_write	Sets forced writes on or off in the database header
isc_dpb_no_reserve	Alters the reserve page space attribute in the database header
isc_dpb_set_db_readonly	Makes the database read-only or changes the database from read-only to read-write
isc_dpb_set_page_buffers (on Superserver)	Sets the size of the page cache (see below)

On Classic, isc_dpb_set_page_buffers can still be requested by an ordinary user. It will set the buffer size temporarily for that user and that session only. When used by the SYSDBA on either Superserver or Classic, it will change the buffer count in the database header, i.e., make a permanent change to the default buffer size.

Chapter 7 Errata

Page	Erratum
98	The example API category which states: "Database Security (e.g. isc_attach_database())" should be: "Database Security (e.g. **isc_add_user()**)"



Part Three

Firebird Data Types

Topics in This Part

Chapter 8
About Firebird
Data Types

Chapter 8 Errata

Page	Erratum
117	Under the heading "Pre-Defined Date Literals", the second sentence of the intro reads "In dialect 1, the strings can be used directly; in dialect 3, they must be cast to type. " It should be changed to read "In legacy SQL, the strings could be used directly; in both dialects of Firebird, they must be cast to type in most situations." Beneath Table 8-2, add the sentence: "Any attempt to use a date literal in an expression without casting will cause an exception."
118	The first sentence reads "In a dialect 1 database, this statement returns exact server time:" Change this to "In legacy SQL, this statement would return exact server time:" Change the sentence "In a dialect 3 database, the date literal must be cast as a TIMESTAMP type:" to "In Firebird, the date literal must be cast as a TIMESTAMP type:"
ditto	The lead-in to the next example reads:

"This UPDATE statement sets a date column to server time plus one day in dialect 1:"

Change this (note MULTIPLE changes here) to

"Firebird still allows date literals to be used alone as the argument in UPDATE and INSERT statements and search criteria. This UPDATE statement sets a date column to server date plus one day:"

The next body sentence reads:

"Here's the same operation in dialect 3, with casting:"

Change this to:

118 (cont.) "In Firebird, using date literals in expressions will cause an exception. A clause such as `SET UPDATE_DATE = 'TODAY' + 1` will not work. Such expressions must be explicitly cast:"

Then, in the succeeding code example, the second line reads:

```
SET UPDATE_DATE = CAST('TOMORROW' AS DATE)
```

Change this to

```
SET UPDATE_DATE = CAST('TODAY' AS DATE) + 1
```

Chapter 9
Number Types

Number types continue to behave as before, according to dialect.

Sequences

A sequence generator is a mechanism for generating successive 64-bit integer values, one at a time. The SEQUENCE object has been introduced in Firebird 2.x as a a synonym for GENERATOR, in accordance with SQL-99.

The syntax term **SEQUENCE** is a described in the SQL specification, whereas **GENERATOR** is a non-standard term inherited from InterBase. Use of the standard SEQUENCE syntax in your applications is recommended.

To refer to the legacy syntax, see *Generators* on Page 130 of Chapter 9.

A sequence generator is a named schema object. In dialect 3 it is a BIGINT, in dialect 1 it is an INTEGER.

Syntax Patterns

```
CREATE { SEQUENCE | GENERATOR } <name>
DROP { SEQUENCE | GENERATOR } <name>
SET GENERATOR <name> TO <start_value>
ALTER SEQUENCE <name> RESTART WITH <start_value>
GEN_ID (<name>, <increment_value>)
NEXT VALUE FOR <name>
```

Examples

1. Equivalent to **CREATE GENERATOR**:

```
CREATE SEQUENCE S_ACCOUNT_ID;
```

2. Equivalent to **SET GENERATOR *n***:

```
ALTER SEQUENCE S_ACCOUNT_ID RESTART WITH 0;
```

ALTER SEQUENCE, like **SET GENERATOR**, is a good way to corrupt your database! Don't use it unless you really mean to break the sequence and destroy the validity of key values generated from it.

Fetching a Sequence Value in an Expression

The SQL-99 compliant **NEXT VALUE FOR <*sequence_name*>** expression syntax is synonymous with **GEN_ID(<*generator-name*>,1)**, complementing the introduction of **CREATE SEQUENCE**.

Examples

1. Calling the **GEN_ID()** function:

```
SELECT GEN_ID(S_ACCOUNT_ID, 10) FROM RDB$DATABASE;
```

2. Using the **NEXT VALUE FOR** expression:

```
INSERT INTO ACCOUNT (ID, NAME)
VALUES (NEXT VALUE FOR S_ACCOUNT_ID, 'Acme Software Ltd');
```

Currently, increment ("step") values not equal to 1 (one) can be achieved only by calling the **GEN_ID()** function. Future versions are expected to provide full support for SQL-99 sequence generators, which allows the required increment values to be defined when creating the sequence.

Unless there is a vital need to use a step value that is not 1, use of a **NEXT VALUE FOR** value expression instead of the **GEN_ID()** function is recommended.

GEN_ID(<name>, 0) allows you to retrieve the current sequence value, but it should never be used in insert/update statements, e.g. to "blind-write" a foreign key value or in an incrementing expression, as it produces a high risk of uniqueness violations in a concurrent environment.

Chapter 9 Errata

Page	Erratum
131	On the last example there is a closing parenthesis missing. It should read as follows: `SELECT GEN_ID(AGenerator, (` `(SELECT GEN_ID(AGenerator, 0) from RDB$DATABASE) * - 1))` `from RDB$DATABASE;`
139	The heading for the sidebar at the top of the page, “Numeric Input and Exponents” has somehow gotten the special format that was used for the expert topics. The whole sidebar should have the proper sidebar format to be consistent with all of the other sidebars in the book.

Chapter 10 Date and Time Types

No new date or time types were added to Firebird 2. However, an important change has been made to some date/time context variables.

Time and Date/Time Variables

Milliseconds

Milliseconds precision has been enabled in both DSQL and PSQL for

- the **CURRENT_TIMESTAMP** context variable
- the timestamp literal **'NOW'**
- the **CURRENT_TIME** context variable

Hundredths and Tenths of Seconds

It is also possible to retrieve the sub-second part of **CURRENT_TIME** and **CURRENT_TIMESTAMP** in hundredths, tenths or as '.000' (full seconds only).

The defaults are milliseconds precision for **CURRENT_TIMESTAMP** and full seconds precision for **CURRENT_TIME**.

The maximum possible precision is 3 which means accuracy of 1/1000 second (one millisecond). This accuracy may be improved in the future versions.

For syntax and examples of use, see Chapter 21, *Expressions and Predicates* 121.

Time and Date/Time Functions

Some additions and enhancements were were made to the collection of internal time and date/time functions in Firebird 2.1.

WEEK Argument for EXTRACT()

In addition to the arguments already supported by the EXTRACT() function (YEAR, MONTH, etc.) the function can now take WEEK as an argument and return the week of the year for a DATE or TIMESTAMP input.

For example,

```
SELECT
  ...
  EXTRACT(WEEK FROM ADATE) AS WEEKOFYEAR,
  ...
FROM ATABLE;
```

The value returned is an integer in the range 1 to 53.

Understand what you are getting when submitting a date that occurs around the turn of the year. Contrary to what your diary might tell you, under the ISO 8601 standards, dates may not overlap weeks in adjoining years and no date can fall into a gap between weeks.

The "ISO year" starts at the first Monday of Week 1 and ends at the Sunday before the new ISO year. If 1 January is on a Monday, Tuesday, Wednesday or Thursday, it is in week 01. If 1 January is on a Friday, Saturday or Sunday, it is in week 52 or 53 of the previous year. You can identify the years that have a "Week 53" by counting the number of Thursdays in the calendar year.

See the full options for EXTRACT() in Chapter 10 of *The Firebird Book*, p.157 ff., *The EXTRACT() Function*.

New Internal Functions

The following new functions for operating on date and/or time values were added in Firebird 2.1:

Function	Purpose	Syntax and Examples
DATEADD()	Returns a date/time/timestamp value incremented or decremented by the specified measure of time.	
DATEDIFF()	Returns an exact numeric value representing the number of units of time elapsed between one date/time value and another. Units supported are YEAR. MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, MILLISECOND.	

Data Type Hints for Casting Date Literals

In days gone by, before the advent of context variables like CURRENT_DATE, CURRENT_TIMESTAMP, et al., we had predefined date literals, such as 'NOW', 'TODAY', 'YESTERDAY' and so on. These predefined date literals survive in Firebird's SQL language set and are still useful.

In InterBase 5.x and lower, the following statement was “legal” and returned a DATE value (remembering that the DATE type then was what is now TIMESTAMP):

```
select 'NOW' from rdb$database /* returns system date and time */
```

In a database of ODS 10 or higher, that statement returns the string 'NOW'. We have had to learn to cast the date literal to get the result we want:

```
select cast('NOW' as TIMESTAMP) from rdb$database
```

For a long time—probably since IB 6— there has been an undocumented “short expression syntax” for casting not just the predefined date/time literals but any date literals. Actually, it is defined in the standard. Most of us were just not aware that it was available. It takes the form <data type> <date literal>. Taking the CAST example above, the short syntax would be as follows:

```
select TIMESTAMP 'NOW'
  FROM RDB$DATABASE
```

Also works with non-predefined literals:

```
SELECT TIME '15:05:45.345' FROM RDB$DATABASE
```

This short syntax can participate in other expressions. The following example illustrates a date/ time arithmetic operation on a predefined literal:

```
update mytable
  set OVERDUE = 'T'
  where DATE 'YESTERDAY' - DATE_DUE > 10
```

Chapter 10 Errata

Page	Erratum
146	In Table 10-3: For the year 98 row, 2998 should be **2098**.
147	The following text/code is incomplete: For example, the date literal.....because there is no month 14. It should read: For example, the date literal.....because there is no month 14. **However, 'CCYY/MM/DD' is accepted: '2004/12/31' will be interpreted as "31 December 2004".**
157	In Table 10.10. EXTRACT() Arguments, limits for YEARDAY appear as 1-366. Should be **0-365**.

Chapter 11
Character Types

The supplementary information in this chapter encompasses a range of features and enhancements affecting the various text types. Material specific to BLOBS of type TEXT (sub_type 1) is discussed in the next chapter. However, much of the material concerning expressions and international language support applies to text BLOBs also.

Character Metadata Conversion

V.2.1

Firebird versions 2.0.x had two problems related to character sets and metadata extraction. You will know you have at least one of these problems if one of the first things you see on connecting to your newly upgraded database is a "Malformed string" error. One aspect of the problem concerns strings that were stored in metadata definitions. It can be fixed by (carefully!) executing some scripts that were packaged up by the Firebird developers and distributed with the Firebird 2.1 binary kits. Instructions 65 are at the end of this chapter.

Features for Text Data

Firebird 2.0 introduced many improvements and new features for text data. Of major significance is the introduction of the portable C/C++ International Components for Unicode (ICU) as an important step in globalising character set support. The new architecture and usage are described in detail later in this chapter 58.

New Functions for Strings

String Manipulation Functions

Two new inbuilt string manipulation functions were added:

LOWER()

LOWER() returns the input argument converted to all lower-case characters.

Example

```
isql -q -ch dos850

SQL> create database 'test.fdb';
SQL> create table t (c char(1) character set dos850);
SQL> insert into t values ('A');
SQL> insert into t values ('E');
SQL> insert into t values ('Á');
SQL> insert into t values ('É');
SQL> select c, lower c from t;
```

```
C      LOWER
====== ======
A      a
E      e
Á      á
É      é
```

TRIM()

TRIM trims characters (default: blanks) from the left and/or right of a string.

Syntax Pattern

```
TRIM ( [ [ <trim specification> ] [ <trim character> ]
FROM ] <value expression> )
```

<trim specification> ::= LEADING | TRAILING | BOTH

<trim character> ::= <value expression>

Rules

- If <trim specification> is not specified, BOTH is assumed.
- If <trim character> is not specified, ' ' is assumed.
- If <trim specification> and/or <trim character> is specified, FROM should be specified.
- If neither <trim specification> nor <trim character> is specified, FROM should not be specified.

Examples

In the first example, all relation names in RDB$RELATION_NAME starting with 'RDB$' will be returned with the leading substring 'RDB$' trimmed off:

```
select
rdb$relation_name,
trim(leading 'RDB$' from rdb$relation_name)
from rdb$relations
where rdb$relation_name starting with 'RDB$';
```

In the next example, both leading and trailing blanks will be trimmed from RDB$RELATION_NAME and the result concatenated to a static string:

```
select
trim(rdb$relation_name) || ' is a system table'
from rdb$relations
where rdb$system_flag = 1;
```

String Size Functions

Three new functions, sharing a common syntax, can return information about the size of strings:

BIT_LENGTH()

Returns the length of a string in bits.

CHAR_LENGTH()/CHARACTER_LENGTH()

Synonymous functions returning the character count of a string.

OCTET_LENGTH()

Returns the length of a string in bytes.

Syntax Pattern

```
<length function> ::=

{ BIT_LENGTH | CHAR_LENGTH | CHARACTER_LENGTH | OCTET_LENGTH } ( <value expression>)
```

Example

```
select
rdb$relation_name,
char_length(rdb$relation_name),
char_length(trim(rdb$relation_name))
from rdb$relations;
```

More New Internal Functions

A useful collection of other string functions, most of which had a past life in one form or another as UDFs, has been implemented as internal functions for the v.2.1 release. For strings, implementations include HASH(), LEFT(), LPAD(), OVERLAY(), POSITION(), REPLACE(), REVERSE(), RIGHT() and RPAD(). All of the new functions are described with examples in *Appendix I*, *Function Summary* 244.

Enhancements for Text BLOBs

Several enhancements have been added for text BLOBs so that, by v.2.1, they can to a large extent masquerade as VARCHARS—see Chapter 11, *Blobs and Arrays* 67.

Character Sets/Collations

A number of new character sets and/or collations were added to the manifest for Firebird 2.0 and 2.1.

The full list of these new sets/collations is in Appendix XIII, *Character Sets and Collations* 274.

UNICODE_FSS Bugs Fixed

In Firebird 1.5.x, UTF8 is an *alias* to UNICODE_FSS, which is not the same as the UTF8 in Firebird 2 and beyond. It is an old version of UTF8 that has a number of problems: it accepts malformed strings and does not enforce correct maximum string length.

In previous versions, a bug with UNICODE_FSS databases caused an exception if a text search parameter was longer than 263 characters. This has been fixed in Firebird 2.0.

Another bug that caused the server to go into a loop when connecting with UNICODE_FSS as the client character set was also fixed.

UTF8 character set

In Firebird 2, UTF8 is a new character set, with collations and without the inherent problems of UNICODE_FSS.

The UNICODE Collations

The UNICODE collations (case sensitive and case insensitive) can be applied to any character set that is present in *fbintl*. They are already registered in *fbintl.conf*, but you need to register them in the databases, with the desired associations and attributes 63.

- UCS_BASIC works identically to UTF8. With no collation specified, sorts are performed in UNICODE code-point order.
- The UNICODE collation sorts using UCA (Unicode Collation Algorithm).

The UNICODE collation (**`<charset>_UNICODE`**) became available for all character sets in *fbintl*.

The UTF-8 collation is a case-insensitive collation for UTF8.

Sort Order Sample

```
isql -q -ch dos850
SQL> create database 'test.fdb';
SQL> create table t (c char(1) character set utf8);
SQL> insert into t values ('a');
SQL> insert into t values ('A');
SQL> insert into t values ('á');
SQL> insert into t values ('b');
```

```
SQL> insert into t values ('B');
SQL> select * from t order by c collate ucs_basic;

C
======
A
B
a
b
á

SQL> select * from t order by c collate unicode;

C
======
a
A
á
b
B
```

Brazilian collations

Two case-insensitive/accent-insensitive collations were created for Brazil: WIN_PTBR (for WIN1252) and PT_BR (for ISO8859_1).

Example Showing Sort order and Equality

```
isql -q -ch dos850
SQL> create database 'test.fdb';
SQL> create table t (c char(1) character set iso8859_1 collate pt_br);
SQL> insert into t values ('a');
SQL> insert into t values ('A');
SQL> insert into t values ('á');
SQL> insert into t values ('b');
SQL> select * from t order by c;

C
======
A
a
á

SQL> select * from t where c = 'â';

C
======
a
```

```
A
â
```

Spanish collations

Collations ES_ES and the new ES_ES_CI_AI automatically use attributes 63 DISABLE-COMPRESSIONS=1;SPECIALS-FIRST=1 in a new or restored ODS 11.1 database.

The ES_ES_CI_AI collation was standardised to current usage.

New INTL Interface for Non-ASCII Character Sets

A feature of Firebird 2 is the introduction of the ICU interface for international character sets.

Existing Architecture

Firebird allows character sets and collations to be defined for any character field or variable declaration. The default character set can also be specified at database create time, to make every CHAR or VARCHAR declaration in the database use that character set if a CHARACTER SET clause is not specified.

When a client attachment is made, the character set that the client is to use to read strings can be specified. If no client character set is specified, it defaults to character set NONE.

Two special character sets, NONE and OCTETS, can be used in declarations. However, OCTETS cannot be used as a connection character set. The two sets are similar, except that the space character of NONE is ASCII 0x20, whereas the space character OCTETS is 0x00. NONE and OCTETS follow different rules to those that other charsets do regarding conversions.

With other character sets, conversion is performed as CHARSET1->UNICODE->CHARSET2, while with NONE and OCTETS, the bytes are just copied verbatim: NONE/OCTETS->CHARSET2 and CHARSET1->NONE/OCTETS.

Enhancements

Some character sets (especially multi-byte) do not accept just any string. Now, the engine verifies that strings are well-formed when assigning from NONE/OCTETS and when strings sent by the client (the statement string and parameters).

Uppercasing

Previously only ASCII characters were uppercased in a character set's default (binary) collation. The binary collation is the one whose name matches the name of the character set. It is the collation used for uppercasing if no collation is specified.

For example,

```
isql -q -ch dos850
SQL> create database 'test.fdb';
SQL> create table t (c char(1) character set dos850);
SQL> insert into t values ('a');
```

```
SQL> insert into t values ('e');
SQL> insert into t values ('á');
SQL> insert into t values ('é');

SQL> select c, upper(c) from t;

C      UPPER
====== ======
a      A
e      E
á      á
é      é
```

In Firebird 2.0 the result is:

```
C      UPPER
====== ======
a      A
e      E
á      Á
é      É
```

Maximum String Length

Previously, the engine did not verify the logical length of multi-byte character set (MBCS) strings. Hence, a UNICODE_FSS field would take three times as many characters as the declared field size, three being the maximum length of one UNICODE_FSS character). The three-byte rule applied, even if the actual character was only one or two bytes.

The old MBCS byte-length rule has been retained for compatibility for legacy character sets. New character sets (UTF8, for example) do not inherit this limitation.

sqlsubtype

When the character set of a CHAR or VARCHAR column is anything but NONE or OCTETS and the attachment character set is not NONE, the *sqlsubtype* member of an XSQLVAR pertaining to that column now contains the ID of the attachment (connection) character set instead of the ID of the column's character set.

Language Plug-ins

New character sets and collations are implemented through dynamic libraries and installed in the server with a *manifest file*. Not all implemented character sets and collations need to be listed in the manifest file. Only those listed are available and duplications are not loaded. The new plug-in architecture uses it to locate character sets and collations in the libraries.

If a character set/collation is declared more than once, it is not loaded and the error is reported in the log.

The manifest file should be put in the **`$rootdir/intl`** with a ".conf" extension. For an example, see **`fbintl.conf`**.

Dynamic Libraries

For Windows, the dynamic libraries for the plug-in language support are located in the **`\icu`** directory of your Firebird installation, with names starting with "icu", e.g., Firebird 2.0 shipped with icudtl30.dll, icuin30.dll and icuuc30.dll.

For Linux, the shared objects and symlinks are installed in the **`/lib`** subdirectory with names starting with "libicu".

You must deploy the ICU libraries. This applies to the Windows Embedded Server model as well, in which the libraries are deployed in the application root directory.

Adding More Character Sets to a Database

For installing additional character sets and collations into a database, the character sets and collations should be registered in the database's system tables (rdb$character_sets and rdb$collations). The file **`/misc/intl.sql`**, in your Firebird 2 installation, is a script of stored procedures for registering and unregistering them.

Example of a Section from `fbintl.conf`

The symbol **`$(this)`** is used to indicate the same directory as the manifest file and the library extension should be omitted.

```
<intl_module fbintl>
filename $(this)/fbintl
</intl_module>

<charset ISO8859_1>
intl_module fbintl
collation ISO8859_1
collation DA_DA
collation DE_DE
collation EN_UK
collation EN_US
collation ES_ES
collation PT_BR
collation PT_PT
</charset>
```

```
<charset WIN1250>
intl_module fbintl
collation WIN1250
collation PXW_CSY
collation PXW_HUN
collation PXW_HUNDC
</charset>
```

Using ICU Character Sets

All non-wide and ASCII-based character sets present in ICU can be used by Firebird 2.1. To reduce the size of the distribution kit, we customize ICU to include only essential character sets and any for which there was a specific feature request.

If the character set you need is not included, you can replace the ICU libraries with another complete module, found at our site or already installed in your operating system.

Registering an ICU Character Set Module

To use an alternative character set module, you need to register it in two places:

1. in the server's language configuration file, intl/fbintl.conf
2. in each database that is going to use it

Registering a Character Set on the Server

Using a text editor, register the module in *intl/fbintl.conf*, as follows.-

```
<charset         NAME>
    intl_module     fbintl
    collation       NAME [REAL-NAME]
</charset>
```

For example, to register a new character set and two collations, add the following to *fbintl.conf*:

```
<charset         GB>
    intl_module     fbintl
    collation       GB
    collation       GB18030
</charset>
```

! You should prepare and register the module in the server before you register the character set[s] in the database.

Registering a Character Set in a Database

To make the character set and your required collation available in a database is a two-step task:

1. Run the procedure *sp_register_character_set*, the source for which can be found in *misc/intl.sql* beneath your Firebird 2.1 root, AND—
2. Use a CREATE COLLATION statement 62 to register each collation

Step 1: Using the Stored Procedure

Here is a sample declaration that you might have added to *fbintl.conf* :

The stored procedure takes two arguments: a string that is the character set's identifier as declared in the configuration file and a smallint that is the maximum number of bytes a single character can occupy in the encoding. For our example:

```
execute procedure sp_register_character_set ('GB', 4);
```

Step 2: Registering the Collations

For the purpose of our example, the syntax for registering the two collations in a database is very simple:

```
CREATE COLLATION GB
    FOR GB;

CREATE COLLATION GB18030
    FOR GB;
```

More complex directives can be used in CREATE COLLATION, as we discover next.

The CREATE COLLATION Statement

If you are using the Firebird 2.1 server and your database is ODS 11.1 or higher, the new CREATE COLLATION statement provides a way for you to register your collation to your database through dynamic SQL.

Syntax Pattern

```
CREATE COLLATION <name>
    FOR <charset>
    [ FROM <base> | FROM EXTERNAL ('<name>') ]
    [ NO PAD | PAD SPACE ]
    [ CASE SENSITIVE | CASE INSENSITIVE ]
    [ ACCENT SENSITIVE | ACCENT INSENSITIVE ]
    [ '<specific-attributes>' ]
```

Specific attributes should be separated by semicolons and are case sensitive.

The new collation should be declared in a .conf file in $root/intl directory before you prepare and execute the CREATE COLLATION statement.

Examples

```
CREATE COLLATION UNICODE_ENUS_CI
    FOR UTF8
    FROM UNICODE
```

```
    CASE INSENSITIVE
    'LOCALE=en_US';

CREATE COLLATION NEW_COLLATION
    FOR WIN1252
    PAD SPACE;
```

Use the DDL statement `DROP COLLATION <collation-name)` to de-register an unwanted collation.

More examples follow in the next section, showing how to use specific attributes.

Conventions and Attributes

The naming convention you should use is the character set name followed by an underscore character followed by the collation name:

`<characterset>_<collation>`

The names should be as in *fbintl.conf* (i.e. ISO8859_1 instead of ISO88591, for example).

For example,

```
CREATE COLLATION WIN1252_UNICODE
  FOR WIN1252;

CREATE COLLATION WIN1252_UNICODE_CI
  FOR WIN1252
  FROM WIN1252_UNICODE
  CASE INSENSITIVE;
```

Specific Attributes for Collations

- Some attributes may not work with some collations, even though they do not report an error.
- The attributes are stored at database creation time, so the changes do not apply to databases with an ODS lower than 11.1.

DISABLE-COMPRESSIONS

Prevents compressions (otherwise referred to as "contractions") from changing the order of a group of characters.

Valid for collations of narrow character sets.

Format: DISABLE-COMPRESSIONS={0 | 1}

Example

```
DISABLE-COMPRESSIONS=1
```

DISABLE-EXPANSIONS

Prevents expansions from changing the order of a character to sort as a group of characters.

Valid for collations of narrow character sets.

Format: DISABLE-EXPANSIONS={0 | 1}

Example

```
DISABLE-EXPANSIONS=1
```

ICU-VERSION

Specifies the version of the ICU library to be used for UNICODE and UNICODE_CI.

Valid values are the ones defined in the config file (*intl/fbintl.conf*) in the entry **`intl_module/icu_versions`**.

Format: ICU-VERSION={default | major.minor}

Example

```
ICU-VERSION=3.0
```

LOCALE

Specifies the collation locale for UNICODE and UNICODE_CI.

Requires the complete version of the ICU libraries.

Format: LOCALE=xx_XX

Example

```
LOCALE=en_US
```

MULTI-LEVEL

Registers that the collation uses more than one level for ordering purposes.

Valid for collations of narrow character sets.

Format: MULTI-LEVEL={0 | 1}

Example

```
MULTI-LEVEL=1
```

SPECIALS-FIRST

Specifies that special characters (spaces, symbols, etc) precede alphanumeric characters.

Valid for collations of narrow character sets.

Format: SPECIALS-FIRST={0 | 1}

Example

```
SPECIALS-FIRST=1
```

In isql, use **`SHOW COLLATION <characterset>_<collation>`** to display the attributes.

Metadata Text Conversion

Firebird versions 2.0.x had two problems related to character sets and metadata extraction. You will know you have at least one of these problems if one of the first things you see on connecting to your newly upgraded database is a "Malformed string" error.

Problem 1

When creating or altering objects, text associated with metadata was not transliterated from the client character set to the system (UNICODE_FSS) character set of these BLOB columns. Instead, raw bytes were stored there.

The types of text affected were PSQL sources, descriptions, text associated with constraints and defaults, and so on.

> The problem can still occur if CREATE or ALTER operations are performed with the connection character set as NONE or UNICODE_FSS and you are using non-UNICODE_FSS data or if you process scripts containing strings that were written in an external editor and stored in ASCII or ANSI encoding.

Problem 2

In reads from text BLOBs, transliteration from the BLOB character set to the client character set was not being performed. There is no pre-packaged solution to this problem but the metadata scripts used for solving Problem 1 should give you a good idea of what you will need to do with wrongly encoded text BLOBs to put your malformed data in good shape.

Repairing Your Metadata Text

If your metadata text was created with encoding that was consistently wrong, it can be repaired with procedures distributed in your Firebird 2.1 binary kit. The database will have to be repaired in order to read the metadata correctly. The repair script will be installed beneath your Firebird root directory in

`/misc/upgrade/metadata/metadata_charset_create.sql`

The procedure involves multiple passes through the database, using scripts.

It is strongly recommended that you disconnect and reconnect before each pass.

Steps

The database should already have been converted to ODS11.1 by way of a gbak backup and restore.

Before doing anything, make a file copy of the database.

In the examples that follow, the string **`$fbroot$`** represents the path to your Firebird installation root directory, e.g. /opt/firebird.

Create the procedures in the database

```
isql /path/to/your/database.fdb

SQL> input '$fbroot$/misc/upgrade/metadata/metadata_charset_create.sql';
```

Check your database

```
isql /path/to/your/database.fdb

SQL> select * from rdb$check_metadata;
```

The **rdb$check_metadata** procedure will return all objects that are touched by it.

- If no exception is raised, your metadata is OK and you can go to the section Remove the upgrade procedures 66
- Otherwise, the first bad object is the last one listed before the exception.

Fixing the metadata

To fix the metadata, you need to know in what character set the objects were created. The upgrade script will work correctly only if all your metadata was created using the same character set.

The **rdb$fix_metadata** procedure will return the same data as **rdb$check_metadata**, but it will change the metadata texts.

It should be run once!

```
isql /path/to/your/database.fdb

SQL> input '$fbroot$/misc/upgrade/metatdata/metadata_charset_create.sql';

SQL> select * from rdb$fix_metadata('WIN1252');  -- replace WIN1252 by your charset

SQL> commit;
```

After this, you can remove the upgrade procedures.

Remove the upgrade procedures

```
isql /path/to/your/database.fdb

SQL> input '$fbroot$/misc/upgrade/metadata/metadata_charset_drop.sql';
```

Chapter 12
BLOBs and Arrays

Enhancements for Text BLOBs

There is quite a lot that is new for text BLOBs, especially in v.2.1. Many of the improvements made for character types apply to text generally, including text stored in and accessed from BLOBs of sub_type 1 (moniker TEXT).

Compatibility with VarChar

2.1 At various levels of evaluation, the engine now treats text BLOBs that are within the 32,765-byte string size limit as though they were VARCHAR. Operations that now allow text BLOBs to behave like strings are assignments, conversions and concatenations, as well as the functions CAST, LOWER, UPPER, TRIM and SUBSTRING.

COLLATE clauses

A DML **COLLATE** clause is now allowed with BLOBs.

Example

```
select blob_column from table
where blob_column collate unicode = 'foo';
```

Equality Comparisons Between BLOBs

Comparison can be performed on the entire content of a text BLOB.

Character Set Conversion

Conversion between character sets is now possible when assigning to a BLOB from a string or another BLOB.

Descriptive Monikers for Subtypes

Previously, the only allowed syntax for declaring a BLOB filter was:

```
declare filter <name>
  input_type <sub_type_number>
  output_type <sub_type_number>
  entry_point <function_in_library>
  module_name <library_name>;
```

A new, alternative new syntax allows the BLOB filter to be identified by a known moniker:

```
declare filter <name>
  input_type <moniker>
  output_type <moniker>
  entry_point <function_in_library>
  module_name <library_name>;
```

where <moniker> refers to a subtype identifier known to the engine.

Initially they are BINARY (for sub_type 0) TEXT (for sub_type 1) and others mostly for internal usage.

Pre-defined Types

To list the predefined types, do

```
select RDB$TYPE, RDB$TYPE_NAME, RDB$SYSTEM_FLAG
from rdb$types
where rdb$field_name = 'RDB$FIELD_SUB_TYPE';
```

```
RDB$TYPE  RDB$TYPE_NAME                RDB$SYSTEM_FLAG
========= ============================ =================
        0 BINARY                                     1
        1 TEXT                                       1
        2 BLR                                        1
        3 ACL                                        1
        4 RANGES                                     1
        5 SUMMARY                                    1
        6 FORMAT                                     1
        7 TRANSACTION_DESCRIPTION                    1
        8 EXTERNAL_FILE_DESCRIPTION                  1
```

Example

Original declaration:

```
declare filter permit
  input_type 0
  output_type 3
entry_point 'f' module_name 'p';
```

Alternative declaration:

```
declare filter permit
  input_type binary
  output_type acl
entry_point 'f' module_name 'p';
```

User-defined Types

An adventurous user could write a new moniker in RDB$TYPES and use it, since a moniker is parsed only at declaration time. The engine keeps the numerical value.

User-defined BLOB filters should use only negative sub_type values.

Example

Declaring a name for a user defined blob subtype:

```
SQL> insert into rdb$types
CON> values('RDB$FIELD_SUB_TYPE', -100, 'XDR', 'test type', 0);
SQL> commit; -- essential!
SQL> declare filter marmalade input_type xdr output_type text
CON> entry_point 'p2' module_name 'p';
SQL> show filter marmalade;

BLOB Filter: MARMALADE
Input subtype: -100 Output subtype: 1
Filter library is p
Entry point is p2
```

Some BLOB Bugs Fixed

Previously, the SUBSTRING() function did not work correctly with a text BLOB having a character set attribute. This was fixed.

Another bug was fixed, whereby pattern matching with multi-byte text BLOBs was being performed in binary mode.

String search operators now work correctly with BLOBs of any size. Issues with only the first segment being searched and with searches missing matches that straddle segment boundaries are now gone.

One BLOB Bug Not Fixed

During the Firebird 2.1 release cycle, changes were introduced to reject sorts (ORDER BY, GROUP BY and SELECT DISTINCT operations) at prepare time if the sort clause implicitly or explicitly involved sorting on a BLOB or ARRAY column.

That change was reversed in the final pre-release version, not because it was wrong but because so many users complained that it broke the behaviour of legacy applications.

This reversion to "bad old behaviour" does not in any way imply that such queries will magically return correct results. A BLOB cannot be converted to a sortable type and so, as previously, DISTINCT sortings and ORDER BY arguments that involve BLOBs, will use the BLOB_ID. As before, GROUP BY arguments that are BLOB types will prepare successfully, but will cause run-time exceptions.

Chapter 12 Errata

Page	Erratum
187	Under the heading "Defining Arrays", the sentence "For example, the following statement defines both a regular character column and ..." This is inconsistent with the example shown. Change this part of the sentence so it reads sentence "For example, the following statement defines a 64-bit integer column and ..."

Chapter 13
Domains

Very little about domains changed in 2.0.x releases.

The ability to use domains as types for PSQL arguments and variables appeared in Firebird 2.1.

Constraint Changes

Some minor changes relating to constraints may affect legacy code.

CHECK Constraints and NULL

Formerly, CHECK constraints were not SQL standard-compliant with respect to the handling of NULL. For example, CHECK (DEPTNO IN (10, 20, 30)) should allow NULL in the DEPTNO column but it did not.

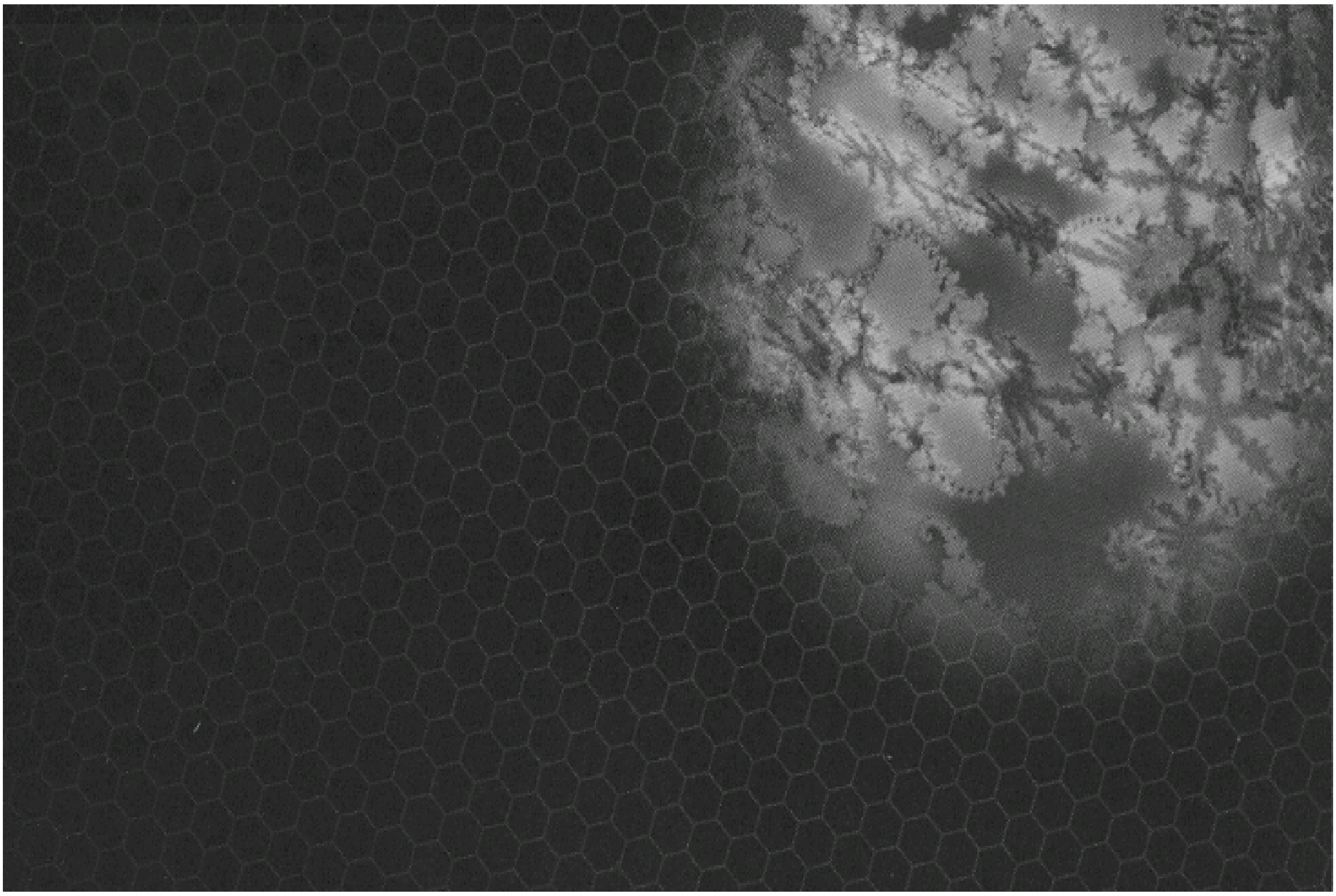
In Firebird 2.0, if you need to make NULL invalid in a CHECK constraint, you must do so explicitly by extending the constraint. Using the example above:

```
CHECK (DEPTNO IN (10, 20, 30)
AND DEPTNO IS NOT NULL)
```

NOT NULL Constraints

View definitions no longer inherit a NOT NULL constraint from the domain of the underlying table's column definition.

See Chapter 24, *Views* 145, for more details.



Part Four

A Database and Its Objects

Topics in This Part

Chapter 14
From Drawing Board to Database

Chapter 14 Errata

Page	Erratum
221	The last sentence of the "TIP" section reads "Ensure that all DML statements are committed before introducing any DML." It should read : "Ensure that all **DDL** statements are committed before introducing any DML."
226	Script example in centre of page will not work because CREATE DATABASE arguments are in incorrect order. The example code reads: `SET SQL DIALECT 3;` `CREATE DATABASE 'd:\databases\MyDatabase.fdb'` `PAGE_SIZE 8192` `DEFAULT CHARACTER SET ISO8859_1` *`USER 'SYSDBA' PASSWORD 'masterkey';`* It should be changed to: `SET SQL DIALECT 3;` `CREATE DATABASE 'd:\databases\MyDatabase.fdb'` `USER 'SYSDBA' PASSWORD 'masterkey'` `PAGE_SIZE 8192` `DEFAULT CHARACTER SET ISO8859_1 ;`

Chapter 15 Creating and Maintaining a Database

Everything that is applicable to v.1.5 database creation and maintenance applies to Firebird 2 databases. However, several database-level enhancements are of interest to the developer.

Caching and Input/Output Features

Ever in pursuit of improved performance around I/O, especially in the light of the growing prevalence of high-performance, high-capacity storage hardware, the Firebird developers introduced a number of enhancements in the areas of caching and page allocation.

Minimum Page Size Raised

Page sizes smaller than 4,096 bytes have been off the menu for some time. The Firebird 2.x security database (security2.fdb) requires a minimum page size of 4 Kb. With the Firebird 2.1 release, for ODS 11.1 databases, page sizes of 1024 and 2048 bytes have been deprecated. The Firebird 2.1 server can still connect to databases of lower ODS with the small page sizes.

Cache Limit Raised to 128,000 Pages

Databases created under Firebird 2 can have a default cache size of up to 128,000 pages. With the maximum page size of 16 Kb, it is thus now possible to utilise up to 2 Gb of RAM for caching on 64-bit systems running a 64-bit Superserver.

It should not be assumed that 32-bit systems with large amounts of RAM can be configured with a huge cache in the belief that the server can be made to use more of the available RAM. A 32-bit process is limited to a total of 2 Gb of RAM, regardless of how much is installed.

If you are running a Classic server with multiple users, don't consider configuring a huge cache. The Classic architecture creates a separate process for each connection, assigning a default cache of the configured size to each process. By contrast, Superserver is a threaded process that assigns connections to threads and shares the cache among all threads.

Cache-thrash Problem Resolved

The long-term bug that exhibited as cache sizes of more than 8,000 pages on Superserver causing extraordinary thrashing of resources was fixed.

Bypass Filesystem Caching on Superserver

Firebird uses and maintains its own cache in memory for page buffers. The operating system, in turn, may re-cache Firebird's cache in its own filesystem cache. If Firebird is configured to use a cache that is large relative to the available RAM and Forced Writes is on, this cache duplication drains resources for little or no benefit.

Often, when the operating system tries to cache a big file, it moves the Firebird page cache to the swap, causing intensive, unnecessary paging. In practice, if the Firebird page cache size for Superserver is set to more than 80 per cent of the available RAM, resource problems will be extreme.

Now, Superserver on both Windows and POSIX can be configured by a new configuration parameter, *MaxFileSystemCache* 197, to prevent or enable filesystem caching. It may provide the benefit of freeing more memory for other operations such as sorting and, where there are multiple databases, reduce the demands made on host resources.

For Classic, there is no escaping filesystem caching.

Filesystem caching is of some benefit on file writes, but only if Forced Writes is OFF, which is not recommended for most conditions.

Disk Allocation in Chunks

Until v.2.1, Firebird had no special rules about allocating disk space for database file pages. Because of dependencies between pages that it maintains itself, to service its “careful write” strategy, it has just written to newly-allocated pages in indeterminate order.

For databases using ODS 11.1 and higher, Firebird servers from v.2.1 onward use a different algorithm for allocating disk space, to address two recognised problems associated with the existing approach: *corruptions* arising from out-of-space conditions on disk and the *file fragmentation* that results from numerous small allocations of disk space.

The Solution

The solution was to introduce some new rules and rationales to govern page writes when new page allocations are required from the filesystem. Now, new pages are pre-allocated from disk and "initialised" in the page inventory before data is written to them. If the initialisation fails due to inadequate free space, the engine will withhold the data in cache and throw an I/O exception. Corruption is thereby avoided, since it is guaranteed that dirty pages in the cache will not be written unless disk space is allocated.

For now, this aspect of the solution is effective only on Windows, since Linux filesystems currently do not expose the necessary API hooks to enable Firebird to detect an approaching out-of-space condition. However, it is known that developments in Linux will make them available in popular filesystems eventually.

To mitigate the overhead of extra writes to the page inventory, pages are pre-allocated in batches of up to 128 Kb, instead of one-by-one. In addition to providing a "safety window" for writes from cache, this batching also reduces fragmentation.

Adjusting the Chunk Size

The upper size limit of the pre-allocated chunks can be configured using the parameter ***DatabaseGrowthIncrement*** in *firebird.conf*. Be sure to read the details regarding this configuration, under *DatabaseGrowthIncrement* 197 in Chapter 36, *Configuration and Special Features*.

A Nasty Problem on Linux is Fixed

For maximum database safety, we configure databases for synchronous writes, a.k.a. Forced Writes ON. This mode—strongly recommended for normal production usage—makes the write() system call return only after the physical write to disk is complete. In turn, it guarantees that, after a COMMIT, any data modified by the transaction is physically on the hard-drive, not waiting in the operating system's cache.

During some of his explorations for Firebird 2.1, one of the core developers discovered that, thanks to a bug in file synchronisation code in the Linux kernel, Forced Writes *never worked on Linux*. You can read about his forensics in the Firebird 2.1 release notes.

Forced Writes has been reimplemented in Firebird 2.1 and it now WORKS. The repaired code was backported to Firebird 2.0.4.

Tip for users of Firebird versions 2.0.3 and lower on Linux

Here's a tip if you want to do an instant fix for the problem in an older version of Firebird: use the “sync” option when mounting any partition with a Firebird database on board.

Here is an example of a line in /etc/fstab that achieves this:

```
/dev/sda9    /usr/database    ext3    noatime,sync    1   2
```

More Database-Level Features

Database Triggers

For the first time, with a Firebird 2.1 or higher server and an ODS 11.1 or higher database, you can define triggers that fire in events beyond the boundaries of statement execution. The term *database trigger* covers PSQL modules that you can define to be executed at *transaction* or *connection* level. For more about this topic, see Chapter 31, *Triggers* 170.

Database Monitoring

Firebird 2.1 introduces the ability to monitor server-side activity happening inside a particular database. The engine can now deliver a set of so-called “virtual” tables on demand from a database of ODS 11.1 or higher, providing snapshots of the current activity within a database. For more about this topic, see Chapter 41, Database Monitoring 229.

Reworking of Garbage Collection

Since Firebird 1.0 and earlier, the Superserver engine has performed background garbage collection, maintaining information about each new record version produced by an UPDATE or DELETE statement. As soon as the old versions are no longer "interesting", i.e. when they become older than the Oldest Snapshot transaction (seen in the gstat -header output) the engine signals for them to be removed by the garbage collector.

Background GC eliminates the need to re-read the pages containing these versions via a SELECT COUNT(*) FROM aTable or other table-scanning query from a user, as occurs in Classic and in versions of InterBase prior to v.6.0. This earlier GC mechanism is known as cooperative garbage collection.

Background GC also averts the possibility that those pages will be missed because they are seldom read. (A sweep, of course, would find those unused record versions and clear them, but the next sweep is not necessarily going to happen soon.) A further benefit is the reduction in I/O, because of the higher probability that subsequently requested pages still reside in the buffer cache.

Between the point where the engine notifies the garbage collector about a page containing unused versions and the point when the garbage collector gets around to reading that page, a new transaction could update a record on it. The garbage collector cannot clean up this record if this later transaction number is higher than the Oldest Snapshot or is still active. The engine again notifies the garbage collector about this page number, overriding the earlier notification about it and the garbage will be cleaned at some later time.

In Firebird 2.0 Superserver, both cooperative and background garbage collection are now possible. To manage it, the new configuration parameter `GCPolicy` was introduced into `firebird.conf`. It can be set to:

- **cooperative** - garbage collection will be performed only in cooperative mode (like Classic) and the engine will not track old record versions. This reverts GC behaviour to that of IB 5.6 and earlier. It is the only option for Classic.
- **background** - garbage collection will be performed only by background threads, as is the case for Firebird 1.5 and earlier. User table-scan requests will not remove unused record versions but will cause the GC thread to be notified about any page where an unused record version is detected. The engine will also remember those page numbers where UPDATE and DELETE statements created back versions.
- **combined** - both background and cooperative garbage collection are performed. Obsolete back versions found on the same page are removed immediately. However, if the back version "chain" extends onto other pages, cooperative GC for that page does not proceed and a notification is left for the backround GC thread instead.

1. The Classic server ignores this parameter and always works in "cooperative" mode.
2. For Superser V.2.1.3 and V.2.05, the default GCPolicy was changed from **combined** to **background**.

Performance Effects

With the default *combined* behaviour you can expect better overall performance as the in-line garbage collection tends to curtail the growth of version chains under high load.

However, it might also cause some queries to be slower to begin returning data if the volume of old record versions in the affected tables is especially high. Particularly susceptible to this problem would be databases of ODS10 and lower, because of less effective garbage collection on indexes.

The **GCPolicy** parameter in *firebird.conf* allows the former behaviour to be reinstated if you have databases exhibiting this problem.

A GC Optimisation

The background garbage collector process was reading all back versions of records on a page, including those created by active transactions. Since back versions of active records cannot be considered for garbage collection, it was wasteful to read them. A v.2.1 optimisation averts this problem.

New COMMENT Statement for DDL

The COMMENT statement has been implemented for inserting optional descriptions for various metadata objects. Comment text is stored in a BLOB of subtype TEXT. The feature can be applied any of the following object types:

```
DOMAIN | TABLE | VIEW | PROCEDURE | TRIGGER
EXTERNAL FUNCTION | FILTER | EXCEPTION
GENERATOR | SEQUENCE | INDEX | ROLE
CHARACTER SET | COLLATION
```

For details, refer to *The COMMENT Statement* 95 in Chapter 19, *Firebird's SQL Language*.

Chapter 16
Tables

Improvements for Tables

Some small improvements have been made for defining and maintaining tables.

Also of interest is the introduction of syntax for two additional styles of virtual table.

One is the *derived table*, a single- or multi-column set derived from a subquery embedded in a SELECT query, that can be treated as though they it were a real table.

For details, see Derived Tables 104 in Chapter 20, *DML Queries*.

The other is the *global temporary table*, or GTT, introduced in Firebird 2.1.

For details, see Global Temporary Tables 147 in Chapter 24, *Views and Other Virtual Tables*

Table Definition Enhancements

The following enhancements to table definition syntax were introduced.

SET/DROP DEFAULT

ALTER TABLE syntax has been enhanced to make it possible to set or drop a default value on a column, in the same way as it has hitherto been possible to do with domains.

Syntax Pattern

```
ALTER TABLE t
  ALTER [COLUMN] c
  SET DEFAULT <default_value>;

ALTER TABLE t
  ALTER [COLUMN] c
  DROP DEFAULT;
```

If you change the type of a field, there are conditions wherein a default may remain in place. For example, if a column is declared with a domain as its type, and the domain has a default, then any default subsequently declared for the column overrides the domain's default. Dropping the column's default will revert the default to that of the domain.

Array fields cannot have a default value.

SQL2003-Compliant Alternative for Computed Fields

SQL-compliant alternative syntax GENERATED ALWAYS AS was implemented for defining a computed field in CREATE/ALTER TABLE.

Syntax Pattern

```
<column name> [<type>] GENERATED ALWAYS AS ( <expr> )
```

It is fully equivalent semantically to the legacy form:

```
<column name> [<type>] COMPUTED [BY] ( <expr> )
```

Example

```
CREATE TABLE T (
  PK INT,
  EXPR GENERATED ALWAYS AS (PK + 1))
```

CHECK Constraints and NULL

Formerly, CHECK constraints were not SQL standard-compliant with respect to the handling of NULL. For example, CHECK (DEPTNO IN (10, 20, 30)) should allow NULL in the DEPTNO column but it did not.

In Firebird 2.0, if you need to make NULL invalid in a CHECK constraint, you must do so explicitly by extending the constraint. Using the example above:

```
CHECK (DEPTNO IN (10, 20, 30)
AND DEPTNO IS NOT NULL)
```

The Firebird Null Guide has been updated to reflect changes in Firebird 2. You can download your copy from http://www.firebirdsql.org/index.php?op=doc#category_1

Improvement for External Tables

Previously, the external file linked to an external table would remain write-locked until all users are logged out. The engine will now release external table files as soon as they are no longer being referenced by user requests (allocated statements). This is a welcome improvement for those who use external tables for repeated batch imports or exports.

If an external table has been referenced by a PSQL module then the situation hasn't changed, because these modules stay in the cache once used and so the file lock remains intact.

The Firebird Book — A Reference for Database Developers

Chapter 16 Errata

Page	Erratum
290	The syntax pattern for ALTER TABLE is incorrect. It reads: `ALTER TABLE name DROP colname [, colname ...];` It should be: `ALTER TABLE name DROP colname [, DROP colname ...];`

Chapter 17
Referential Integrity

The following enhancement was introduced to simplify the creation of foreign key constraints.

Exclusive Access Rule Relaxed

Creating a FOREIGN KEY constraint no longer requires exclusive access to the database.

Neverthless, you can still expect an *Object in Use* exception if the tables affected by creating a FOREIGN KEY constraint are involved in an interesting transaction.

Chapter 18
Indexes

A raft of improvements to indexing and the options available to the optimizer appears in Firebird 2.0.

Indexing and Optimizer Enhancements

Firebird 2's new and reworked index code is very fast and tolerant of large numbers of duplicates. A 40-bit record number now included on "non leaf-level" pages is used for sorting duplicate key entries. A full reworking of the index compression algorithm has made an enormous improvement in the performance of many queries, especially key lookup on inserts/deletes with many duplicates —NULLs in foreign keys, and garbage collection, for example.

About the New Index Structures

The aims achieved by the new structures were:

- better support for deleting an index key out of many duplicates (caused slow garbage collection)
- support for bigger record numbers than 32-bits (40 bits)
- to increase the limit on index key size (1/4 page-size)

The author of the changes, Arno Brinkman, prepared a detailed description of the inner workings of the new structures, which you can read in Chapter 8 of the Firebird 2 release notes, in your /doc subdirectory.

Notable Changes to Indexing

Some of the more notable changes to indexing rules and features are described in the following pages. You should pay attention to compatibility issues that may affect what needs to be done before you finally reach the point of upgrading databases.

252-Byte Index Size Limit is Gone

The old aggregate key length limit of 252 bytes is removed. Now the limit depends on page size: the maximum size of the key in bytes is one-quarter of the page size, extending the limit to 1,024 bytes with a 4Kb page size, 2,048 bytes on 8Kb pages, and so on.

Expression Indexes

Arbitrary expressions applied to values in a row in dynamic DDL can now be indexed, allowing indexed access paths to be available for search predicates that are based on expressions. Expression indices have exactly the same features and limitations as regular indices, except that, by definition, they cannot be composed of multiple segments.

Syntax Pattern

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

Examples

1. Indexing a column on its upper-cased value:

```
CREATE INDEX IDX1 ON T1
COMPUTED BY ( UPPER(COL1 COLLATE PT_BR) );
COMMIT;

/**/
SELECT * FROM T1
WHERE UPPER(COL1 COLLATE PT_BR) = 'ÔÛÂÀ'
-- PLAN (T1 INDEX (IDX1))
```

2. Indexing on a more complex expression:

```
CREATE INDEX IDX2 ON T2
COMPUTED BY ( EXTRACT(YEAR FROM COL2) || EXTRACT(MONTH FROM COL2) );
COMMIT;

/**/
SELECT * FROM T2
ORDER BY EXTRACT(YEAR FROM COL2) || EXTRACT(MONTH FROM COL2)
-- PLAN (T2 ORDER IDX2)
```

In order to allow the engine to choose an indexed access path, the expression used in the search predicate must match *exactly* the expression used in the index declaration, otherwise the index will not be available for any retrieval or sorting operation.

Handling of Null Keys

In general, the engine has been taught to ignore NULLs during an index scan whenever it makes sense to ignore them. Previously, NULL keys were always scanned for all predicates. Starting with v.2.0, NULL keys are usually skipped before the scan begins, thus allowing faster index scans.

Null keys are now bypassed when performing checks for uniqueness. If a new key is inserted into a unique index, the engine gains a performance benefit by skiping all NULL keys before starting to check for key duplication.

The predicates IS NULL and IS NOT DISTINCT FROM still require scanning of NULL keys, thus disabling the NULL-skipping optimization.

Selectivity Maintenance per Segment

Index selectivities are now calculated and stored for each segment of an index, opening more opportunities to the optimizer for clever access path decisions in cases involving partial index matches.

For a compound index on columns (A, B, C), for example, three selectivity values will be calculated, reflecting a full index match as well as all partial matches. The effect is that the selectivity of the multi-segment index involves all of the ways the compound index could be used, i.e., those of segment A alone (as it would be if it were a single-segment index), segments A and B combined (as it would be if it were a double-segment index) and the full three-segment match (A, B, C).

The per-segment selectivity values are stored in the column RDB$STATISTICS of table RDB$INDEX_SEGMENTS. The column of the same name in RDB$INDICES is kept for compatibility and still represents the total index selectivity, that is used for a full index match.

Query Plans

Due to a bug, some previous versions of Firebird would accept user-specified plans with missing table references. A plan must refer to all tables in the query and, under the new conditions, an exception will be thrown if you attempt to use one with a missing table reference.

If you encounter an exception related to plans, e.g. Table T is not referenced in plan, it will be necessary to inspect your procedure and trigger sources and adjust the plans to make them semantically correct.

Such errors could also show up during the restore process when you are migrating databases to the new version. It will be necessary to correct these conditions in the original database, under the existing server, before you attempt to perform a backup/restore cycle.

Improved PLAN Clause

A PLAN clause optionally allows you to provide your own instructions to the engine and have it ignore the plan supplied by the optimizer. Firebird 2 enhancements allow you to specify more possible paths for the engine.

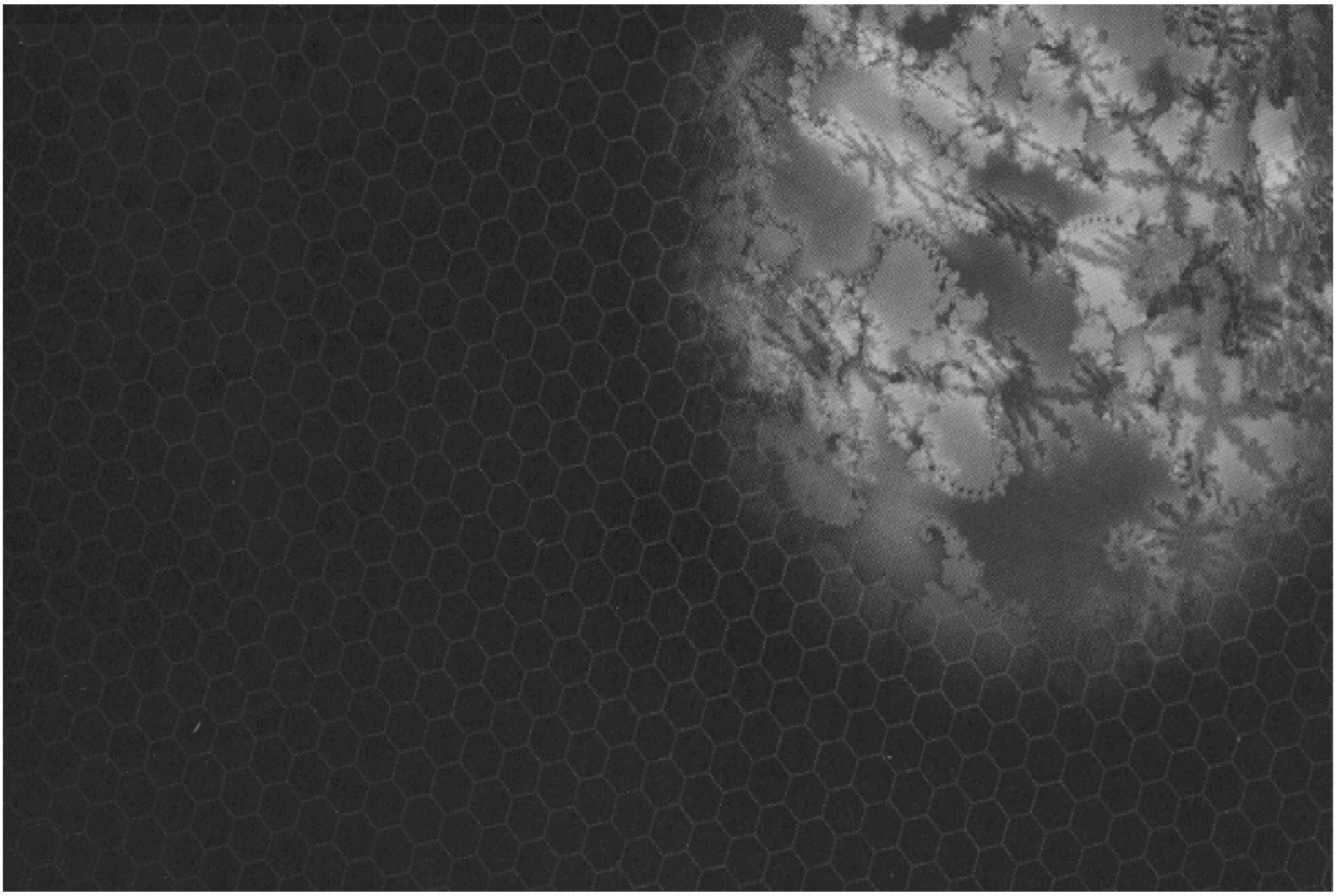
For example:

```
PLAN (A ORDER IDX1 INDEX (IDX2, IDX3))
```

For more details, refer to the topic *User-specified Query Plans* 137 in Chapter 22.

Chapter 18 Errata

Page	Erratum
338	Halfway down table 18-3, the second sentence for the Description for the entry "Next transaction" reads "The difference between the oldest active transaction and the next transaction determines when database sweeping occurs." It should read : "The difference between the oldest active transaction and the **oldest snapshot transaction** determines when database sweeping occurs."



Part Five

Firebird SQL

Topics in This Part

Chapter 19 Firebird's SQL Language

During Firebird 2 development, serious focus was placed on standards compliance. Throughout this supplement you will notice frequent references to the SQL-92 and SQL-200n (or SQL-3) standards, both for new language feature support and for enhancements to the existing language sets. Descriptions of the improvements and changes are distributed across many chapters of this supplement. The highlights and references are listed below.

In several cases, new, standards-conformant language features have been added to complement existing implementations that are peculiar to Firebird. Such parallel implementations should assist those who are writing applications layers intended to interface with multiple SQL back-ends.

Parallel implementations are not always 100 per cent interchangeable. Make a point of understanding what you might miss (or gain) from using the alternative syntax implementation.

Data Manipulation Language (DML)

Derived Tables

Support for derived tables in DSQL has been implemented according to the SQL200n standard. A derived table is a set derived from a dynamic SELECT statement, e.g. **`SELECT * FROM (SELECT...).`** Complex queries can be built by nesting or joining derived tables.

For details, see *Derived Tables* 104 in Chapter 20, *DML Queries*.

Execute Block

The SQL language extension **`EXECUTE BLOCK`** makes procedural code (PSQL) available to return values from dynamic **`SELECT`** statements. Any self-contained block of PSQL code that returns a single value can be executed in dynamic SQL as if it were a stored procedure.

For details, see Chapter 20, *DML Queries* 106.

ROWS Syntax

Used, like FIRST n..., to limit the number of rows retrieved from a select expression, this alternative syntax accords with the latest SQL standard.

For details, see Chapter 20, *DML Queries* 112.

Expressions Handling

Milliseconds Precision for CURRENT_TIMESTAMP

The context variable CURRENT_TIMESTAMP now returns milliseconds by default—in former versions it truncated sub-seconds back to seconds. You can still retrieve the truncated value if you need to, by specifying the required accuracy explicitly as a new, optional argument, e.g. CURRENT_TIMESTAMP(0).

For details, see Chapter 21, *Expressions and Predicates* 121.

Reimplemented String Search Operators

The string search operators (LIKE, STARTING WITH and CONTAINING) have been reworked to improve performance, allow null as an escape character and correct problems at segment boundaries when searching text BLOBs.

For details, see Chapter 21, *Expressions and Predicates* 120.

Run-time Checking for Concatenation Overflow

Expressions involving concatenations that could potentially overflow are no longer rejected at compile time. Instead, the length of the intended output is evaluated at run-time and will raise an overflow exception only if the actual result would exceed 32,765 bytes.

For details, see Chapter 21, *Expressions and Predicates* 120.

UDFs Can Now Handle a NULL Result

Several string functions in the external function library *ib_udf* have been upgraded to take advantage of changes that allow NULL to be interpreted correctly if it is returned by a UDF call.

For details, see Chapter 21, *Expressions and Predicates* 130. Details of where to find a script for upgrading pre-V.2 databases to work with the new *ib_udf* library can be found here 131.

Transactions

WAIT Timeout for Transactions

The blocking WAIT lock resolution mode for transactions has been extended to include provision for a finite interval for waiting. If the timeout is exceeded, a *lock_timeout* exception is raised along with the lock conflict or deadlock exception that gave rise to the wait.

The new, optional **LOCK TIMEOUT** <*value*> clause of the **SET TRANSACTION** statement activates the feature in DSQL. A corresponding new constant has been added to the transaction parameter block in the API.

For details, see Chapter 26, *Configuring Transactions* 152.

New ROLLBACK RETAIN Syntax

Firebird 2.0 adds an optional **RETAIN** clause to the DSQL **ROLLBACK** statement to make it consistent with **COMMIT [[WITH] [RETAIN]**.

For details, see Chapter 26, *Configuring Transactions* 152.

Procedural SQL (PSQL)

Explicit (Named) Cursors

Multiple named (i.e. explicit) cursors are now supported in PSQL and in DSQL **EXECUTE BLOCK** statements. More information in the chapter Explicit Cursors.

For details, see *Explicit Cursors* 162 in Chapter 29, *Developing PSQL Modules*.

PSQL Invariant Tracking Reworked

Invariant tracking in PSQL and request cloning logic were reworked to fix a number of issues with recursive procedures. The changes will affect particularly the accuracy and performance of complex expression evaluations in this environment.

For details, see Appendix XIII , *Miscellaneous* 328.

Data Definition Language (DDL)

Reworking of Updatable Views

The implementation of updatable views has been reimplemented to resolve some undocumented misbehaviour in previous versions. Because developers have sometimes taken advantage of the old anomalies, it will be important to examine legacy code carefully with respect to use of updatable views.

For details, see Chapter 24, *Views* 145.

A description of the anomalous behaviour can be found on Page 555, under the title *About "Double-dipped" Updates*.

Sequences

A sequence generator is a mechanism for generating successive exact 64-bit integer values, one at a time. The SEQUENCE object has been introduced in Firebird 2.x as a a synonym for GENERATOR, in accordance with SQL-99.

For details, see Chapter 9, *Number Types* 46.

The COMMENT Statement

The COMMENT statement has been implemented for inserting optional descriptions for various metadata objects. Comment text is stored in a BLOB of subtype TEXT.

Syntax Pattern

```
COMMENT ON DATABASE IS {'txt'|NULL};
COMMENT ON <basic_type> name IS {'txt'|NULL};
COMMENT ON COLUMN tblviewname.fieldname IS {'txt'|NULL};
COMMENT ON PARAMETER procname.parname IS {'txt'|NULL};
```

An empty literal string ('') will act as NULL since the internal code (DYN in this case) works this way with BLOBs.

Applicable Objects

The ***<basic_type>*** can be any of the following object types:

```
DOMAIN | TABLE | VIEW | PROCEDURE | TRIGGER
EXTERNAL FUNCTION | FILTER | EXCEPTION
GENERATOR | SEQUENCE | INDEX | ROLE
CHARACTER SET | COLLATION | SECURITY CLASS*
```

*not implemented, because this type is hidden.

Example

```
CREATE DOMAIN D_BOOLEAN CHAR
CHECK (VALUE IN ('T', 'F'));
COMMIT;

COMMENT ON DOMAIN D_BOOLEAN
IS 'T=True, F=False, nulls allowed';
COMMIT;
```

SET/DROP Default for Table Columns

A small enhancement was made to ALTER TABLE ALTER [COLUMN] syntax to allow setting and dropping of column defaults on existing columns.

See Chapter 16, *Tables* 80, for details.

Modify a UDF Declaration

`ALTER EXTERNAL FUNCTION` has been implemented, to enable the **`entry_point`** or the **`module_name`** to be changed when the UDF declaration cannot be dropped due to existing dependencies.

Exclusive Access Unnecessary for Creating Foreign Keys

The previous restriction preventing creation of FOREIGN KEY constraints on a database with active connections has been removed.

Neverthless, you can still expect an *Object in Use* exception if the tables affected by creating a FOREIGN KEY constraint are involved in an interesting transaction.

Reserved Words

A number of new reserved keywords were introduced and restrictions changed on some existing ones. The full list is available in Firebird's CVS tree in /doc/sql.extentions/README.keywords.

See Appendix XI, *Reserved Words* 317, for a completely revised list.

You must ensure that your DSQL statements and procedure/trigger sources are not using the new keywords as identifiers.

In a Dialect 3 database, such identifiers can be redefined using the same words, as long as the identifiers are enclosed in double-quotes. In a Dialect 1 database there is no way to retain them: they must be redefined with new, legal words.

And More...

There is much more new SQL stuff, including some important enforcements of standards that will affect legacy application code in cases where developers exploited the tolerance of older implementations in InterBase and Firebird 1.0.x to faulty syntax. Details will be found in Parts Four (DDL), Five (DML) and Seven (PSQL). Language changes and additions to transaction parameters and capabilities are in Chapter 26.

Chapter 20
DML Queries

Firebird's data manipulation language (DML) is the subset that encompasses the SELECT, INSERT, UPDATE and DELETE operations. Far-reaching enhancements in the underlying mechanisms in the engine, especially in the areas of index management and optimisation, have had flow-on effects for DML.

New additions to the facilities available in DML mean that a review of the current syntax rules and syntax for the options available for SELECT statements is timely. The topic entitled *SELECT Statement & Expression Syntax* in this chapter summarises these.

General Changes

Some changes have an effect across the board in DML syntax that may affect the validity of statements used in legacy application code and stored procedures. They include the following elements.

Stricter DSQL Parsing of Table Aliases

Alias handling and ambiguous field reference detection have been tightened considerably. Legacy applications where attention to potential ambiguities has been neglected will need to be checked and adjusted as an essential part of migrating to Firebird 2.x.

Summary of Tightened Enforcement

When an alias is provided for a table name, that alias must be used. It is no longer valid to intermix the use of a table's name and its alias.

Ambiguity checking now checks first for ambiguity at the current level of scope, making it valid in some conditions for columns to be used without qualifiers at a higher scope level.

For those with legacy software that will be disturbed by the new restrictions, a temporary workaround would be to set on the new (but short-lived) configuration parameter ***RelaxedAliasChecking*** 196.

Examples

1. When an alias is present it must be used; or no alias at all is allowed.

The following query, tolerated in FB1.5 and earlier versions, will now correctly report an error:

```
SELECT
RDB$RELATIONS.RDB$RELATION_NAME
FROM
RDB$RELATIONS R
```

The error reported will be

`Field "RDB$RELATIONS.RDB$RELATION_NAME" could not be found`.

The preferred correct usage is:

```
SELECT
  R.RDB$RELATION_NAME
FROM RDB$RELATIONS R
```

The following alternative is untidy, but it will work:

```
SELECT
  RDB$RELATION_NAME
FROM RDB$RELATIONS R
```

2. The next statement will now correctly detect the aliased FieldID from the subquery and distinguish it from the non-aliased FieldID specified for updating:

```
UPDATE TableA
SET FieldA =
  (SELECT SUM(A.FieldB) FROM TableA A
   WHERE A.FieldID = TableA.FieldID)
```

In Firebird it is possible to provide an alias in an update statement, but many other database vendors do not support it. These statement syntaxes will improve the interchangeability of Firebird's SQL with other SQL database products.

---

3. In Firebird 1.5 and earlier, the following example did not run correctly:

```
SELECT
  RDB$RELATIONS.RDB$RELATION_NAME,
  R2.RDB$RELATION_NAME
FROM RDB$RELATIONS
JOIN RDB$RELATIONS R2 ON
  (R2.RDB$RELATION_NAME = RDB$RELATIONS.RDB$RELATION_NAME)
```

If RDB$RELATIONS contained 90 records, it would return the Cartesian product (90 * 90 = 8100 records) but in Firebird 2 it will correctly return 90 records.

4. This failed in Firebird 1.5, but is possible in Firebird 2:

```
SELECT
  (SELECT RDB$RELATION_NAME FROM RDB$DATABASE)
FROM RDB$RELATIONS
```

5. As an example of how ambiguity checking has been tightened, the following query would run in Firebird 1.5 without reporting an ambiguity, but will fail in Firebird 2.x:

```
SELECT
```

```
(SELECT
FIRST 1 RDB$RELATION_NAME
FROM
RDB$RELATIONS R1
JOIN RDB$RELATIONS R2 ON
(R2.RDB$RELATION_NAME = R1.RDB$RELATION_NAME))
FROM
RDB$DATABASE
```

Multiple Hits to Same Column Now Illegal

Statements that make multiple "hits" on the same column in an INSERT or UPDATE statement are illegal syntax and will now throw an exception. Thus, a statement like

```
INSERT INTO T(A, B, A) ...
```

or

```
UPDATE T SET A = x, B = y, A = z
```

will be rejected in Firebird 2.0 and above, even though it was tolerated in InterBase and previous Firebird versions.

New DML Language Features

Common Table Expressions

A common table expression (CTE) is like a view that is defined locally within a main query at run-time. However, the engine treats a CTE like a derived table, without the intermediate materialisation of the data that occurs with views. Unlike a derived table, a CTE can be self-referencing and can be referenced multiple times in the same query. Its characteristics make it particularly useful for designing recursive queries that are easy to manage and maintain.

Benefits of CTEs

Using CTEs allows you to specify dynamic queries that are recursive:

- The engine begins execution from a non-recursive set.
- For each row evaluated in the outer set, it starts executing each recursive member one-by-one, using the current values from the outer row as parameters.
- If the currently executing instance of a recursive member produces no rows, execution loops back one level and gets the next row from the outer result set.

A recursive CTE is much less demanding on memory and CPU than an equivalent recursive stored procedure.

Syntax and Rules for CTEs

Syntax Pattern

```
select ::-
  select_expr for_update_clause lock_clause
select_expr ::-
  with_clause select_expr_body order_clause rows_clause
           | select_expr_body order_clause rows_clause
with_clause ::-
  WITH RECURSIVE with_list | WITH with_list
with_list ::-
  with_item | with_item ',' with_list
with_item ::-
  symbol_table_alias_name derived_column_list
    AS '(' select_expr ')'
select_expr_body ::-
  query_term
  | select_expr_body UNION distinct_noise query_term
           | select_expr_body UNION ALL query_term
```

A less formal representation illustrates the pattern more clearly:

```
WITH [RECURSIVE]
 CTE_A [(a1, a2, …)]
 AS ( SELECT … ),
 CTE_B [(b1, b2, …)]
 AS ( SELECT … ),
...
SELECT ...
  FROM CTE_A, CTE_B, TAB1, TAB2 ...
 WHERE ...
```

The rules vary according to whether the CTE is recursive or non-recursive.

Recursion Limit

Currently the recursion depth has hard-coded limit of 1024.

Non-Recursive CTEs

- Multiple table expressions can be defined in one query
- Any clause legal in a SELECT specification is legal in table expressions
- Table expressions can reference one another
- References between expressions should not have loops

- Table expressions can be used within any part of the main query or another table expression
- The same table expression can be used more than once in the main query
- Table expressions (as subqueries) can be used in INSERT, UPDATE and DELETE statements
- Table expressions are legal in PSQL code
- WITH statements cannot be nested

Example of a non-recursive CTE

```
WITH
  DEPT_YEAR_BUDGET AS (
    SELECT FISCAL_YEAR, DEPT_NO,
        SUM(PROJECTED_BUDGET) AS BUDGET
      FROM PROJ_DEPT_BUDGET
    GROUP BY FISCAL_YEAR, DEPT_NO
  )
SELECT D.DEPT_NO, D.DEPARTMENT,
  B_1993.BUDGET AS B_1993, B_1994.BUDGET AS B_1994,
       B_1995.BUDGET AS B_1995, B_1996.BUDGET AS B_1996
  FROM DEPARTMENT D
    LEFT JOIN DEPT_YEAR_BUDGET B_1993
      ON D.DEPT_NO = B_1993.DEPT_NO
      AND B_1993.FISCAL_YEAR = 1993
    LEFT JOIN DEPT_YEAR_BUDGET B_1994
      ON D.DEPT_NO = B_1994.DEPT_NO
      AND B_1994.FISCAL_YEAR = 1994
    LEFT JOIN DEPT_YEAR_BUDGET B_1995
      ON D.DEPT_NO = B_1995.DEPT_NO
      AND B_1995.FISCAL_YEAR = 1995
    LEFT JOIN DEPT_YEAR_BUDGET B_1996
      ON D.DEPT_NO = B_1996.DEPT_NO
      AND B_1996.FISCAL_YEAR = 1996
  WHERE EXISTS (
    SELECT * FROM PROJ_DEPT_BUDGET B
    WHERE D.DEPT_NO = B.DEPT_NO);
```

Recursive CTEs

A recursive CTE is self-referencing (has a reference to itself) and is a UNION of recursive and non-recursive members, with these restrictions:

- At least one non-recursive member (anchor) must be present
- Non-recursive members are placed *first* in the UNION

- Recursive members are separated from anchor members and from one another with UNION ALL clauses, i.e.,

  non-recursive member (anchor)

  UNION [ALL | DISTINCT]

  non-recursive member (anchor)

  UNION [ALL | DISTINCT]

  non-recursive member (anchor)

  UNION ALL

  recursive member

  UNION ALL

  recursive member

- References between CTEs should not have loops
- Aggregates (DISTINCT, GROUP BY, HAVING) and aggregate functions (SUM, COUNT, MAX etc) are not allowed in recursive members
- A recursive member can have only one reference to itself and only in a FROM clause
- A recursive reference cannot participate in an outer join

Example of a recursive CTE

```
WITH RECURSIVE
  DEPT_YEAR_BUDGET AS
  (
    SELECT FISCAL_YEAR, DEPT_NO,
        SUM(PROJECTED_BUDGET) AS BUDGET
      FROM PROJ_DEPT_BUDGET
    GROUP BY FISCAL_YEAR, DEPT_NO
  ),
  DEPT_TREE AS
  (
    SELECT DEPT_NO, HEAD_DEPT, DEPARTMENT,
        CAST('' AS VARCHAR(255)) AS INDENT
      FROM DEPARTMENT
     WHERE HEAD_DEPT IS NULL
    UNION ALL
    SELECT D.DEPT_NO, D.HEAD_DEPT, D.DEPARTMENT,
    H.INDENT || '  '
      FROM DEPARTMENT D
      JOIN DEPT_TREE H
        ON D.HEAD_DEPT = H.DEPT_NO
```

```
 )
 SELECT D.DEPT_NO,
D.INDENT || D.DEPARTMENT AS DEPARTMENT,
B_1993.BUDGET AS B_1993,
B_1994.BUDGET AS B_1994,
B_1995.BUDGET AS B_1995,
B_1996.BUDGET AS B_1996
 FROM DEPT_TREE D
   LEFT JOIN DEPT_YEAR_BUDGET B_1993
     ON D.DEPT_NO = B_1993.DEPT_NO
     AND B_1993.FISCAL_YEAR = 1993
   LEFT JOIN DEPT_YEAR_BUDGET B_1994
     ON D.DEPT_NO = B_1994.DEPT_NO
     AND B_1994.FISCAL_YEAR = 1994
   LEFT JOIN DEPT_YEAR_BUDGET B_1995
     ON D.DEPT_NO = B_1995.DEPT_NO
     AND B_1995.FISCAL_YEAR = 1995
   LEFT JOIN DEPT_YEAR_BUDGET B_1996
     ON D.DEPT_NO = B_1996.DEPT_NO
     AND B_1996.FISCAL_YEAR = 1996
```

Derived Tables

Dynamic SQL in Firebird 2 supports *derived tables* as defined by SQL200n standards. A derived table is a form of "virtual table" that is returned to a FROM clause as a set derived from a dynamic subquery. Derived tables can be nested, if required, to build complex queries and they can be involved in joins as though they were normal tables or views.

Syntax Pattern

```
SELECT
<select list>
FROM
<table reference list>

<table reference list> ::= <table reference> [{<comma> <table reference>}...]

<table reference> ::=
<table primary>
| <joined table>

<table primary> ::=
<table> [[AS] <correlation name>]
| <derived table>
```

```
<derived table> ::=
<query expression> [[AS] <correlation name>]
[<left paren> <derived column list> <right paren>]

<derived column list> ::= <column name> [{<comma> <column name>}...]
```

Examples

a) Simple derived table:

```
SELECT * FROM
(SELECT
    RDB$RELATION_NAME,
    RDB$RELATION_ID
  FROM RDB$RELATIONS) AS R (RELATION_NAME, RELATION_ID)
```

b) Aggregate on a derived table which also contains an aggregate

```
SELECT
DT.FIELDS,
Count(*)
FROM
(SELECT
  R.RDB$RELATION_NAME,
  Count(*)
  FROM RDB$RELATIONS R
  JOIN RDB$RELATION_FIELDS RF
  ON (RF.RDB$RELATION_NAME = R.RDB$RELATION_NAME)
  GROUP BY R.RDB$RELATION_NAME) AS DT (RELATION_NAME, FIELDS)
GROUP BY
DT.FIELDS
```

c) UNION and ORDER BY

```
SELECT DT.*
FROM
  (SELECT
    R.RDB$RELATION_NAME,
    R.RDB$RELATION_ID
    FROM RDB$RELATIONS R
  UNION ALL
  SELECT
    R.RDB$OWNER_NAME,
    R.RDB$RELATION_ID
   FROM RDB$RELATIONS R
```

```
  ORDER BY 2) AS DT
WHERE
  DT.RDB$RELATION_ID <= 4
```

- Every column in the derived table must have a name. Unnamed expressions like constants should be added with an alias or the column list should be used.
- The number of columns in the column list should be the same as the number of columns from the query expression.
- The optimizer can handle a derived table very efficiently. However, if the derived table is involved in an inner join and contains a subquery, then no join order can be made.

EXECUTE BLOCK Statement

The SQL language extension EXECUTE BLOCK makes "dynamic PSQL" available to SELECT specifications. It has the effect of allowing a self-contained block of PSQL code to be executed in dynamic SQL as if it were a stored procedure.

Syntax pattern

```
EXECUTE BLOCK [ (param datatype = ?, param datatype = ?, ...) ]
[ RETURNS (param datatype, param datatype, ...) ]
AS
[DECLARE VARIABLE var datatype; ...]
BEGIN
...
END
```

For the client, the call *isc_dsql_sql_info()* with the parameter *isc_info_sql_stmt_type* returns

- *isc_info_sql_stmt_select* if the block has output parameters. The semantics of a call is similar to a SELECT query: the client has a cursor open, can fetch data from it, and must close it after use.
- *isc_info_sql_stmt_exec_procedure* if the block has no output parameters. The semantics of a call is similar to an EXECUTE query: the client has no cursor and execution continues until it reaches the end of the block or is terminated by a SUSPEND.

The client should preprocess only the head of the SQL statement or use '?' instead of ':' as the parameter indicator because, in the body of the block, there may be references to local variables or arguments with a colon prefixed.

Example

The user SQL is

```
EXECUTE BLOCK (X INTEGER = :X)
RETURNS (Y VARCHAR)
```

```
AS
DECLARE V INTEGER;
BEGIN
  INSERT INTO T(...) VALUES (... :X ...);
  SELECT ... FROM T INTO :Y;
  SUSPEND;
END
```

The preprocessed SQL is

```
EXECUTE BLOCK (X INTEGER = ?)
RETURNS (Y VARCHAR)
AS
DECLARE V INTEGER;
BEGIN
  INSERT INTO T(...) VALUES (... :X ...);
  SELECT ... FROM T INTO :Y;
  SUSPEND;
END
```

UPDATE OR INSERT Statement

2.1 The UPDATE OR INSERT syntax has been introduced to enable a record to be either updated or inserted, according to whether or not it already exists (checked with IS NOT DISTINCT). The statement is available in both DSQL and PSQL.

Syntax Pattern

```
UPDATE OR INSERT INTO <table or view> [(<column_list>)]
    VALUES (<value_list>)
    [MATCHING <column_list>]
    [RETURNING <column_list> [INTO <variable_list>]]
```

Examples

```
UPDATE OR INSERT INTO T1 (F1, F2)
    VALUES (:F1, :F2);

UPDATE OR INSERT INTO EMPLOYEE (ID, NAME)
    VALUES (:ID, :NAME)
    RETURNING ID;

UPDATE OR INSERT INTO T1 (F1, F2)
    VALUES (:F1, :F2)
    MATCHING (F1);

UPDATE OR INSERT INTO EMPLOYEE (ID, NAME)
    VALUES (:ID, :NAME)
```

```
RETURNING OLD.NAME;
```

---

1. When MATCHING is omitted, the existence of a primary key is required.
2. INSERT and UPDATE permissions are needed on <table or view>.
3. If the RETURNING clause is present, then the statement is described as isc_info_sql_stmt_exec_procedure by the API; otherwise, it is described as isc_info_sql_stmt_insert.
4. A "multiple rows in singleton select" error will be raised if the RETURNING clause is present and more than one record matches the search condition.

MERGE Statement

The MERGE syntax enables a record to be either updated or inserted, depending on the result of test, which may be an existence condition determined from another set.

The statement is available in both DSQL and PSQL.

Syntax Pattern

```
<merge statement> ::=
  MERGE
  INTO <table or view> [ [AS] <correlation name> ]
  USING <table or view or derived table> [ [AS] <correlation name> ]
  ON <match-condition>
  [ <merge when matched> ]
  [ <merge when not matched> ]

  <merge when matched> ::=
    WHEN MATCHED THEN
      UPDATE SET <list-of-assignments>

  <merge when not matched> ::=
    WHEN NOT MATCHED THEN
      INSERT ( <column list> )
      VALUES ( <mapped-list-of-values> )
```

A right join is made between the INTO and USING tables using the **`<match-condition>`**. UPDATE occurs when a record exists in the left-hand (INTO) set, otherwise INSERT is called. If no record is returned in the join, INSERT is not called.

- At least one of **`<merge when matched>`** and **`<merge when not matched>`** should be specified
- Each should be specified not more than once.

Example

```
MERGE INTO CUSTOMER C
  USING (SELECT * FROM CONTACTS
    WHERE STATUS = 'X') CT
  ON (C.CUST_ID = CT.CUST_ID)
  WHEN MATCHED THEN
    UPDATE SET
      CONFIRM_DATE = CT.CONFIRM_DATE
  WHEN NOT MATCHED THEN
    INSERT (CUST_ID, LAST_NAME, FIRST_NAME)
     VALUES (CT.CUST_ID, CT.LAST_NAME, CT.FIRST_NAME)
```

The RETURNING Clause

The purpose of this SQL enhancement is to enable the set, or a subset, of the column values stored into a table as a result of an INSERT, UPDATE OR DELETE statement to be returned to the client.

The most popular usage of RETURNING ... is for retrieving the value generated for a primary key inside a BEFORE-trigger. Because the RETURNING clause is designed to return a singleton set in on completing an operation on a single record, it is not valid to specify the clause in a statement that inserts, updates or deletes multiple records.

It is optional and is available in both DSQL and PSQL, although the rules differ slightly.

In DSQL, the execution of the operation itself and the return of the set occur in a single protocol round trip. The set is always returned from a DSQL request if specified, even if the operation has no effect on any record. In its current implementation, therefore, the potential exists to return an "empty" set.

Syntax Patterns

```
INSERT INTO ... VALUES (...)
    [RETURNING <column_list> [INTO <variable_list>]]

INSERT INTO ... SELECT ...
    [RETURNING <column_list> [INTO <variable_list>]]

UPDATE OR INSERT INTO ... VALUES (...) ...
    [RETURNING <column_list> [INTO <variable_list>]]

UPDATE ... [RETURNING <column_list> [INTO <variable_list>]]

DELETE FROM ...
    [RETURNING <column_list> [INTO <variable_list>]]
```

DSQL:

```
INSERT INTO ... VALUES (...) [RETURNING <column_list>]
```

PSQL:

```
INSERT INTO ... VALUES (...) [RETURNING <column_list> INTO <variable_list>]
```

- The INTO part to assign local variables is allowed in PSQL only and will be rejected in DSQL.
- Returning values from cursor-based inserts **`(INSERT INTO ... SELECT ... RETURNING ...)`** is not supported.
- Any explicit record change (update or delete) performed by AFTER triggers is ignored by the **`RETURNING`** clause.
- OLD and NEW context variables can be used in the RETURNING clause of UPDATE and INSERT OR UPDATE statements.
- In UPDATE and INSERT OR UPDATE statements, field references that are unqualified or qualified by table name or relation alias are resolved to the value of the corresponding NEW context variable.

Examples

1. Returns the values of two replaceable parameters from a DSQL statement:

```
INSERT INTO T1 (F1, F2)
VALUES (:F1, :F2)
RETURNING F1, F2;
```

2. Returns the value of a column named ID that is populated by a trigger calling a SEQUENCE (GENERATOR) into a PSQL variable:

```
INSERT INTO T2 (F1, F2)
VALUES (1, 2)
RETURNING ID INTO :PK;

-----------------------

INSERT INTO T1 (F1, F2)
    VALUES (:F1, :F2)
      RETURNING F1, F2 INTO :V1, :V2;

INSERT INTO T2 (F1, F2)
    VALUES (1, 2)
      RETURNING ID INTO :PK;

DELETE FROM T1
  WHERE F1 = 1
    RETURNING F2;

UPDATE T1
    SET F2 = F2 * 10
```

```
RETURNING OLD.F2, NEW.F2;
```

API

If the RETURNING clause is present, then the statement is described as *isc_info_sql_stmt_exec_procedure* by the API (instead of *isc_info_sql_stmt_insert*). Existing connectivity drivers would be expected to support this feature.

INSERT with Defaults

It is now possible to INSERT without supplying values, if *Before Insert* triggers and/or declared defaults are available for every column and none is dependent on the presence of any supplied 'NEW' value.

Example

```
INSERT INTO <table>
  DEFAULT VALUES
  [RETURNING <values>]
```

ROWS Syntax

ROWS syntax is used to limit the number of rows retrieved from a select expression. For an uppermost-level select statement, it would specify the number of rows to be returned to the host program.

A more understandable alternative to the FIRST/SKIP clauses, the ROWS syntax accords with the latest SQL standard and brings some extra benefits. It can be used in unions, any kind of subquery and in UPDATE or DELETE statements.

It is available in both DSQL and PSQL.

Syntax Pattern

```
SELECT ...
[ORDER BY <expr_list>]
ROWS <expr1> [TO <expr2>]
```

Examples

1.

```
SELECT * FROM T1
UNION ALL
SELECT * FROM T2
ORDER BY COL
ROWS 10 TO 100
```

2.

```
SELECT COL1, COL2,
( SELECT COL3 FROM T3 ORDER BY COL4 DESC ROWS 1 )
```

```
FROM T4
```

3.

```
DELETE FROM T5
ORDER BY COL5
ROWS 1
```

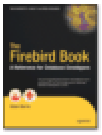
- When <expr2> is omitted, then ROWS <expr1> is semantically equivalent to FIRST <expr1>.
- When both <expr1> and <expr2> are used, then ROWS <expr1> TO <expr2> means the same as FIRST (<expr2> - <expr1> + 1) SKIP (<expr1> - 1)
- There is nothing that is semantically equivalent to a SKIP clause used without a FIRST clause.

ROLLBACK RETAIN Syntax

The ROLLBACK RETAIN statement for transactions is now supported in DSQL.

A “rollback retaining” feature was introduced in InterBase 6.0, but this rollback mode could be used only via an API call to isc_rollback_retaining(). By contrast, “commit retaining” could be used either via an API call to isc_commit_retaining() or by using a DSQL COMMIT RETAIN statement.

Firebird 2.0 adds an optional RETAIN clause to the DSQL ROLLBACK statement to make it consistent with COMMIT [RETAIN]. Its effect is to cause a rollback of the transaction without releasing the snapshot view commanded by the transaction.



Syntax Pattern

The syntax for ROLLBACK RETAIN follows that of COMMIT RETAIN—see Chapter 27, *Programming with Transactions*, at pages 546 ff.

Enhancements to NULL Logic

Some useful features involving NULL in DSQL have been implemented.

IS [NOT] DISTINCT FROM

The new **`IS [NOT] DISTINCT FROM`** predicate has been introduced that treates two NULL operands as equal. It behaves exactly like the equality/inequality predicates, but, instead of testing for equality, it tests whether one operand is distinct from the other.

Thus, **`IS NOT DISTINCT FROM`** returns True if the two operands are both NULL (or evaluate as NULL), as well as when both operands are not null and are equal.

It is available in both DSQL and PSQL.

Syntax Pattern

```
<value> IS [NOT] DISTINCT FROM <value>
```

Examples

1.

```
SELECT T1.* FROM T1
JOIN T2
ON T1.NAME IS NOT DISTINCT FROM T2.NAME;
```

2.

```
SELECT T.* FROM T
WHERE T.MARK IS DISTINCT FROM 'test';
```

- Because the DISTINCT predicate considers that two NULLs are not distinct, it never evaluates to the truth value UNKNOWN. Like the IS [NOT] NULL predicate, it can only be True or False.
- The NOT DISTINCT predicate can be optimized using an index, if one is available.

Relaxed Operator Rules for NULL Comparisons

A NULL literal can now be treated as a value in all expressions without returning a syntax error. You may now specify expressions such as

```
A = NULL
B > NULL
A + NULL
B || NULL
```

All such expressions evaluate to NULL. The change does not alter the semantics of the engine with regard to nullness, it simply relaxes the syntax restrictions a little. So, for example, the test A = NULL will return FALSE whether A is NULL or some value, whereas A IS NULL will return TRUE if A is NULL and FALSE otherwise. The only difference now is that A = NULL does not cause a syntax error.

NULLs Ordering Now Standards-Compliant

Placement of nulls in an ordered set has been changed to accord with the SQL standard that null ordering be consistent, i.e. if ASC[ENDING] order puts them at the bottom, then DESC[ENDING] puts them at the top; or vice-versa. This applies only to databases created under the new on-disk structure, since it needs to use the index changes in order to work.

If you override the default nulls placement, no index can be used for sorting. That is, no index will be used for an ASCENDING sort if NULLS LAST is specified, nor for a DESCENDING sort if NULLS FIRST is specified.

Examples

```
Database: proc.fdb
SQL> create table gnull(a int);
SQL> insert into gnull values(null);
SQL> insert into gnull values(1);
SQL> select a from gnull order by a;
A
============
<null>
1

SQL> select a from gnull order by a asc;
A
============
<null>
1

SQL> select a from gnull order by a desc;
A
============
1
<null>

SQL> select a from gnull order by a asc nulls first;
A
============
<null>
1

SQL> select a from gnull order by a asc nulls last;
A
============
1
<null>

SQL> select a from gnull order by a desc nulls last;
A
```

```
============
1
<null>

SQL> select a from gnull order by a desc nulls first;
A
============
<null>
1
```

Plans for Updates and Deletes

Users can now specify explicit plans for UPDATE/DELETE statements in order to optimize them manually. It is also possible to limit the number of affected rows with a ROWS clause, optionally used in combination with an ORDER BY clause to have a sorted recordset for the operation.

Syntax Pattern

```
UPDATE ... SET ... WHERE ...
[PLAN <plan items>]
[ORDER BY <value list>]
[ROWS <value> [TO <value>]]
```

or

```
DELETE ... FROM ...
[PLAN <plan items>]
[ORDER BY <value list>]
[ROWS <value> [TO <value>]]
```

SELECT Statement & Expression Syntax

by Dmitry Yemanov, Firebird Project Lead

A *SELECT statement* is used to return data to the caller (PSQL module or the client program)

SELECT expressions retrieve portions of data to populate columns that could be in the final result set or in any of the intermediate sets.

SELECT expressions are also known as *subqueries*.

Syntax rules

```
<select statement> ::=
<select expression> [FOR UPDATE] [WITH LOCK]
<select expression> ::=
<query specification> [UNION [{ALL | DISTINCT}] <query specification>]
<query specification> ::=
SELECT [FIRST <value>] [SKIP <value>] <select list>
FROM <table expression list>
WHERE <search condition>
GROUP BY <group value list>
HAVING <group condition>
PLAN <plan item list>
ORDER BY <sort value list>
ROWS <value> [TO <value>]
<table expression> ::=
<table name> | <joined table> | <derived table>
<joined table> ::=
{<cross join> | <qualified join>}
<cross join> ::=
<table expression> CROSS JOIN <table expression>
<qualified join> ::=
<table expression> [{INNER | {LEFT | RIGHT | FULL} [OUTER]}]
  JOIN <table expression>
  ON <join condition>
<derived table> ::=
'(' <select expression> ')'
```

Semantics

- FOR UPDATE mode and row locking can be applied to a final dataset—they are not available to a subquery
- Unions are allowed inside any subquery
- The clauses FIRST, SKIP, PLAN, ORDER BY, ROWS are allowed for any subquery

- Either FIRST/SKIP or ROWS is allowed, but a syntax error is thrown if you try to mix the syntaxes
- An INSERT statement accepts a select expression to define a set to be inserted into a table. Its SELECT part supports all the features defined for select statements/expressions
- UPDATE and DELETE statements are always based on an implicit cursor iterating through its target table and limited with the WHERE clause. You may also specify the final parts of the select expression syntax to limit the number of affected rows or optimize the statement.
- Clauses allowed at the end of UPDATE/DELETE statements are PLAN, ORDER BY and ROWS.

Chapter 21 Expressions and Predicates

Firebird 2 brings many new implementations and improvements to enrich both its capability to manipulate and derive data and its conformance with standards.

In many cases, improvements in the behaviour of existing expression elements are effects of performance-enhancing changes in indexing and the optimizer routines. A few conditions exist where repairing the logical effects of an expression may result in loss of performance. One such is discussed next.

Changes to Some Existential Predicates

Existential predicates, a.k.a. existence testers, are logical operators that test the conditions of a set defined by a subquery and return a Boolean result. EXISTS, IN and ALL are some examples of existential predicates. The set defined by the subquery is often referred to as “the inner table” of the predicate.

Firebird and, before that, InterBase, produced incorrect results for the ALL and NOT IN predicates that were hard to detect. That problem has been corrected in Firebird 2.0.

The change means that

- those predicates now return a result that is reliably correct—and consequently may affect the logical assumptions made by legacy application code or PSQL modules that used them
- indexes on the inner tables cannot be used and **performance may be significantly slower** than it was for the same query in InterBase and previous versions of Firebird

---

NOT EXISTS is approximately equivalent to NOT IN and will allow Firebird to use indexes.

For more information about these predicators, see *Existential Predicates* in Chapter 21, page 400.

Context Variables and Functions

A number of new facilities has been added to extend the context information that can be retrieved.

Time and Date/Time Variables

Milliseconds

Milliseconds precision has been enabled in both DSQL and PSQL for

- the **CURRENT_TIMESTAMP** context variable
- the timestamp literal **'NOW'**
- the **CURRENT_TIME** context variable

Hundredths and Tenths of Seconds

It is also possible to retrieve the sub-second part of **CURRENT_TIME** and **CURRENT_TIMESTAMP** in hundredths, tenths or as '.000' (full seconds only).

The defaults are milliseconds precision for **CURRENT_TIMESTAMP** and full seconds precision for **CURRENT_TIME**.

The maximum possible precision is 3 which means accuracy of 1/1000 second (one millisecond). This accuracy may be improved in the future versions.

Syntax Pattern

```
CURRENT_TIME [(<seconds precision>)]
CURRENT_TIMESTAMP [(<seconds precision>)]
```

Examples

1. Retrieve the current time with zero in the sub-seconds part:

```
SQL> SELECT CURRENT_TIME(0) FROM RDB$DATABASE;

CURRENT_TIME
=============
22:53:22.0000
```

Notice that, seconds precision being the default for **CURRENT_TIME**, we would have the same result using **SELECT CURRENT_TIME FROM RDB$DATABASE**.

2. Retrieve current time with milliseconds:

```
SQL> SELECT CURRENT_TIME(3) FROM RDB$DATABASE;

 CURRENT_TIME
=============
22:53:38.6380
```

3. Retrieve current timestamp with milliseconds:

```
SQL> SELECT CURRENT_TIMESTAMP FROM RDB$DATABASE;
```

```
        CURRENT_TIMESTAMP
=========================
2007-02-24 22:56:28.1620
```

4. Retrieve current timestamp with tenths of seconds:

```
SQL> select current_timestamp(2) from rdb$database;

        CURRENT_TIMESTAMP
=========================
2007-02-24 22:56:53.9400
```

Set/Get Functions for Context Variables

Values of context variables can now be obtained using the new system functions **RDB$GET_CONTEXT** and **RDB$SET_CONTEXT**. These are built-in functions that give access through SQL to some information about the current connection and current transaction. They also provide a mechanism to retrieve user context data and associate it with the transaction or connection.

Syntax Pattern

```
RDB$SET_CONTEXT( <namespace>, <variable>, <value> )

RDB$GET_CONTEXT( <namespace>, <variable> )
```

These functions are really a form of external function that exists inside the database intead of being called from a dynamically loaded library. The following declarations are made automatically by the engine at database creation time:

Declarations

```
DECLARE EXTERNAL FUNCTION RDB$GET_CONTEXT
VARCHAR(80),
VARCHAR(80)
RETURNS VARCHAR(255) FREE_IT;

DECLARE EXTERNAL FUNCTION RDB$SET_CONTEXT
VARCHAR(80),
VARCHAR(80),
VARCHAR(255)
RETURNS INTEGER BY VALUE;
```

Using the Context Functions

RDB$SET_CONTEXT and RDB$GET_CONTEXT set and retrieve the current value of a context variable. Groups of context variables with similar properties are identified by n*amespace identifiers*.

The namespace determines the usage rules, such as whether the variables may be read and written to, and by whom.

Namespace and variable names are case-sensitive.

RDB$GET_CONTEXT

RDB$GET_CONTEXT retrieves current value of a variable. If the variable does not exist in the namespace, the function returns NULL.

RDB$SET_CONTEXT

RDB$SET_CONTEXT sets a value for specific variable, if it is writable. The function returns a value of 1 if the variable existed before the call and 0 otherwise.

Pre-defined Namespaces

A fixed number of pre-defined namespaces is available:

USER_SESSION

Offers access to session-specific user-defined variables. You can define and set values for variables with any name in this context.

USER_TRANSACTION

Offers similar possibilities for individual transactions.

SYSTEM

Provides read-only access to the following variables:

Variable	**Description**
NETWORK_PROTOCOL	The network protocol used by client to connect. Currently used values: “TCPv4”, “WNET”, “XNET” and NULL.
CLIENT_ADDRESS	The wire protocol address of the remote client, represented as a string. The value is an IP address in form “xxx.xxx.xxx.xxx “for TCPv4 protocol; the local process ID for XNET protocol; and NULL for any other protocol.
DB_NAME	Canonical name of the current database. It is either the alias name (if connection via file names is disallowed DatabaseAccess = NONE) or, otherwise, the fully expanded database file name.

ISOLATION_LEVEL	The isolation level of the current transaction. The returned value will be one of “READ COMMITTED”, “SNAPSHOT”, “CONSISTENCY".
TRANSACTION_ID	The numeric ID of the current transaction. The returned value is the same as would be returned by the context variable CURRENT_TRANSACTION.
SESSION_ID	The numeric ID of the current session. The returned value is the same as would be returned by the context variable CURRENT_CONNECTION.
CURRENT_USER	The current user. The returned value is the same as would be returned by the context variable CURRENT_USER or the predefined variable USER.
CURRENT_ROLE	Current role for the connection. Returns the same value as the context variable CURRENT_ROLE.
ENGINE_VERSION	Version of the Firebird Server engine.

To delete a variable from a context, set its value to NULL.

The number of variables stored for each transaction or session context is limited to 1000 to avoid DoS attacks against the Firebird Server.

Example of Use

```
set term ^;
create procedure set_context(User_ID varchar(40), Trn_ID integer) as
begin
RDB$SET_CONTEXT('USER_TRANSACTION', 'Trn_ID', Trn_ID);
RDB$SET_CONTEXT('USER_TRANSACTION', 'User_ID', User_ID);
end ^

create table journal (
jrn_id integer not null primary key,
jrn_lastuser varchar(40),
jrn_lastaddr varchar(255),
jrn_lasttransaction integer
)^

CREATE TRIGGER UI_JOURNAL FOR JOURNAL AFTER INSERT OR UPDATE
```

```
as
begin
new.jrn_lastuser = rdb$get_context('USER_TRANSACTION', 'User_ID');
new.jrn_lastaddr = rdb$get_context('SYSTEM', 'CLIENT_ADDRESS');
new.jrn_lasttransaction = rdb$get_context('USER_TRANSACTION', 'Trn_ID');
end ^
commit ^
execute procedure set_context('freeuser', 1) ^

insert into journal(jrn_id) values(0) ^
set term ;^
```

Since rdb$set_context returns 1 or zero, it can be made to work with a simple SELECT statement.

```
SQL> select rdb$set_context('USER_SESSION', 'Philippe', 'fr')
CNT> from rdb$database;

RDB$SET_CONTEXT
===============
0
```

The returned value 0 means it was not defined when we read it. We have set it to 'fr' now, though:

```
SQL> select rdb$set_context('USER_SESSION', 'Philippe', 'be')
CNT> from rdb$database;

RDB$SET_CONTEXT
===============
1
```

The returned value 1 means it was defined already. Now we have changed it to 'be':

```
SQL> select rdb$set_context('USER_SESSION', 'Philippe', NULL)
CNT> from rdb$database;

RDB$SET_CONTEXT
===============
1
```

The return value 1 says it existed before. Now, we have made it undefined by changing it to NULL:

```
SQL> select rdb$set_context('USER_SESSION', 'Philippe', NULL)
CNT> from rdb$database;

RDB$SET_CONTEXT
===============
0
```

Now that it is NULL, it is undefined (deleted), so the result is 0.

Other New Functions

Many new internal functions were added in Firebird 2.1, including the internal implementation and enhancement of a large number of the functions hitherto available only as external functions (UDFs). They are fully documented in Appendix I. 242

The LIST() Function

The LIST() function takes a scalar value expression and returns all the found values, aggregated into a comma-delimited list, as a text BLOB. An optional second argument allows a different delimiter to be specified.

Syntax Pattern

The syntax pattern is:

```
LIST ( [ ALL | DISTINCT ] <value expression> [, <delimiter value> ] )

<delimiter value> ::=
      { <string literal> | <parameter> | <variable> }
```

Rules

1. If neither ALL nor DISTINCT is specified, ALL is implied.
2. If <delimiter value> is omitted, a comma is used to separate the concatenated values.

Numeric and date/time values are implicitly converted to strings during evaluation.

The result is always a BLOB of subtype TEXT except where the output is formed from a BLOB column of a different subtype.

The output order of the listed values depends on the order of the set from which the list is extracted, i.e., there is no way to specify an output order explicitly.

Examples

```
SELECT LIST(CL.CUST_NO, '; ')
FROM (SELECT CUSTOMER FROM CUSTOMER
        ORDER BY 1) AS CL
```

The output:

CLIST
3D-Pad Corp.; Anini Vacation Rentals; Buttle, Griffith and Co.; Central Bank; DT Syster

ns, LTD.; Dallas Technologies; DataServe International; Dynamic Intelligence Corp; Dyn

In the next example, we make use of a new feature whereby a TEXT BLOB can be cast as a string type, to show how handy this function can be when we exploit its aggregating talents:

```
SELECT
  DEPT_NO,
  CAST(LIST(LAST_NAME) AS VARCHAR(500)) AS ALIST
FROM EMPLOYEE
  GROUP BY 1
```

The output:

DEPT NO	ALIST
000	Lee,Bender
100	MacDonald,Yanowski
110	Baldwin,Leung
115	Ichida,Yamamoto
120	Bennet,Reeves,Stansbury
121	Osborne
123	Glon
125	Ferrari
130	Lambert,Weston
140	Sutherland
180	Johnson,Nordstrom
600	Nelson,Brown
621	Young,Ramanathan,Bishop,Green
622	Forest,Burbank,Guckenheimer
623	Young,De Souza,Phong,Parker,Johnson
670	O'Brien,Cook
671	Papadopoulos,Fisher,Page
672	Williams,Montgomery
900	Hall,Steadman

Fetching a Sequence Value in an Expression

Complementing the introduction of the *SQL-99-compliant syntax for creating sequences* 46 for generating unique BigInt series comes a new functional expression syntax to generate and return the next value in a sequence.

NEXT VALUE FOR Syntax

The SQL-99 compliant **NEXT VALUE FOR *<sequence_name>*** expression syntax is synonymous with **GEN_ID(*<generator-name>*,1)**.

Examples

1. Calling the GEN_ID() function:

```
SELECT GEN_ID(S_ACCOUNT_ID, 1) FROM RDB$DATABASE;
```

2. Using the **NEXT VALUE FOR** expression:

```
INSERT INTO ACCOUNT (ID, NAME)
VALUES (NEXT VALUE FOR S_ACCOUNT_ID, 'Acme Software Ltd');
```

Currently, increment ("step") values not equal to 1 (one) can be achieved only by calling the **GEN_ID()** function. Future versions are expected to provide full support for SQL-99 sequence generators, which allows the required increment values to be specified at the DDL level.

Unless there is a vital need to use a step value that is not 1, use of a **NEXT VALUE FOR** value expression instead of the **GEN_ID()** function is recommended.

GEN_ID(<name>, 0) allows you to retrieve the current sequence value, but it should never be used in insert/update statements, as it produces a high risk of uniqueness violations in a concurrent environment.

IIF() Expressions

Syntax was added to support expressions predicated by **IIF (<search_condition>, <value1>, <value2>),** as a shortcut for

```
CASE
  WHEN <search_condition> THEN <value1>
ELSE <value2>
END
```

It returns the value of the first sub-expression if the given search condition evaluates to TRUE, otherwise it returns a value of the second sub-expression.

Example

The following example returns the absolute value of the column or variable *VAL*

```
SELECT IIF(VAL >= 0, VAL, -VAL) FROM OPERATION
```

In the example, if *VAL* is null, then null will be returned.

Improvements

CAST() Hints

The infamous "Datatype unknown" error when attempting some castings has been eliminated. It is now possible to use CAST to advise the engine about the data type of a parameter.

Example

```
SELECT CAST(? AS INT) FROM RDB$DATABASE
```

SUBSTRING() Function Enhanced

The built-in function SUBSTRING() can now take arbitrary expressions in its parameters.

Formerly, the inbuilt SUBSTRING() function accepted only constants as its second and third arguments (start position and length, respectively). Now, the arguments can be anything that resolves to a value, including host parameters, function results, expressions, subqueries, etc.

Thanks to the SQL standards committee, the length of the resulting column is the same as the length of the first argument. This means that, in the following

```
x = varchar(50);
substring(x from 1 for 1);
```

the new column has a length of 50, not 1.

2.1 Indexed Reads for MIN() and MAX()

Indexed MIN/MAX aggregates would produce three indexed reads instead of the expected single read. So, with an ASC index on the non-nullable COL, the query

```
SELECT MIN(COL) FROM TAB
```

should be completely equivalent, to

```
SELECT FIRST 1 COL FROM TAB
ORDER BY 1 ASC
```

with both performing a single record read. However, formerly, the first query required three indexed reads while the second one required just the expected single read.

Now, they both resolve to a single read.

The same optimization applies to the MAX() function when mapped to a DESC index.

Improvements for String Search Operators

1. The operators now work correctly with BLOBs of any size. Issues with only the first segment being searched and with searches missing matches that straddle segment boundaries are now gone.
2. Pattern matching now uses a single-pass Knuth-Morris-Pratt algorithm, improving performance when complex patterns are used.
3. The engine no longer crashes when NULL is used as ESCAPE character for LIKE

Run-time Checking for Concatenation Overflow

Compile-time checking for concatenation overflow has been replaced by run-time checking.

From Firebird 1.0 onward, concatenation operations have been checked for the possibility that the resulting string might exceed the string length limit of 32,000 bytes, i.e. overflow. This check was performed during the statement prepare, using the declared operand sizes and would throw an error for an expression such as:

```
CAST('qwe' AS VARCHAR(30000)) || CAST('rty' AS VARCHAR(30000))
```

From Firebird 2.0 onward, this expression throws only a warning at prepare time and the overflow check is repeated at runtime, using the sizes of the actual operands. The result is that our example will be executed without errors being thrown. The isc_concat_overflow exception is now thrown only for actual overflows, thus bringing the behaviour of overflow detection for concatenation into line with that for arithmetic operations.

External Functions (UDFs)

Some small but welcome changes for external function usage come with Firebird 2.0. For example, previously the diagnostics regarding a missing or unusable function module have made it hard to tell whether a module was missing or access to it was being denied due to the ***UDFAccess*** setting in *firebird.conf*. Now, the engine returns separate, understandable messages for each case.

Changes have been made to some legacy UDFs. See Appendix I, *Function Summary* 242.

NULL Signalling

When processing an external function call, the engine sends (usually) a pointer reference to whatever data it has for a parameter, regardless of the data type declared for it. The function itself is left to decide whether it can convert the parameter to the type it expects. A problem with UDFs in the past has been the lack of a generic way to signal to the external function code that a null was being sent in a parameter.

In programming a UDF, the author could only assume that the function might receive a null and try to work around it by assuming an empty string if it got a null reference for a string, zero for a null numeric, and so on. The UDFs that are distributed with Firebird previously have worked on those assumptions or, in the case of some of the functions in the FbUDF library, have taken the more complex route of supplying raw descriptors. Besides, there is always the risk of crashing any number of existing public and private UDFs that do not expect NULL and, in some language environments, do not recognise it.

Ideally, the typical declaration needed to be kept simple—no descriptors—while at the same time being able to signal null to a UDF that is known to be able to handle null.

The solution that has been implemented is an enhanced syntax that allows the keyword NULL to be appended to the UDF parameter type if the UDF is known to be capable of processing a pointer to data that consists of SQL NULL. No other change is required.

Example

```
declare external function sample
int null
returns int by value...;
```

Changed ib_udf Functions

The string functions ASCII_CHAR, LOWER, "LOWER", LPAD, LTRIM, RPAD, RTRIM, SUBSTR and SUBSTRLEN in the Firebird 2 version of the library **`ib_udf`** have been enhanced to take a NULL signal on their inputs and have it interpreted correctly.

The code in those functions has been modified to recognize null only when NULL is signaled by the engine. From Firebird 2.0 onward, those functions no longer assume that an empty string means a NULL string.

The functions won't crash if you don't upgrade: they will simply be unable to detect NULL.

Using NULL Signalling with the Changed Functions

If you are already using functions from **`ib_udf`** and want to take advantage of null signaling (and null recognition) in the adapted functions, a script named **`ib_udf_upgrade.sql`** comes with Firebird 2.x in the **`../misc/upgrade/`** directory that applies this enhancement to your existing declarations for these UDFs in pre-v.2 databases.

This script should be used only when all of the following are true:

- ➢ you are running a Firebird 2.x server
- ➢ you are using the new **`ib_udf`** library distributed with Firebird 2.x
- ➢ operation requests in applications and PSQL modules are modified to pass Null in the function's argument as Null, rather than some workaround like an empty string.

It is recommended to do this upgrade when no other users are connected to the database. Remember to commit your work after running the script.

If you have never used any **`ib_udf`** functions in your database and want to do so, you should connect to the database and run the script **`$firebird/UDF/ib_udf2.sql`**, preferably when no other users are connected, and commit afterwards.

❖ *Note the "2" at the end of the script name—ib_udf***2***.sql not ib_udf.sql.*

Modify a UDF Declaration

`ALTER EXTERNAL FUNCTION` has been implemented, to enable the **`entry_point`** or the **`module_name`** to be changed when the UDF declaration cannot be dropped due to existing dependencies.

Bug Fixes

Some persistent old bugs involving concatenation, numeric fields and character set have been fixed.

Several of these are highlighted in Chapter 11, *Character Types* 53.

Chapter 21 Errata

Page	Erratum
420	Under minor heading "Arguments", it reads: "WEEKDAY extracts the day of the week (having Sunday = 1, Monday = 2, and so on)..." The phrase should read: "WEEKDAY extracts the day of the week (having Sunday = **0**, Monday = **1**, and so on)..."

Chapter 22
Querying
Multiple Tables

The enhancements discussed in this chapter concern improvements in the SQL language implementation of joins, unions and subqueries. It should be studied in conjunction with Chapter 18, Indexes, since the behaviour and performance of multi-table queries are closely interleaved with the indexing and optimizer improvements.

Enhancements to UNION Handling

The rules for UNION queries have been improved in several ways.

UNION DISTINCT Keyword

Formerly, Firebird did not support the explicit inclusion of the optional keyword DISTINCT when specifying a simple UNION. Firebird 2.x now accords with the SQL-99 specification and allows UNION DISTINCT as a synonym for simple UNION.

Since DISTINCT is the default mode, according to the standard, the change is really quite a minor one.

Syntax Pattern

```
UNION [{DISTINCT | ALL}]
```

Improved Type Coercion in UNIONs

Automatic type coercion logic between subsets of a union is now more intelligent. Resolution of the data type of the result of an aggregation over values of compatible data types, such as case expressions and columns at the same position in a union query expression, now uses smarter rules.

Syntax Rules

Let DTS be the set of data types over which we must determine the final result data type.

1. All of the data types in DTS shall be comparable.

2. Case:

   a. If any of the data types in DTS is character string, then:

      i. If any of the data types in DTS is variable-length character string, then the result data type is variable-length character string with maximum length in characters equal to the largest maximum amongst the data types in DTS.

ii. Otherwise, the result data type is fixed-length character string with length in characters equal to the maximum of the lengths in characters of the data types in DTS.

iii. The character set/collation is used from the first character string data type in DTS.

b. If all of the data types in DTS are exact numeric, then the result data type is exact numeric with scale equal to the maximum of the scales of the data types in DTS and the maximum precision of all data types in DTS.

Checking for precision overflows is done at run-time only. The developer should take measures to avoid the aggregation resolving to a precision overflow.

---

c. If any data type in DTS is approximate numeric, then each data type in DTS shall be numeric else an error is thrown.

d. If some data type in DTS is a date/time data type, then every data type in DTS shall be a date/time data type having the same date/time type.

e. If any data type in DTS is BLOB, then each data type in DTS shall be BLOB and all with the same sub-type.

An ANY/ALL/IN Subquery Can Now be a UNION Set

The subquery element of an ANY, ALL or IN search may now be a UNION query.

Subqueries and INSERT Statements

SELECT specifications used in subqueries and in INSERT INTO <insert-specification> SELECT.. statements can now specify a UNION set.

New JOIN Types

Firebird 2 gave us the CROSS JOIN, while v.2.1 brought two variants of NATURAL JOIN 135.

CROSS JOIN

CROSS JOIN is now supported. Logically, this syntax pattern:

```
A CROSS JOIN B
```

is equivalent to either of the following:

```
A INNER JOIN B ON 1 = 1
```

or, simply:

```
FROM A, B
```

Named Columns Join and NATURAL JOIN

Two more JOIN types are introduced: the NAMED COLUMNS join and its close relative, the NATURAL join.

Syntax and Rules

```
<named columns join> ::=
  <table reference> <join type> JOIN <table reference>
    USING ( <column list> )
<natural join> ::=
 <table reference> NATURAL <join type> JOIN <table primary>
```

Named columns join

1. All columns specified in <column list> should exist in the tables at both sides.
2. An equi-join (<left table>.<column> = <right table>.<column>) is automatically created for all columns (ANDed).
3. The USING columns can be accessed without qualifiers—in this case, the result is equivalent to COALESCE(<left table>.<column>, <right table>.<column>).
4. In “SELECT *”, USING columns are expanded once, using the above rule.

Natural join

1. A “named columns join” is automatically created with all columns common to the left and right tables.
2. If there is no common column, a CROSS JOIN is created.

Examples

```
select * from employee
  join department
  using (dept_no);

select * from employee_project
  natural join employee
  natural join project;
```

Avoiding Pointless Equi-joins

In the rare case where a cross join of three or more tables involved table[s] that contained no records, performance could be extremely slow. From V.2.1.2 onward, the optimizer short-circuits an equi-join that previously would have caused walks through populated tables looking for matches in empty tables.

RDB$DB_KEY Returns NULL in Outer Joins

By some anomaly, the physical RDB$DB_KEY has always returned a value on every output row when specified in an outer join, thereby making a test predicated on the assumption that a non-match returns NULL in all fields return False when it ought to return True. Now, RDB$DB_KEY returns NULL when it should do so.

User-specified Query Plans

Several improvements have been made for handling of user-specified query plans.

- Short-circuit optimization for user-supplied plans has been implemented
- A user-specified access path can be supplied for any SELECT-based statement or clause
- Complex outer joins can now be optimized manually to good effect because the optimizer now propagates the relevant plan fragments down to nested levels
- A user-supplied plan will be checked for correctness in outer joins

Syntax rules

The following schema describing the syntax rules should be helpful when composing plans.

```
PLAN ( { <stream_retrieval> | <sorted_streams> | <joined_streams> } )

<stream_retrieval> ::= { <natural_scan> | <indexed_retrieval> |
<navigational_scan> }

<natural_scan> ::= <stream_alias> NATURAL

<indexed_retrieval> ::= <stream_alias> INDEX ( <index_name>
[, <index_name> ...] )

<navigational_scan> ::= <stream_alias> ORDER <index_name>
[ INDEX ( <index_name> [, <index_name> ...] ) ]

<sorted_streams> ::= SORT ( <stream_retrieval> )

<joined_streams> ::= JOIN ( <stream_retrieval>, <stream_retrieval>
[, <stream_retrieval> ...] )
| [SORT] MERGE ( <sorted_streams>, <sorted_streams> )
```

About the Retrieval Methods

Natural scan means that all rows are fetched in their natural storage order. Thus, all pages must be read before search criteria are validated.

Indexed retrieval uses an index range scan to find row ids that match the given search criteria. The found matches are combined in a sparse bitmap which is sorted by page numbers, so every data page will be read only once. After that the table pages are read and required rows are fetched from them.

Navigational scan uses an index to return rows in the given order, if such an operation is appropriate, viz.

- The index b-tree is walked from the leftmost node to the rightmost one.
- If any search criterion is used on a column specified in an ORDER BY clause, the navigation is limited to some subtree path, depending on a predicate.
- If any search criterion is used on other columns which are indexed, then a range index scan is performed in advance and every fetched key has its row id validated against the resulting bitmap.

- Then a data page is read and the required row is fetched.

A navigational scan incurs random page I/O, as reads are not optimized

Sort Operations

A sort operation performs an external sort of the given stream retrieval.

Join Operations

A join can be performed either via the *nested loops* algorithm (JOIN plan) or via the *sort merge* algorithm (MERGE plan).

- An *inner nested loop join* may contain as many streams as are required to be joined. All of them are equivalent.
- An *outer nested loops join* always operates with two streams so, in cases where three or more outer streams are joined, you will see nested JOIN clauses in the plan.

Sort Merge Operations

A sort merge operates with two input streams which are sorted beforehand, then merged in a single run.

Examples

```
SELECT RDB$RELATION_NAME
FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME LIKE 'RDB$%'
PLAN (RDB$RELATIONS NATURAL)
ORDER BY RDB$RELATION_NAME
```

```
SELECT R.RDB$RELATION_NAME, RF.RDB$FIELD_NAME
FROM RDB$RELATIONS R
JOIN RDB$RELATION_FIELDS RF
ON R.RDB$RELATION_NAME = RF.RDB$RELATION_NAME
PLAN MERGE (SORT (R NATURAL), SORT (RF NATURAL))
```

- A PLAN clause may be used in all SELECT expressions, including subqueries, derived tables and view definitions. It can be also used in UPDATE and DELETE statements, since they are implicitly based on select expressions.
- If a PLAN clause contains some invalid retrieval specification, an error will be returned if its presence makes the query impossible; otherwise the bad clause will be silently ignored.
- From Firebird 2.0 onward, an ORDER <*navigational_index*> INDEX (<*filter_indices*>) style of plan is understood by the engine and can be used.

Optimizer Improvements

Better cost-based calculation has been included in the optimizer routines as part of a major collection of changes done in Firebird 2.0 to improve many aspects of performance.

Optimizations Affecting All Databases

The following changes affect all databases, including previous ODS versions.

Faster Processing of Dirty Pages

Firebird 2.0 offers a more efficient processing of the list of modified pages—the "dirty pages" tree. It affects all kinds of batch data modifications performed in a single transaction and eliminates the known issues with performance getting slower when using a buffer cache of more than 10,000 pages. The overall performance of data modifications benefits from this change, too.

New Maximum Cache Size

The maximum page cache size has been increased to 128,000 pages, making possible a 2Gb cache for a database with a 16Kb page size.

Faster Evaluation of OR and IN()

Multiple OR and IN(set of constants) now return their Boolean results faster, thanks to an optimization of sparse bitmap operations.

Improved UNIQUE Retrieval

The refurbished optimizer uses a more realistic cost value for unique retrieval.

Optimization of NOT Conditions Improved

NOT conditions are simplified and, where possible, are restated to get the benefit of using an index.

Example

```
(NOT NOT A = 0) -> (A = 0)

(NOT A > 0) -> (A <= 0)
```

As a rule of thumb, anything that spares the optimizer some work will make performance better. Avoid predicates that require NOT or <> if you have the opportunity to restate them positively yourself!

HAVING Conjunctions Distributed to the WHERE Clause

If a HAVING clause, or any outer-level SELECT, refers to a field being grouped by, this conjunct is distributed deeper in the execution path than the grouping, thus allowing an index scan to be used. In other words, it allows the HAVING clause not only be treated as the WHERE clause in this case, but also be optimized the same way.

Examples

```
select rdb$relation_id, count(*)
from rdb$relations
group by rdb$relation_id
having rdb$relation_id > 10

--
select * from (
select rdb$relation_id, count(*)
from rdb$relations
group by rdb$relation_id
) as grp (id, cnt)
where grp.id > 10
```

In both cases, an index scan is performed instead of a full scan.

UNION Conjunctions

The optimizer will distribute UNION conjunctions to the inner streams when possible.

Improved Handling of CROSS JOIN and Merge/SORT

Improved cross join and merge/sort handling.

Better Choice of Join Order for Mixed Inner/Outer Joins

New optimizer logic allows more options for choosing a reasonable join order for intermixed inner and outer joins.

Equality Comparison on Expressions

A MERGE PLAN may now be generated for joins that use equality comparison on expressions.

For ODS 11 Databases only

This group of optimizations affects databases that were created under Firebird 2.

Segment-level Selectivities are Used

The implementation of *segment-level index selectivity* is discussed in detail in Chapter 18 86.

Better Support for IS NULL and STARTING WITH

Previously, the predicates IS NULL and STARTING WITH were optimized separately from others, thus causing non-optimal plans in complex Boolean expressions involving ANDs and ORs. On ODS11 databases, these predicates are optimized in a regular way and hence benefit from all possible optimization strategies.

Matching of Both OR and AND Nodes to Indexes

Complex Boolean expressions consisting of many AND/OR predicates are now entirely mapped to available indices if at all possible. Previously, such complex expressions could be optimized badly.

Better JOIN Orders

Cost estimations have been enhanced to improve JOIN orders.

Indexed Order Enabled for Outer Joins

It is now possible for indexed order—“navigational walk”— to be utilised for outer joins.

Chapter 22 Errata

Page	Erratum
466	Near top of page, the code sample that reads:

```
SELECT ...
WHERE PARENT_COUNTRY = 'AU' OR 1=1
```

should be

```
SELECT ...
WHERE PARENT_COUNTRY = 'AU' OR 1=0
```

Chapter 23 Ordered and Aggregated Sets

Ordered sets are those that are specified to be returned in a particular row sequence that is governed by an ORDER BY clause following the optional search specification (WHERE clause).

Aggregated sets are those that are specified to be returned in aggregated groups according to specifications in a GROUP BY clause. The type and levelling of aggregation is typically governed by an aggregating function, such as SUM(), AVG(), COUNT(), MAX(), etc., and by the sequence of the output columns in the GROUP BY clause. Since v.1.5, it has been illegal to include in the SELECT list of an aggregated statement any output columns that are not reflected in the GROUP BY clause.

Efforts to enhance and sanitise the syntax support for ordered and aggregate sets have continued. That effort has necessarily been interleaved with improvements implemented in optimization.

Improvements in Sorting

Some useful improvements have been made to SQL sorting operations.

Ordering or Grouping by Alias Name

Column aliases are now allowed in both **ORDER BY** and **GROUP BY** clauses.

Examples

```
SELECT RDB$RELATION_ID AS ID
FROM RDB$RELATIONS
ORDER BY ID
```

```
SELECT RDB$RELATION_NAME AS ID, COUNT(*)
FROM RDB$RELATION_FIELDS
GROUP BY ID
```

GROUP BY Arbitrary Expressions

A **GROUP BY** condition can now be any valid expression.

Example

```
...
GROUP BY
SUBSTRING(CAST((A * B) / 2 AS VARCHAR(15)) FROM 1 FOR 2)
```

*Order SELECT * Sets by Degree Number*

Order by degree (ordinal column position) now works on a **SELECT *** list.

Example

```
SELECT * FROM RDB$RELATIONS
ORDER BY 9
```

This syntax causes the column list to be expanded and taken into account when determining which column the number refers to. The effect is that, now, a query like

```
SELECT T1.*, T2.COL FROM T1, T2
ORDER BY 2
```

will sort on the second column of table T1, while previous versions would sort on T2.COL.

According to grammar rules in effect since v.1.5, **ORDER BY *<value_expression>*** is allowed and **<value_expression>** could be a variable or a parameter. It is tempting to assume that **ORDER BY <degree_number>** could thus be validly represented as a replaceable input parameter, or an expression containing a parameter.

However, while the DSQL parser does not reject the parameterised **ORDER BY** clause expression if it resolves to an integer, the optimizer requires an absolute, constant value in order to identify the position in the output list of the ordering column or derived field. If a parameter is accepted by the parser, the output will undergo a “dummy sort” and the returned set will be unsorted.

NULLS Now “Lowest” for Sorts

NULL is now treated as the lowest possible value for ordering purposes and sets ordered on nullable criteria are sorted accordingly. Thus:

- for ascending sorts NULLs are placed at the beginning of the result set
- for descending sorts NULLs are placed at the end of the result set

In former versions, NULLs were always at the end. If you have client code or PSQL definitions that rely on the legacy NULLs placement, it will be necessary to use the NULLS LAST option in your ORDER BY clauses for ascending sorts.

It should be noted also that use of NULLS LAST or NULLS FIRST to “break” the standard sort order for NULLs will prevent the use of an index for the sort operation.

A bug, whereby an ORDER BY on a big column with a COLLATE clause would terminate the server, was fixed.

Chapter 24
Views & Other Virtual Tables

Views have long been a useful type of SQL object for developers wanting to package complex or frequently accessed sets into pre-compiled queries that behave like tables. Views have acquired a largely unjustified bad reputation in the past due to poor implementation of some aspects. For Firebird 2 and 2.1, a considerable amount of effort has gone into improving and stabilising the usage of views .

V.2.1 also brings a completely new kind of virtual table into the developer's armoury: the global temporary table, abbreviated by all to GTT. Like views, they are precompiled objects, but there the similarity ends. The purpose of GTTs is to provide precompiled containers into which rows can be inserted in run-time and manipulated as though they were persistent data. GTTs are discussed later in this chapter 147.

Views

If you are migrating existing applications or PSQL modules that accessed tables through views, pay attention to the compatibility notes.

Extensions to View Syntax and Usage

Besides the improvements for updatable views, the allowable syntaxes for defining them have been extended by bringing the rules for view definitions into line with the full syntax available for any SELECT statement.

This process was made complete in v.2.1 by enabling the use of named cursors on views in PSQL modules. For more information about that feature, refer to Chapter 29, Developing PSQL Modules, in the topic *Named Cursors* 162.

Reworking of Updatable Views

A reworking has been done to resolve problems with views that are implicitly updatable, but still have update triggers. This is an important change that will affect systems written to take advantage of the undocumented [mis]behaviour in previous versions.

Views made updatable via triggers no longer perform direct table operations In former versions, a naturally updatable view with triggers passed the DML operation to the underlying table and executed the triggers as well. The result was that, if you followed the official documentation and used triggers to perform a table update (inserted to, updated or deleted from the underlying table), the operation was done twice: once executing the view's trigger code and again executing the table's trigger code. This situation caused performance problems or exceptions, particularly if blobs were involved.

Now, if you define triggers for a naturally updatable view, it becomes effectively like a non-updatable view that has triggers to make it updatable, in that a DML request has to be defined on the view to make the operation on the underlying table happen, viz.,

1. if the view's triggers define a DML operation on the underlying table, the operation in question is executed once and the table triggers will operate on the outcome of the view's triggers
2. if the view's triggers do not define any DML request on the underlying table then no DML operation will take place in that table

Some existing code may depend on the assumption that requesting a DML operation on an updatable view with triggers defined would cause the said operation to occur automatically, as it does for an updatable view with no triggers. For example, this "feature" might have been used as a quick way to write records to a log table en route to the "real" update. Now, it will be necessary to adjust your view trigger code in order to make the update happen at all.

Changed Logic for View Updates

A NOT NULL constraint that was applied to the underlying base column of a table during definition of the table passes to the view. However, if the table column inherited the NOT NULL constraint from a domain, it will not be passed to the column in the view.

Extensions to CREATE VIEW

From Firebird 2.0 onward, views are treated as fully-featured SELECT expressions. Consequently, the clauses FIRST/SKIP, ROWS, UNION, ORDER BY and PLAN are now allowed in views and will work as expected.

Column aliases can now be processed as column names in the **CREATE VIEW** definition. The main benefit of this improvement is that, if you want to output columns from a single-table view definition with names other than those of the columns of the underlying table, it is no longer a requirement that you supply a full list of column names as an argument to the view name in your CREATE statement.

For example, previously, if you wanted to output TAB.COL1 with the new identifier CODE and TAB.COL2 with the new identifier NAME, you would need a declaration like this:

```
CREATE VIEW V_TEST (CODE, NAME) AS
  SELECT ID,
         COL1,
         COL2
  FROM TAB;
```

From v.2.1, with ODS 11.1 and higher databases, the following is accepted:

```
CREATE VIEW V_TEST AS
  SELECT ID,
         COL1 AS CODE,
         COL2 AS NAME
  FROM TAB;
```

Global Temporary Tables

Global temporary tables (GTTs) are tables that are stored in the system catalogue with permanent metadata, but whose data is created for non-persistent storage at run-time. The "lifetime" of the stored data is aligned with the scope of either the transaction or the connection. The metadata of the GTT are shared among all connections and transactions but data from different connections (or transactions, depending on the scope) are isolated from each other.

Scope of GTT Data

GTTs come in two flavours

- with data that persists for the lifetime of connection in which the specified GTT was referenced. In the system tables, it is identified with a relation type of 4.
- with data that persists only for the lifetime of the referencing transaction. In the system tables, it is identified with a relation type of 5.

Instantiation

An instance of a GTT—a set of data rows created by and visible within the given connection or transaction—is created when the GTT is referenced for the first time, usually at statement prepare time. Each instance has its own private set of pages on which data and indexes are stored. The data rows and indexes have the same physical storage layout as permanent tables.

When the connection or transaction ends, all pages of a GTT instance are released immediately. It is similar to what happens when a DROP TABLE is performed, except that the metadata definition is retained, of course. This is much quicker than the traditional row-by-row delete + garbage collection of deleted record versions.

Temporary Storage

The data and index pages of all GTT instances are placed in separate temporary files. Each connection has its own temporary file created the first time the connection references some GTT. These temporary files are always opened with Forced Writes = OFF, regardless of the database setting for Forced Writes.

Syntax Pattern for Creating GTTs

Creating a definition for a temporary table is very similar to creating a regular table::

```
CREATE GLOBAL TEMPORARY TABLE (
  <column-definitions>
  [, <constraint-definitions> ] )
  [ON COMMIT <DELETE | PRESERVE> ROWS]
```

CREATE GLOBAL TEMPORARY TABLE is a regular DDL object creation statement that is processed by the engine the same way as a CREATE TABLE statement is processed. Accordingly, it not possible to create or drop a GTT within a stored procedure or trigger.

The ON COMMIT Clause

The ON COMMIT clause sets the scope of the data persistence for your GTT:

- ON COMMIT DELETE ROWS creates a GTT whose rows are cleared completely from the database immediately the transaction ends. This is the default type of GTT created if the ON COMMIT clause is omitted.
- ON COMMIT PRESERVE ROWS creates a GTT whose data will persist in the database from when the user first accesses it until the connection ends. The user can insert and delete rows in one or many transactions.

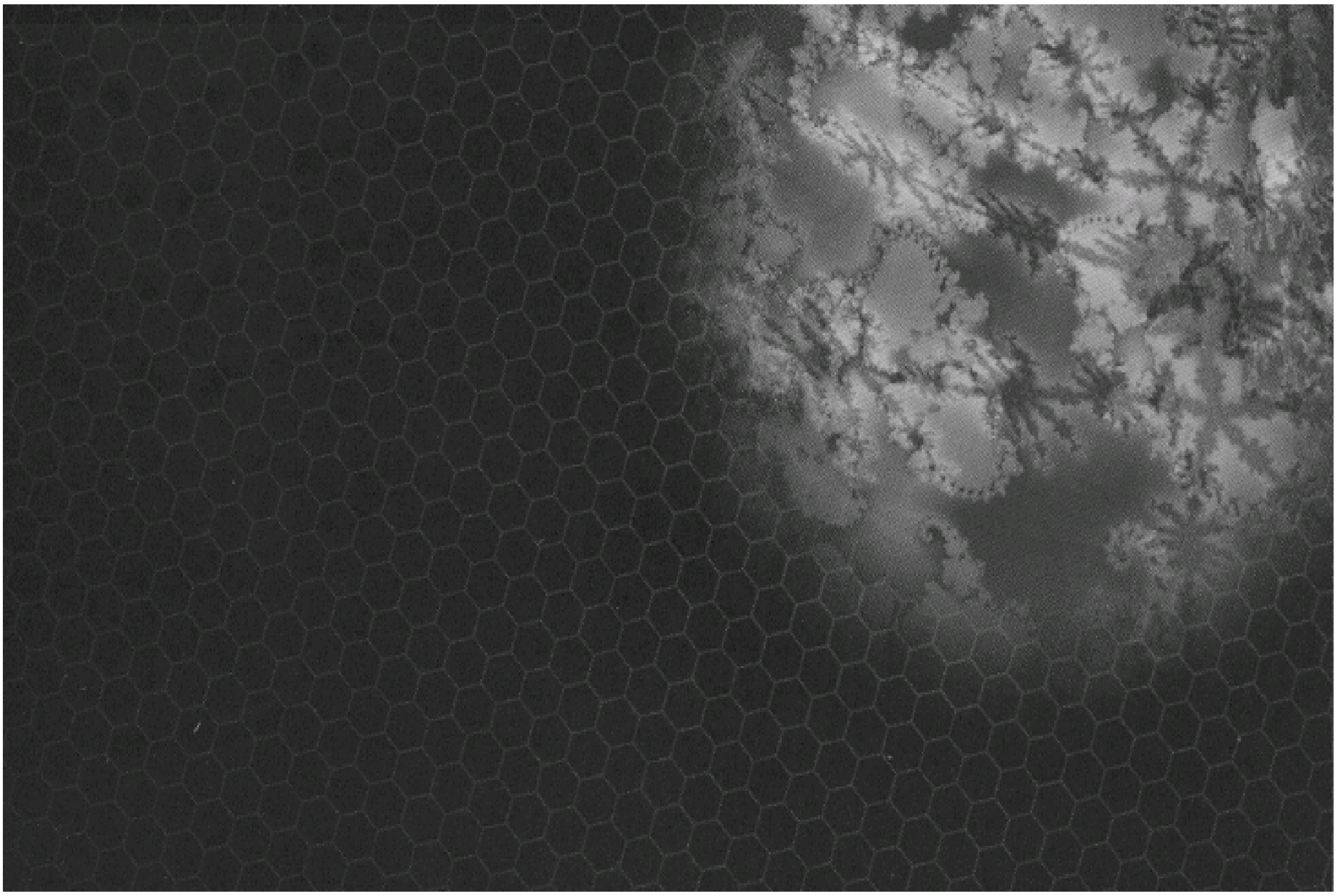
The clearing of records at end-of-life does not cause DELETE triggers to fire.

There is no limit to how many GTT instances can coexist. If you have *n* transactions active simultaneously and each transaction has referenced the same ON COMMIT DELETE ROWS GTT then you will have *n* instances of that GTT.

GTT Object Restrictions

The same structural features that apply to regular tables (domains, indexes, triggers, field-level and table level constraints) are also available to a GTT, with certain restrictions on how GTTs and regular tables can interrelate:

- FOREIGN KEY references between persistent and temporary tables are forbidden
- A GTT with ON COMMIT PRESERVE ROWS cannot have a reference on a GTT with ON COMMIT DELETE ROWS
- No column constraint for a regular table nor domain constraint can refer to a GTT



Part Six

Transactions

Topics in This Part

Chapter 25
Overview of
Firebird Transactions

No supplementary information.

Chapter 26
Configuring Transactions

Although little has changed regarding transactions in Firebird 2, the much-requested capability to configure a lock time-out period for “wait and hope” transactions has been implemented at last.

Additional TPB constants

Three new constants were added to the transaction parameter buffer (TPB), the optional vector that applications used to configure transaction attributes when starting transactions. The three new constants are:

- LOCK TIMEOUT for specifying a timeout period for WAIT transactions
- NO AUTO UNDO to prevent a transaction from maintaining an undo log
- IGNORE LIMBO ignores record versions that were created by limbo transactions. It is for internal use mainly, used by the *gfix* utility.

Configuring Lock Timeout

All Firebird versions provide two transaction lock resolution modes: NO WAIT and WAIT.

NO WAIT mode is the default lock resolution strategy. It means that an immediate exception occurs if the transaction cannot acquire a lock on a record it is requesting to update. For the transaction to proceed or end as the result of the exception, the application must handle the lock condition and either try again or roll the transaction back.

WAIT effects a blocking pause which ends only when the conflicting concurrent transaction (call it Transaction B) ends, by being committed or rolled back, thus allowing the requesting transaction (Transaction A) to acquire the lock. This is the “wait and hope” configuration. It would be used under conditions where Transaction B is known not to be performing any operations that would affect the state of the set of records commanded by Transaction A's snapshot. The usefulness of WAIT lock resolution is limited otherwise, since Transaction A would be likely to “lose the race”anyway, if Transaction B succeeds in committing writes.

The new feature extends the WAIT mode by allowing a finite time interval to wait for Transaction B to end. If the timeout has passed, an error (*isc_lock_timeout*) is reported and, for Transaction A, the client is back in the same situation as though NO WAIT had been configured.

Timeout intervals are specified per transaction.

TPB Constant isc_tpb_lock_timeout

At the API level, the feature uses a new transaction parameter buffer (TPB) constant, *isc_tpb_lock_timeout*. In practice, many of the common data access layers for applications wrap the API transaction settings and may explicitly make DSQL access a no-op.

DSQL Setting

A language interface to transaction settings is available through a DSQL statement, **SET TRANSACTION**. The syntax of the statement has been extended to include the optional clause **LOCK TIMEOUT <value>**, where the <value> argument is the time in seconds that the transaction waits to acquire a lock on a record before giving up and reporting an error.

Syntax

```
SET TRANSACTION LOCK TIMEOUT n;
```

Example

To set a lock timeout of 10 seconds in an *isql* session:

```
SQL> SET TRANSACTION LOCK TIMEOUT 10;
```

The command is used in DSQL to start a transaction without using the specialized API call to create a new transaction. To the already existing options, the following have been added. Notice this is not new functionality: it's available through the TPB since years ago (they appear in ibase.h as items for TPBs). This extension only makes those options available to clients that want to start a transaction by executing a DSQL command, like isql's command prompt.

Lock timeout has to be zero or positive and is not valid if NO WAIT is specified for lock resolution.

Disabling the Undo Log

By default, an undo log is maintained in memory for each statement for which a transaction succeeds in writing to disk. It acts as history to which the engine will refer, to find and remove (or revert) these records if the fails for some reason. The undo log does not affect transaction consistency: it functions as an aid to cleanup in these failure situations, thus avoiding the need for those unused records to be garbage-collected. After a commit or rollback completes, the undo log is erased.

The **NO AUTO UNDO** attribute prevents the transaction from keeping an undo log. It is useful for very large batch inserts where there is no likelihood of failure since, besides being a source of resource overhead, undo logging has the potential to be abandoned by the engine if the log structure gets too large.

Without the undo logs, subsequent transactions reading the unused records will collect the garbage. The NO AUTO UNDO attribute is specified per transaction.

TPB Constant isc_no_auto_undo

At the API level, the feature uses a new transaction parameter buffer (TPB) constant, *isc_tpb_no_auto_undo.* In practice, many of the common data access layers for applications wrap the API transaction settings and may explicitly make DSQL access a no-op.

DSQL Setting

The **SET TRANSACTION** syntax extensions include the optional clause **NO AUTO UNDO** to disable the undo logging.

Syntax

```
SET TRANSACTION NO AUTO UNDO;
```

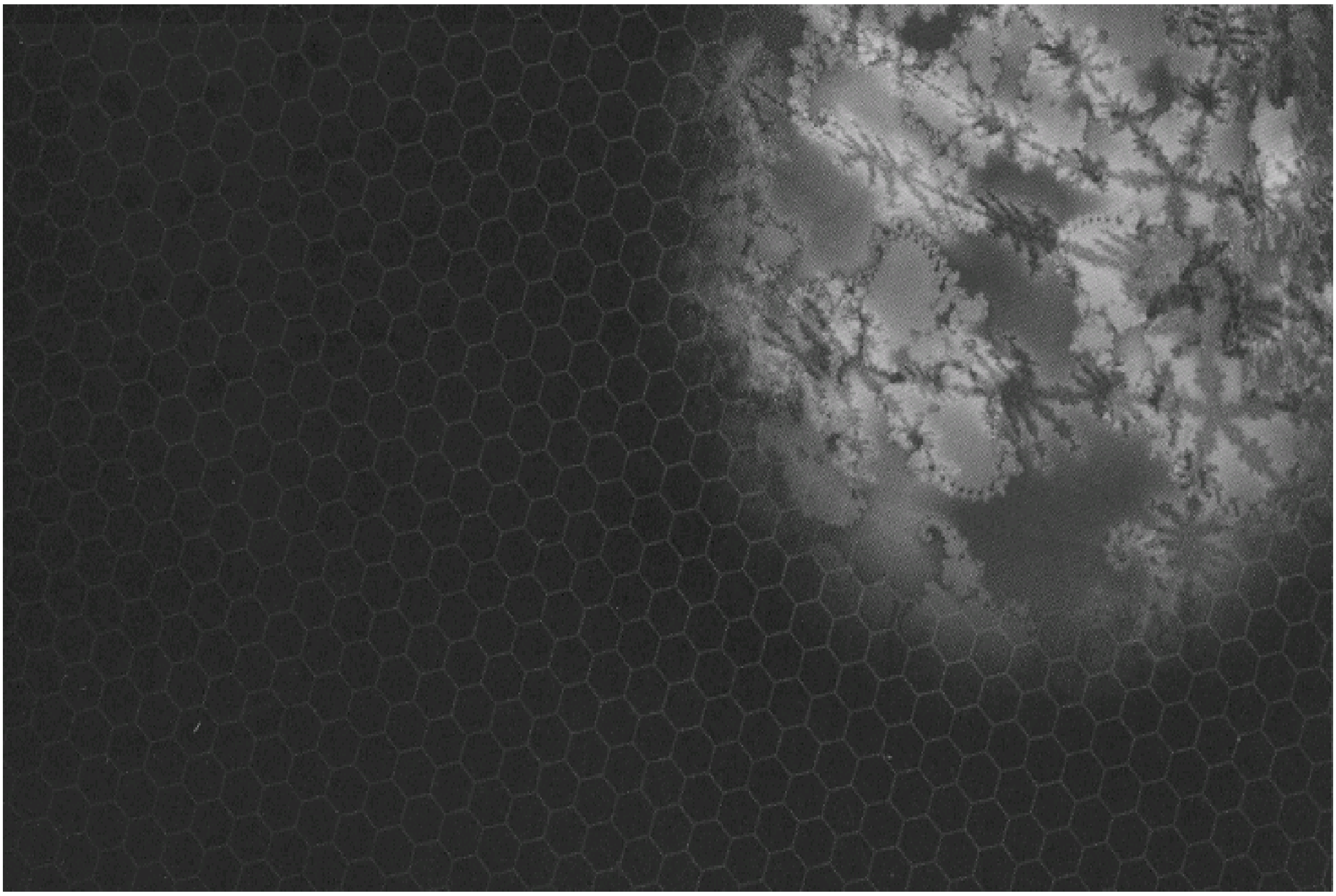
For transactions that do not change any records, **NO AUTO UNDO** has no effect.

Chapter 27 Programming with Transactions

An additional constant has been added to the transaction parameter buffer structure that may be of interest in your application development:

isc_tpb_lock_timeout for specifying a timeout period for WAIT transactions

For details, refer to the previous chapter.



Part Seven

Server Programming

Topics in This Part

Chapter 28 Introduction to Firebird Programming

No supplementary information.

Chapter 29 Developing PSQL Modules

In the initial Firebird 2.0 release, a deliberate restriction was imposed to prevent anyone from dropping, altering or recreating a PSQL module if it had been used since the database was opened. An attempt to prepare the DDL statement would result in an "Object in Use" exception.

Many people complained that the restriction was unacceptable because they depended on performing these metadata changes "on the fly". The restriction was therefore removed at release 2.0.1. However, the reversion in no way implies that performing DDL on active PSQL modules is "safer" in Firebird 2.0.1 than it was in V.1.5.

PSQL Improvements

Procedural SQL (PSQL) is the set of SQL extensions that Firebird provides for developing modules of code for repeatable execution on the server. Several new extensions were added for Firebird 2 and some existing elements have been enhanced.

Variable and Parameter Definitions

Firebird 2.1 makes some new options available when defining variables and parameter arguments for PSQL modules.

Domains for PSQL Variables

It is now possible to use a domain when declaring the data types of arguments and variables in PSQL modules. Depending on your requirements, you can declare the argument or variable using

- the domain identifier alone, in lieu of the native data type identifier, to have the variable inherit all of the attributes of the domain; or
- the data type of the domain, without inheriting CHECK constraints and the DEFAULT value (if declared in the domain), by including the TYPE OF keyword in the declaration.

Syntax for the Arguments

```
data_type ::=
    <builtin_data_type>
    | <domain_name>
    | TYPE OF <domain_name>
```

Examples

```
CREATE DOMAIN DOM AS INTEGER;
CREATE PROCEDURE SP (
  I1 TYPE OF DOM,
  I2 DOM)
RETURNS (
  O1 TYPE OF DOM,
  O2 DOM)
AS
  DECLARE VARIABLE V1 TYPE OF DOM;
  DECLARE VARIABLE V2 DOM;
BEGIN
  ...
END
```

An ALTER DOMAIN operation has the potential to invalidate the precompiled code (BLR) stored for a PSQL module that uses the affected domain. If that happens, there are flags on both trigger and procedure records in the system tables that get set on. You can inspect the state of these flags in the output of SHOW PROCEDURE[S] and SHOW TRIGGER[S] in the *isql* utility.

COLLATE Clause Now Allowed

Collations can now be applied to the definition of PSQL text variables, including stored procedure parameters.

LEAVE <label> Syntax

LEAVE <label> syntax introduced in Firebird 2.0 allows PSQL loops to be marked with labels and terminated in Java style. The purpose is to stop execution of the current block and unwind back to the specified label. Subsequent execution resumes at the statement *following* the terminated loop.

Syntax Pattern

```
<label_name>: <loop_statement>
...
LEAVE [<label_name>]
```

where <loop_statement> is one of

- `WHILE`
- `FOR SELECT`
- `FOR EXECUTE STATEMENT`

Examples

1. Using LEAVE without a label, to terminate reiteration of a FOR...SELECT loop:

```
FOR SELECT
  COALESCE(RDB$SYSTEM_FLAG, 0),
  RDB$RELATION_NAME
FROM RDB$RELATIONS
ORDER BY 1
INTO :RTYPE, :RNAME
DO
BEGIN
  IF (RTYPE = 0) THEN
    SUSPEND;
  ELSE
    LEAVE; -- exits current loop
END
```

2. Setting a label for terminating a WHILE loop once a condition is met:

```
BEGIN
  COUNTER = 100;
  L1:
  WHILE (COUNTER >= 0) DO
  BEGIN
    IF (COUNTER < 50) THEN
      LEAVE L1; -- exits WHILE loop
    CNT = CNT - l;
  END
  -- EXECUTION RESUMES HERE
```

3. Using two labels to set exit behaviours for an outer and an inner loop:

```
STMT1 = 'SELECT RDB$RELATION_NAME FROM RDB$RELATIONS';
L1:
FOR EXECUTE STATEMENT :STMT1 INTO :RNAME
DO
BEGIN
  STMT2 = 'SELECT RDB$FIELD_NAME FROM RDB$RELATION_FIELDS
  WHERE RDB$RELATION_NAME = ';
  L2:
  FOR EXECUTE STATEMENT :STMT2 || :RNAME INTO :FNAME
  DO
  BEGIN
    IF (RNAME = 'RDB$DATABASE') THEN
      LEAVE L1; -- exits the outer loop
    ELSE IF (RNAME = 'RDB$RELATIONS') THEN
      LEAVE L2; -- exits the inner loop
```

```
    ELSE
      SUSPEND;
  END
END
```

LEAVE without an explicit label means interrupting the current (most inner) loop.

ROW_COUNT Enhancement

The context variable **ROW_COUNT** has been enhanced so that it can now return the number of rows returned by a **SELECT** statement. An example of the new usage would be to check whether a singleton **SELECT ... INTO** statement has performed an assignment:

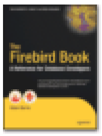
```
..
BEGIN
  SELECT COL FROM TAB INTO :VAR;

  IF (ROW_COUNT = 0) THEN
    EXCEPTION NO_DATA_FOUND;
END
..
```

See also its usage in the examples below for *named cursors*.

Named Cursors

Support for multiple named (a.k.a. explicit) cursors are now supported, both in PSQL modules and in DSQL EXECUTE BLOCK statements.

Unlike the previous cursor implementation (still available) which uses an inline cursor declaration in the body of the module (**AS CURSOR cursorname**), the enhanced implementation requires the cursors to be declared in the *header section*, as variables of other types are.



For details about the structure and elements of a PSQL module, see Chapter 29, *Developing PSQL Modules*. The previous cursor support in PSQL is described in Chapter 30, page 614, *Cursors in PSQL*.

Syntax pattern

```
DECLARE [VARIABLE] <cursor_name> CURSOR FOR ( <select_statement> );
OPEN <cursor_name>;
```

```
FETCH <cursor_name> INTO <var_name> [, <var_name> ...];
CLOSE <cursor_name>;
```

Examples

1. Declare a cursor and use it to read out a list of names to be returned from a selectable stored procedure:

```
DECLARE RNAME CHAR(31);
DECLARE C CURSOR FOR
  ( SELECT RDB$RELATION_NAME
    FROM RDB$RELATIONS );
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
  BEGIN
    FETCH C INTO :RNAME;
    IF (ROW_COUNT = 0) THEN
      LEAVE;
    SUSPEND;
  END
  CLOSE C;
END
```

The **`ROW_COUNT`** system variable can be used after each **`FETCH`** statement to check whether any row was returned.

2. Nest an explicit cursor inside a FOR...SELECT loop whose iterations supply a value for a search parameter that is declared for the nested cursor:

```
DECLARE RNAME CHAR(31);
DECLARE FNAME CHAR(31);
DECLARE C CURSOR FOR
  ( SELECT RDB$FIELD_NAME
    FROM RDB$RELATION_FIELDS
    WHERE RDB$RELATION_NAME = :RNAME
    ORDER BY RDB$FIELD_POSITION );
BEGIN
  FOR
  SELECT RDB$RELATION_NAME FROM RDB$RELATIONS
    INTO :RNAME  DO
    BEGIN
      OPEN C;
      FETCH C INTO :FNAME;
      CLOSE C;
```

```
    SUSPEND;
  END
END
```

Cursor Names

Cursor names are required to be unique among all cursors named in the module. The scope for uniqueness includes cursors declared in the module's body section using the older **`FOR SELECT....AS CURSOR cname`** syntax.

Don't try to intermix the syntaxes of the old-style in-line (**`FOR SELECT...AS CURSOR`**) cursor and the new declared cursor when referring to a particular cursor. For example, an attempt to apply a **`FETCH`** or **`CLOSE`** statement to a cursor that was declared in-line will throw an exception at compile-time.

Although it is not a wonderful idea from the point of view of self-documentation, a cursor can have the same name as another type of variable within the same context.

Positioned Updates

Positioned updates and deletes with cursors using the **`WHERE CURRENT OF`** clause are allowed.

Example

```
DECLARE BATCHID BIGINT;
DECLARE MANUF_DATE TIMESTAMP;
DECLARE QSTAMP BIGINT;
DECLARE C CURSOR FOR
  ( SELECT BATCH_ID, MANUF_DATE FROM NEW_STOCK
    WHERE CAST(MANUF_DATE AS DATE) <= CURRENT_DATE
    AND SERIAL_NO IS NULL
    ORDER BY BATCH_ID, MANUF_DATE );
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
  BEGIN
    FETCH C INTO :BATCHID, ;
    IF (ROW_COUNT = 0) THEN
      LEAVE;
    QSTAMP = NEXT VALUE FOR S_QSTAMP;
    UPDATE NEW_STOCK
    SET SERIAL_NO = '||:BATCHID ||'/':QSTAMP
```

```
    WHERE CURRENT OF C;
  END
  CLOSE C;
END
```

Cursors on Views

The cursor operator WHERE CURRENT OF can now step through a cursor set selected from a view, just as it does in a cursor set output from a SELECT on a table.

Example

```
...
FOR SELECT ...
   FROM MY_VIEW INTO ... AS CURSOR VIEW_CURSOR DO
BEGIN
  ...
  DELETE FROM MY_VIEW
    WHERE CURRENT OF VIEW_CURSOR;
  ...
END
```

Attempts to open a cursor that is already open, or to fetch from or close a cursor that is already closed, will fail.

All cursors which were not explicitly closed will be closed automatically on exit from the current PSQL module or (executable) block.

In Firebird 1.5 and earlier, referring to "current of <cursor>"outside the scope of the cursor loop was accepted by the PSQL parser, allowing the likelihood of run-time errors occurring as a result. Now, it will be rejected in the procedure or trigger definition.

Invoking RDB$SET_CONTEXT as a Void Function

In PSQL, the internal UDF *RDB$SET_CONTEXT* can be called as though it a were void function (or, for Object Pascal programmers, a *procedure*).

Example

```
BEGIN
...
RDB$SET_CONTEXT('USER_TRANSACTION', 'MY_VAR', '123');
...
END
```

For details about usage of RDB$SET_CONTEXT, refer to the topic *Set/Get Functions for Context Variables* 122 in Chapter 21, *Expressions and Predicates*.

Chapter 30
Stored Procedures

Improvements

Firebird 2.0 introduced one PSQL enhancement that is specific to stored procedures.

Defaults for Input Arguments

Default values can now be declared for stored procedure arguments. The syntax is similar to that for defining a default value for a column or domain, except that the PSQL assignment symbol (=) is used instead of the keyword DEFAULT.

Arguments with default values are strictly positional and must occur *last* in the argument list. The caller is required to supply values for all of the arguments with no declared defaults and may omit any where the defaults are to be used. Substitution of default values occurs at run-time.

You must declare any arguments not having defaults assigned before any that are declared with default values. For example, it is illegal to do something like this: supply arg1, arg2, miss arg3, set arg4...

If you define a procedure P2 with defaults, call it from another procedure P1 and omit some final, defaulted arguments, then the default values for P2 will be substituted by the engine at time P2 starts executing. The effect is that, if you change the default values for P2, it is not necessary to recompile P1.

Examples

```
CONNECT ... ;
SET TERM ^;
CREATE PROCEDURE P1 (X INTEGER = 123)
RETURNS (Y INTEGER)
AS
BEGIN
  Y = X;
  SUSPEND;
END ^
COMMIT ^
SET TERM ;^

SELECT * FROM P1;

Y
```

```
============
123

EXECUTE PROCEDURE P1;

Y
============
123

SET TERM ^;
CREATE PROCEDURE P2
RETURNS (Y INTEGER)
AS
BEGIN
  FOR SELECT Y FROM P1 INTO :Y
  DO SUSPEND;
END ^
COMMIT ^
SET TERM ;^

SELECT * FROM P2;

Y
============
123

SET TERM ^;
ALTER PROCEDURE P1 (X INTEGER = CURRENT_TRANSACTION)
RETURNS (Y INTEGER)
AS
BEGIN
  Y = X;
  SUSPEND;
END; ^
COMMIT ^
SET TERM ;^

SELECT * FROM P1;

Y
============
5875

SELECT * FROM P2;

Y
============
123
```

```
COMMIT;

CONNECT ... ;

SELECT * FROM P2;

Y
============
5880
```

The source and BLR for the argument defaults are stored in the system table RDB$FIELDS.

Chapter 31
Triggers

Database Triggers

For the first time, with a Firebird 2.1 or higher server and an ODS 11.1 or higher database, you can define triggers that fire in events beyond the boundaries of statement execution. The term *database trigger* covers PSQL modules that you can define to be executed at *transaction* or *connection* level.

Writing Database Triggers

Writing database triggers is very much like writing statement level triggers. Of course, statement-level context variables such as the NEW and OLD column variables are not available. However, note that syntax pattern 171 for the trigger headers is different, reflecting the different phasing of the database- and transaction-level events.

Trigger Events

Unlike statement-level triggers, database triggers do not split an event into BEFORE and AFTER phases. However, as with statement triggers, you can define multiple triggers for an event and use a POSITION clause to assign an execution order for them.

Five distinct events are available: CONNECT, TRANSACTION START, TRANSACTION COMMIT, TRANSACTION ROLLBACK and DISCONNECT.

The CONNECT Event

CONNECT triggers fire in an initial transaction started after a database connection is established. If an exception occurs that is not handled by the triggers, the initial transaction is rolled back, the attachment is disconnected and the exception is returned to the client application. As with other triggers, an exception handler can also raise a custom exception and pass it to the end of the trigger, with the same effect.

After all CONNECT triggers have executed without unhandled exceptions, the initial transaction is committed and the connection is "live".

The TRANSACTION START Event

Triggers are fired in the newly-created user transaction. Uncaught exceptions are returned to the client and the transaction is rolled back.

The TRANSACTION COMMIT Event

The timing of the TRANSACTION COMMIT event depends on whether the transaction is single-phase or two-phase:

- for a single-phase transaction, immediately the transaction is about to commit.
- for a two-phase transaction, in the *Prepare* phase

Uncaught exceptions roll back the trigger's savepoint and the COMMIT request is aborted. The exception is returned to the client.

The TRANSACTION ROLLBACK Event

TRANSACTION ROLLBACK triggers are fired immediately preceding the roll-back of the transaction. Changes done will be rolled back with the transaction. Exceptions are swallowed.

The DISCONNECT Event

When a *detach* request is received, a transaction is started for the DISCONNECT event. Triggers are fired and, if there are uncaught exceptions, the transaction is is rolled back, the *detach* request is executed and the exceptions are swallowed.

If there are no exceptions, the transaction is committed and the *detach* request is executed.

Restrictions

A database trigger's event type cannot be altered. If you need to make a trigger execute in a different event, you must drop it and create a new one redefining the event type.

Only SYSDBA or the database owner may create, recreate, create or alter, or drop database triggers.

Syntax Pattern for Database Triggers

```
<database-trigger> ::=
  {CREATE | RECREATE | CREATE OR ALTER}
    TRIGGER <name>
    [ACTIVE | INACTIVE]
    ON <event>
    [POSITION <n>]
  AS
    BEGIN
      ...
    END
<event> ::=
      CONNECT
    | TRANSACTION START
    | TRANSACTION COMMIT
    | TRANSACTION ROLLBACK
    | DISCONNECT
```

Utilities Support

New parameters were added to *gbak*, *nbackup* and *isql* to suppress database triggers from running. They are available only to the database owner and SYSDBA. The respective switches are:

```
gbak -nodbtriggers

isql -nodbtriggers

nbackup -T
```

For more information about usage and other changes in these utilities, refer to *Part Nine, Tools* 206.

Improvements in V.2.x

A couple of small PSQL improvements have bee made that are specific to trigger modules.

DDL

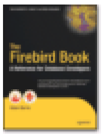
Only one DDL change applies specifically to triggers.

RECREATE TRIGGER

The DDL statement **RECREATE TRIGGER** is now available in DDL. Syntax is the same as for **CREATE TRIGGER**. Semantically, like other RECREATE statements, RECREATE TRIGGER

- creates the trigger if it does not exist already
- drops the trigger and creates it afresh if does exist
- unlike ALTER TRIGGER, will not preserve existing dependencies

The operation will be blocked with an *Object in Use* exception if there are any 'interesting' transactions involving the table.



For **CREATE TRIGGER** syntax, refer to Chapter 31, page 644. Note also the Erratum (below) for this topic.

SQL-2003-Compliant CREATE TRIGGER

Alternative syntax is now available for CREATE TRIGGER that complies with SQL2003. It is optional: Firebird's traditional syntax is still perfectly valid.

Traditional Syntax Pattern

```
create trigger t1
  FOR atable
  [active] before insert or update
as
```

```
  begin
    ...
  end
```

SQL2003 Syntax Pattern

```
create trigger t2
  [active] before insert or update
  ON atable
as
  begin
    ...
  end
```

Differences

- The clause identifying the table precedes the event and action clauses in the traditional syntax whereas it comes after them in the SQL-2003 syntax
- The keyword for the table clause is FOR in the traditional syntax whereas it is ON in the SQL-2003 syntax

Syntaxes using the SQL-2003 pattern are available also for CREATE TRIGGER, RECREATE TRIGGER and CREATE OR ALTER TRIGGER statements.

Restrictions on Illogical NEW/OLD Variable Assignments

Assignments to the OLD context variables are now prohibited for every kind of trigger.

Assignments to NEW context variables in AFTER triggers are also prohibited.

If you get an unexpected error *Cannot update a read-only column*, look for a violation of one of these restrictions as the source of the exception.

Chapter 31 Errata

Page	Erratum
644	Under "Syntax": `[DECLARE VARIABLE variable datatype;...]` should read: `[DECLARE [VARIABLE] variable datatype;...]` since the keyword VARIABLE is optional. Also, it should be noted that, from v.1.5 onward, a starting value for the variable can be included in the declaration: `[DECLARE [VARIABLE] variable datatype = <value>;...]`

Chapter 32 Error Handling and Events

A few improvements have been made in areas of DDL for custom (i.e. user-defined) exceptions and troubleshooting PSQL code modules.

DDL

Longer Custom Exception Messages

The maximum size of the message string for custom exceptions created using **`CREATE EXCEPTION`** has been raised from 78 to 1021 bytes.

Refer to Chapter 32, page 666, *Creating an Exception*.

New Syntaxes for Changing Exceptions

The DDL statements **`RECREATE EXCEPTION`** and **`CREATE OR ALTER EXCEPTION`** have been implemented, in accordance with similar syntaxes already available for creating, recreating or altering objects.

RECREATE EXCEPTION

RECREATE EXCEPTION is exactly like CREATE EXCEPTION if the exception does not already exist. If it does exist, its definition will be completely replaced, if there are no dependencies on it.

CREATE OR ALTER EXCEPTION

CREATE OR ALTER EXCEPTION will create the exception if it does not already exist, or will alter the definition if it does, without affecting dependencies.

Troubleshooting

PSQL Stack Trace

The API client can now extract a simple stack trace Error Status Vector when an exception occurs during PSQL execution (stored procedures or triggers). A stack trace is represented by one string (2048 bytes max.) and consists of all the stored procedure and trigger names, starting from the point where the exception occurred, out to the outermost caller. If the actual trace is longer than 2Kb, it is truncated.

Additional items are appended to the status vector as follows:

isc_stack_trace, *isc_arg_string*, <string length>, <string>

isc_stack_trace is a new error code with value of 335544842L.

Examples

Metadata creation

```
CREATE TABLE ERR (
ID INT NOT NULL PRIMARY KEY,
NAME VARCHAR(16));

CREATE EXCEPTION EX '!';
SET TERM ^;

CREATE OR ALTER PROCEDURE ERR_1 AS
BEGIN
EXCEPTION EX 'ID = 3';
END ^

CREATE OR ALTER TRIGGER ERR_BI FOR ERR
BEFORE INSERT AS
BEGIN
IF (NEW.ID = 2)
THEN EXCEPTION EX 'ID = 2';

IF (NEW.ID = 3)
THEN EXECUTE PROCEDURE ERR_1;

IF (NEW.ID = 4)
THEN NEW.ID = 1 / 0;
END ^

CREATE OR ALTER PROCEDURE ERR_2 AS
BEGIN
INSERT INTO ERR VALUES (3, '333');
END ^
```

1. User exception from a trigger:

```
SQL >INSERT INTO ERR VALUES (2, '2');

Statement failed, SQLCODE = -836
exception 3
-ID = 2
-At trigger 'ERR_BI'
```

2. User exception from a procedure called by a trigger:

```
SQL > INSERT INTO ERR VALUES (3, '3');
```

```
Statement failed, SQLCODE = -836
exception 3
-ID = 3
-At procedure 'ERR_1'
At trigger 'ERR_BI'
```

3. Run-time exception occurring in trigger (division by zero):

```
SQL “INSERT INTO ERR VALUES (4, '4');
```

```
Statement failed, SQLCODE = -802
arithmetic exception, numeric overflow, or string truncation
-At trigger 'ERR_BI'
```

4. User exception from procedure:

```
SQL> EXECUTE PROCEDURE ERR_1;
```

```
Statement failed, SQLCODE = -836
exception 3
-ID = 3
-At procedure 'ERR_1'
```

5. User exception from a procedure with a deeper call stack:

```
SQL> EXECUTE PROCEDURE ERR_2;
```

```
Statement failed, SQLCODE = -836
exception 3
-ID = 3
-At procedure 'ERR_1'
At trigger 'ERR_BI'
At procedure 'ERR_2'
```

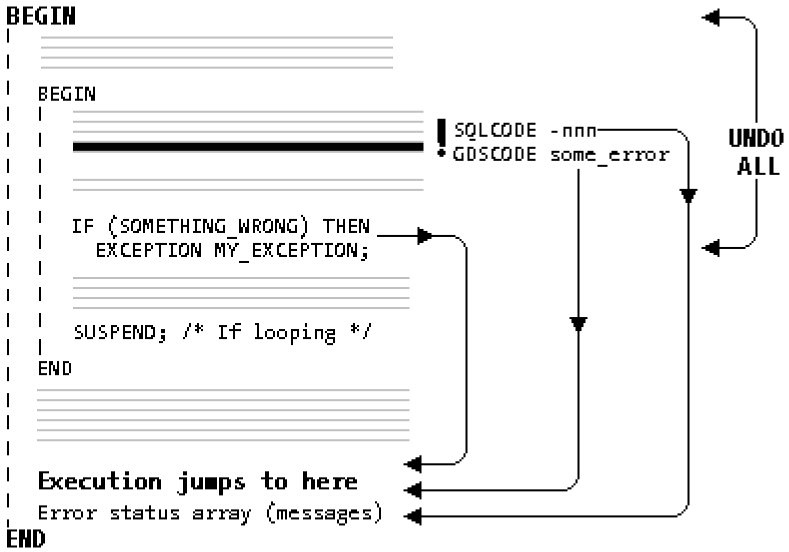
And so on...use your imagination!

Chapter 32 Errata

Page **Erratum**

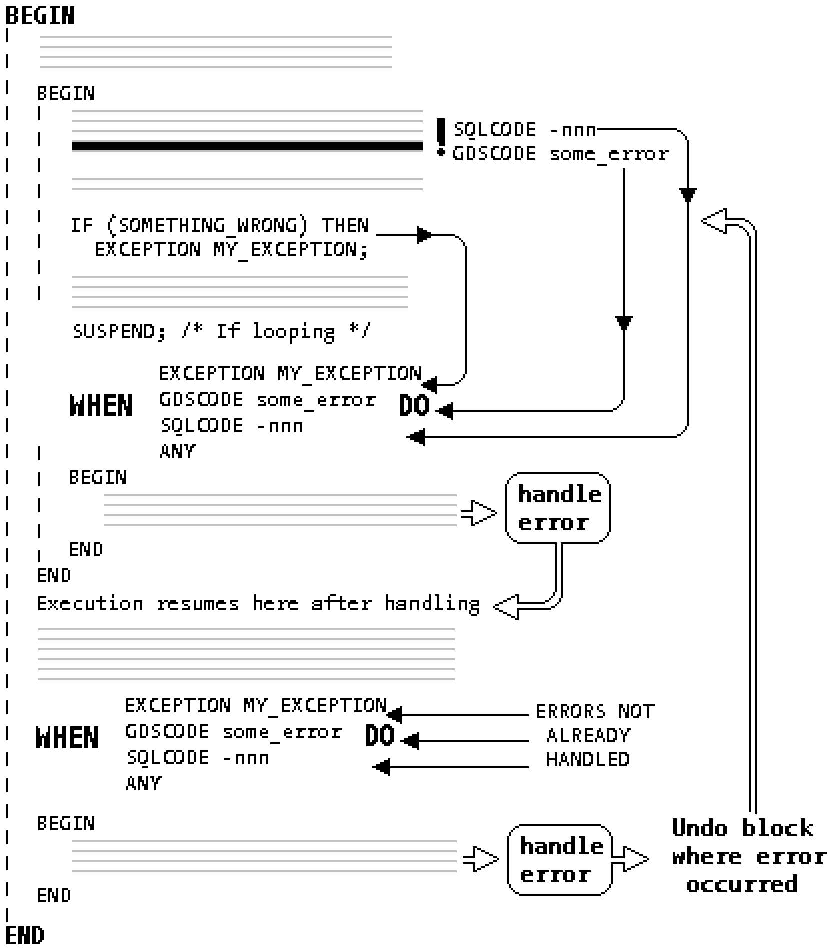
667

This is the wrong graphic, it seems Fig. 32.2. was plugged in there during production, since it is correct in the last proofs eyeballed by the author. Following is the correct 2794f3201.bmp:



668

Figure 32.2 (text in graphic) 'MY_EXCEPTION' should read 'EXCEPTION MY_EXCEPTION'. Following is the corrected graphic:



669

Beneath heading "The WHEN Statement" the passage between "A WHEN statement takes the form" and the paragraph starting with "<compound-statement> is one statement..." should be completely replaced with:

```
WHEN [EXCEPTION] <exception> DO <compound-statement>
```

where the EXCEPTION keyword is required for user-defined exceptions and *<exception>* can be any one of the following:

```
<exception-name> | GDSCODE code | SQLCODE code | ANY
```

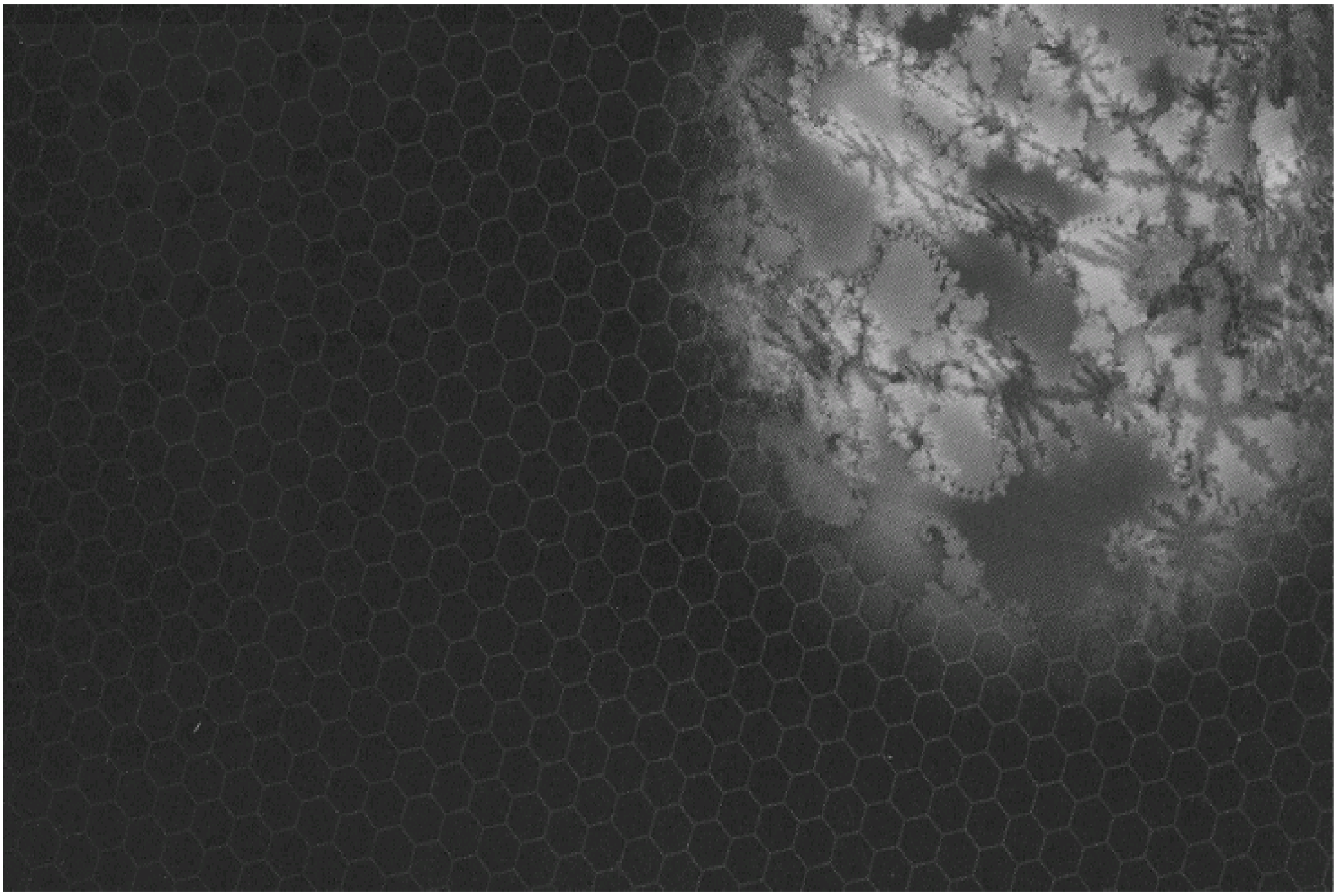
684

The following text under the heading "Using POST_EVENT" is incorrect:

The parameter, event_name, can be....with a numeral. Event names are restricted to 15 characters.

It should read:

The parameter, event_name, can be....with a numeral. Event names are restricted to **64** characters.



Part Eight

Security and Configuration

Topics in This Part

Chapter 33
Security in the Operating Environment

No supplementary information.

Chapter 34 Server Protection

Improving security has had a lot of focus in Firebird 2.0 development. This is the first Firebird release that uses a user authentication database with a different internal structure from the inherited InterBase 6 isc4.gdb, renamed in Firebird 1.5 as security.fdb. Firebird's authentication database is incompatible with those older versions so close attention is needed if you are migrating from one of those.

New Authentication Database

The new authentication database is renamed as **`security2.fdb`**. Inside, the user authentication table, where user names and passwords are stored, is now called RDB$USERS. There is no longer a table named "users" but a new view over RDB$USERS that is named USERS. Through this view, users can change their passwords.

The Firebird 1.5 security database was restructured to use the approach, with minor differences, as described by Ivan Prenosil in Chapter 34, the Special Topic entitled *Customizing User Security*, beginning at page 706.

If you upgrade an existing installation, be sure to upgrade the security database using the provided script in order to keep your existing user logins.

You must make sure that you restore the security database to have a page size of at least 4 Kb. The new security2.fdb will not work with a smaller page size.

For instructions on updating previous security databases, refer to Chapter 1, *Installation* 20.

You will also find a readme file named ***`security_database.txt`*** in the */upgrade* directory beneath the root directory of your installation.

The Security Changes

Security focus was directed at some recognised weaknesses in Firebird's security from malicious attacks:

- the lack of brute-force resistant passwords encryption in the security database
- the ability for any remote user with a valid account to open the security database and read hashes from it (especially interesting in combination with the first point)
- the inability for users to change their own passwords
- the lack of protection against remote brute-forcing of passwords on the server directly

Highlights of Changes

Special measures were taken to make remote connection to the security database completely impossible and to detect and resist hacker-like activity.

No Non-Server Access to security2.fdb

The server will refuse any access to **`security2.fdb`** except through the Services Manager. Don't be surprised if some old program fails on attempting direct access: this is by design. Apart from the enhancement it offers to server security, it also isolates the mechanisms of authentication from the implementation.

Direct connections to the security database are no longer allowed.

User accounts can now be configured only by using the Services API or the *gsec* utility.

The *gsec* utility now uses the Services API.

Access is blocked to the security database while it is in active use by the server. If you need to make some changes to **`security2.fdb`**, just copy it somewhere away from the Firebird root directory and connect to it there as to any normal database.

Non-SYSDBA access to parts of the Services API that return information about users and database paths was disabled in v.2.1. However, a non-privileged user can still retrieve information about itself.

Backing up security2.fdb

For backing up the security database, the Services API is now the only route. When invoking the *gbak* utility for this purpose, you can employ the switch

```
-se[rvice] hostname:service_mgr
```

Native Authentication

Firebird native authentication checks a server-wide security database in order to decide whether a database or server connection request is authorised. The security database stores the user names and passwords of all authorised login identities.

Firebird 1.5 Authentication

In Firebird 1.5 the DES algorithm is used twice to hash the password: first by the client, then by the server, before comparing it with the hash stored in security database. However, this sequence becomes completely broken when the SYSDBA changes a password. The client performs the hash calculation twice and stores the resulting hash directly in the security database. Therefore, hash management is completely client-dependent (or, actually, client-defined).

Firebird 2: Server-side Hashing

To be able to use stronger hashes, another approach was called for. The hash to be stored on the server should always be calculated on the server side. Such a schema already exists in Firebird—in the Services API. This led to the decision to use the Services API for any client activity related to user management. Now, *gsec* and the *isc_user_add(modify, delete)* API functions all use the services to access the security database. ().

Embedded access to Classic server on POSIX is the exception—see *Vulnerabilities*, below.

It became quite easy to make any changes to the way passwords are hashed - it is always performed by the server. It is no longer *gsec*'s problem to calculate the hash for the security database: it simply asks services to do the work!

It is worth noting that the new *gsec* works successfully with older Firebird versions, as long as the server's architecture supports services.

Better Password Encryption

Password encryption/decryption now uses a more secure password hash calculation algorithm.

The SHA-1 Hashing Algorithm

Although it does not increase resistance to a brute-force attack aimed at happening on the right string, the conditions created by the hashing mechanism make it harder for a brute-force attacker to analyse the password by visual means, viz..

- A hash that is valid for one user is invalid for any other, even if they are using the same string as their passwords.
- When a user changes his password—even to exactly the same string as before—the encrypted data stored in **`RDB$USERS.RDB$PASSWD`** is new and unique

Users can modify their own passwords

The SYSDBA remains the keeper of the security database. However, users can now modify their own passwords.

Non-SYSDBA users can no longer see other users' accounts. A non-privileged user can retrieve or modify only its own account.

Protection from Brute-Force Hacking

Current high-speed CPUs and fast WAN connections make it possible to try to brute-force Firebird server users' passwords. This is especially dangerous for Superserver which, since Firebird 1.5, performs user authentication very fast. Classic is slower, since it has to create new process for each connection, attach to the security database within that connection and compile a request to the table RDB$USERS before validating login and password. Superserver caches the connection and request, thus enabling a much faster user validation.

Given the 8-byte maximum length of the traditional Firebird password, the brute-force hacker had a reasonable chance to break into a Firebird Superserver installation.

Active Protection

The v.2.0 Superserver has active protection to make a brute-force attack more difficult. After four failed attempts to log in, the user and IP address are locked out for eight seconds. Any attempt to log in with that particular user name OR from that particular IP address will be denied during that period.

No setup or configuration is required for this feature. It is active automatically as soon as the Firebird 2.0 SuperServer starts up. Attempts to get access to the server using brute-force techniques on accounts and passwords are now detected and locked out.

- Login with password is required from any remote client
- Clients making too many wrong login attempts are blocked from further attempts for a period

Support for brute-force attack protection has been included in both the attachment functions of the Firebird API and the Services API.

Remote Access Restrictions

Remote attachments to the server without a login and password are now prohibited

For attachments to Superserver, even root trying to connect locally without “localhost:” in the database file string will be rejected by the remote interface if a correct login is not supplied.

Embedded access without login/password works as previously.

- On Windows, authentication is bypassed.
- On POSIX, the Unix user name is used to validate access to database files.

Trusted User Authentication on Windows

Windows “Trusted User” security can be applied for authenticating Firebird users on a Windows host. The Trusted User's security context is passed to the Firebird server and, if it succeeds, it is used to determine the Firebird security user name..

When a client connects to a server on a Windows host, simply omitting the username and password from the database or services parameter blocks will automatically cause Windows Trusted User authentication to be applied, in almost all cases.

Illustration

Suppose you have logged in to the Windows server SRV as user 'John'. If you connect to server SRV with isql, without specifying a Firebird user name and password:

```
isql srv:employee
```

and do:

```
SQL> select CURRENT_USER from rdb$database;
```

you will get something like:

```
USER
====================================================
SRV\John
```

SQL Privileges

Windows users can be granted rights to access database objects and roles in the same way as regular Firebird users, emulating the capability that has been always been available users of Unix and Linux hosted Firebird databases.

Here's an environment where using SQL roles will really simplify your life!

Administrators

If a local Adminstrator or a member of the built-in Domain Admins group connects to Firebird using trusted authentication, he/she will be connected as SYSDBA. It is therefore not necessary to grant specific SQL privileges to these users. Naturally, it also means taking care to be precise about dispensing domain privileges to your network users.

Configuring Authentication

Because Trusted User authentication is enabled by default in Firebird 2.1 and higher, it might introduce security vulnerabilities on networks where Windows security is not well controlled. If you have doubts about the security of the Windows network where you are deploying a Firebird 2.1 server, you can reconfigure the user authentication mode in *firebird.conf* to disable it.

The new parameter for configuring the authentication method on Windows is ***Authentication***. The default setting, *mixed*, allows both Windows Trusted User authentication and Firebird's native authentication using a Firebird security login:

```
#Authentication = mixed
```

To configure it so that only Firebird security logins are allowed, providing full compatibility with previous Firebird versions and avoiding trusted authentication altogether, delete the '#' symbol and change the setting to

```
Authentication = native
```

To use only Trusted User authentication and ignore Firebird login security altogether—which may offer better security than *native* if security on the Windows network is properly hardened—change the setting to

```
Authentication = trusted
```

To retain the legacy behaviour, when the **ISC_USER** and **ISC_PASSWORD** variables are set in the environment, they are picked and used *instead of* trusted authentication. However, trusted authentication can be coerced to override the environment variables if they are set—discussed next.

Forcing Trusted Authentication

For the situation where trusted authentication is needed and there is a likelihood that the **ISC_USER** and **ISC_PASSWORD** variables are set, there is a new DPB parameter that you can add to the DPB—*isc_dpb_trusted_auth*.

Most of the Firebird command-line utilities support this parameter by means of the switch **-tru[sted]** (the abbreviated form is available, according to the usual rules for abbreviating switches).

Example

```
C:\Pr~\bin>isql srv:db          -- log in using trusted authentication

C:\Pr~\bin>set ISC_USER=user1

C:\Pr~\bin>set ISC_PASSWORD=12345

C:\Pr~\bin>isql srv:db          -- log in as 'user1' from environment

C:\Pr~\bin>isql -trust srv:db   -- log in using trusted authentication
```

The *qli* and *nbackup* utilities do not follow the pattern: they use single-letter switches that are somewhat arcane. The switch of interest for *qli* is **-K**). The facility to force trusted authentication is yet to be implemented for *nbackup*.

Windows rules for full domain user names allow names longer than the maximum 31 characters allowed by Firebird for user names. The 31-character limit is enforced and, from V.2.1, logins passing longer names are disabled. This will remain the situation until the mapping of OS objects to database objects is implemented in a later Firebird version.

Vulnerabilities

Several known vulnerabilities in the API, particularly those involving string overflows, have been closed. However, it is not a perfect world and, despite the security enhancements reported in the preceding topics, you can still find ways to expose your Firebird server to those with evil intentions. This topic highlights some obvious ones.

Classic Server on POSIX

For reasons both technical and historical, a Classic server on POSIX with embedded clients is especially vulnerable to security exposure. Users having embedded access to databases MUST be given at least read access to the security database.

This is the main reason that made implementing enhanced password hashes an absolute requirement. A malicious user with user-level access to Firebird could easily steal a copy of the security database, take it home and quietly brute-force the old DES hashes! Afterwards, he could change data in critical databases stored on that server. Firebird 2 is much less vulnerable to this kind of compromise.

But the embedded POSIX server had one more problem with security: its implementation of the Services API calls the command-line gsec, as normal users do. Therefore, an embedded user-maintenance utility must have full access to security database.

The main reason to restrict direct access to the security database was to protect it from access by old versions of client software. Fortuitously, it also minimizes the exposure of the embedded Classic on POSIX at the same time, since it is quite unlikely that the combination of an old client and the new server would be present on the production box.

Poor Password Encryption

However, the level of Firebird security is still not satisfactory in one serious respect, so please read this section carefully before opening port 3050 to the Internet.

An important security problem with Firebird still remains unresolved: the transmission of poorly encrypted passwords "in clear"across the network. It is not possible to resolve this problem without breaking old clients.

To put it another way, a user who has set his/her password using a new secure method would be unable to use an older client to attach to the server. Taking this into account with plans to upgrade some aspects of the API in the next version, the decision was made not to change the password transmission method in Firebird 2.0.

The immediate problem can be solved easily by using any IP-tunneling software (such as ZeBeDee) to move data to and from a Firebird server, for both 1.5 and 2.0. It remains the recommended way to access your remote Firebird server across the Internet.

Command-line utilities that take a *-password* parameter are vulnerable to password sniffing, especially when the utility is run from a script. As a step towards hardening against this on POSIX platforms, where the password was displayed in clear in the *ps* (process list) output and elsewhere, now it is output as an asterisk.

Server Multi-hop

Historically, InterBase supported remote server redirection, commonly known as "server multi-hop". It was broken in the original code on which Firebird was built and was restored in Firebird during Firebird 2 development. Release versions go out with this feature disabled. It can be enabled by means of a new configuration parameter named **`Redirection`** in *firebird.conf*.

For more details about Server Multi-hop, see Chapter 2, Network Setup 23.

For details of the Redirection parameter, see Chapter 36, Configuration and Special Features 198.

You should not enable server multi-hop unless you really understand its security implications.

These days, the ability to redirect requests to other servers is dangerous. Suppose you have one carefully protected Firebird server, access to which is possible from the Internet. In a situation where this server has unrestricted access to your internal LAN, it will work as a gateway for incoming requests like (using a POSIX example) `firebird.your.domain.com:internal_server:/private/mydata.fdb`.

Knowing the hostname or IP address of some internal server on your LAN is enough for an intruder: login access to the external server is not required. Such a gateway easily overrides a firewall that is protecting your LAN from outside attack.

gsec Changes

The *gsec* Authentication Manager utility has had a minor enhancement.

gsec Return Code

The *gsec* utility now returns an error code when its commands are invoked directly from the command-line. Zero indicates success; any other code indicates failure.

Chapter 35 Database-Level Security

Database-level security is achieved by granting privileges to users, roles and certain database objects on tables and some other object types. In Firebird, it is a thorough implementation of SQL privilege standards.

Improvements to Privileges

Whatever else can be said about SQL privileges, the observation that they can quickly become a bird's nest of conflicting authorities is an eternal truth. Privilege schemas need to be designed with care and perfect logic. All too often, they are anything but, having been assembled over years by multiple administrators.

One welcome addition has been made to privileges syntax—the ability to revoke an inherited privilege allowing a user to grant rights to roles that it does not own.

REVOKE ADMIN OPTION FROM

SYSDBA, the database creator, or the owner of an object can grant rights on that object to other users. However, those rights can be made inheritable, too. By using WITH GRANT OPTION, the grantor gives the grantee the right to become a grantor of the same rights in turn. This ability can be removed by the original grantor with REVOKE GRANT OPTION FROM user.

However, there's a second form that involves roles. Instead of specifying the same rights for many users (soon it becomes a maintenance nightmare) the database owner or the SYSDBA can create a role, assign a package of rights to that role and then grant the role to one or more users. Any change to the role's rights affect all those users.

By using WITH ADMIN OPTION, the grantor (typically the role creator) gives the grantee the right to become a grantor of the same role in turn. Previously, this ability could not be removed unless the original grantor fiddled with system tables directly. Now, the ability to grant the role can be removed by the original grantor with REVOKE ADMIN OPTION FROM user.

Syntax Pattern

```
REVOKE ADMIN OPTION
  ON ROLE <role-name>
   FROM <user-name> ;
```

Example

```
REVOKE ADMIN OPTION
  ON ROLE TEMP_ADMIN
   FROM TEA_PERSON ;
```

Chapter 35 Errata

Page	**Erratum**
719	In the section headed Granting Privileges (that starts on P.718) an item is missing from the syntax pattern given in the line

```
<privilege> = INSERT | DELETE | etc.
```

Insert the symbol SELECT into the group so that line reads:

```
<privilege> = SELECT | INSERT | DELETE |
UPDATE [(column [,column [,...]] ) ] |
REFERENCES [(column [,column [,...]] ) ] | EXECUTE
```

Chapter 36 Configuration and Special Features

Most elements of server-level configuration are determined by settings for parameters in the file **`firebird.conf`**, in your installation's root directory.

Configuration Parameters

In Firebird 2.0 and 2.1, some important new parameters were added, some existing ones were modified and a few were deprecated.

When altering parameters, make it a rule always to read the documentation that precedes each actual parameter in the **`firebird.conf`** file. In new releases and in sub-releases, it is highly likely that the notes will change (usually for the better!)

New Parameters

The following new parameters were added to the configuration options in Firebird 2.1.

TempCacheBlockSize and ***TempCacheUpperLimit***

From V.2.1 forward, the SortMem* parameters were renamed using terms considered more appropriate to what they actually affect:

- *SortMemBlockSize* was changed to ***TempCacheBlockSize***
- *SortMemUpperLimit* was changed to ***TempCacheUpperLimit***

Authentication

This new parameter is for configuring the authentication method on Windows. The default setting, *mixed*, allows both Windows Trusted User authentication and Firebird's native authentication using a Firebird security login. Alternatives are *native*, to enforce use of Firebird logins, and *trusted*, to have only Windows Trusted User authentication. Further information can be found in Chapter 34, Server Protection 188.

RelaxedAliasChecking

A new configuration parameter added to permit a slight relaxation of the Firebird 2.0.x restrictions on mixing relation aliases and table names in a query. For example, with RelaxedAliasChecking set to true (=1) in firebird.conf, the following query will succeed in Firebird 2.1, whereas it would fail in v.2.0.x, or in v.2.1 with the parameter set to its default of 0:

```
SELECT ATABLE.FIELD1, B.FIELD2
  FROM ATABLE A JOIN BTABLE B
  ON A.ID = BTABLE.ID
```

Understand that this is a temporary facility whose purpose is to provide some headspace for migrating systems using legacy code that exploited the tolerance of InterBase and older Firebird server versions to non-standard SQL usage. It will be permanently removed from a future release

- Don't enable this parameter if you have no “offending” code in your applications or PSQL modules.

MaxFileSystemCache

Sets a threshold determining whether Firebird will allow the page cache to be duplicated to the filesystem cache or not. If this parameter is set to any (integer) value greater than zero, its effect depends on the current default size of the page cache: if the default page cache (in pages) is less than the value of MaxFileSystemCache (in pages) then filesystem caching is enabled, otherwise it is disabled.

This applies both when the page cache buffer size is set implicitly by the DefaultDBCachePages setting or explicitly as a database header attribute.

The default setting for MaxFileSystemCache is 65536 pages, i.e. filesystem caching is enabled.

- To disable filesystem caching always, set *MaxFileSystemCache* to zero
- To enable filesystem caching always, set *MaxFileSystemCache* to an integer value that is sufficiently large to exceed the size of the database page cache. Remember that the effect of this value will be affected by subsequent changes to the page cache size.

DatabaseGrowthIncrement

For better control of disk space preallocation, this new parameter DatabaseGrowthIncrement represents the upper limit for the size, in bytes, of the chunk of disk that will be requested for preallocation as pages for writes from the cache. Default: 134,217,728 bytes (128 MB).

When the engine needs to initialize more disk space, it allocates a block that is 1/16th of the space already allocated, but not less than 128 KB and not greater than the DatabaseGrowthIncrement value. The DatabaseGrowthIncrement value can be raised to increase the maximum size of newly-allocated blocks to more than the default 128 MB. Set it to zero to disable preallocation.

- The lower limit of the block size is purposely hard-coded at 128 Kb and cannot be reconfigured.
- Space is not preallocated for database shadow files.
- Preallocation is disabled for a database that has the “No reserve” option set.

BugCheckAbort

(Linux only) Provides the capability to make the server stop trying to continue operation after a bugcheck and instead, to call abort() immediately and dump a core file. Since a bugcheck usually occurs as a result of a problem the server does not recognise, continuing operation with an unresolved problem is not usually possible anyway, and the core dump can provide useful debug information.

In the more recent Linux distributions the default setups no longer dump core automatically when an application crashes. Users often have troubles trying to get them working. Differing rules for Classic and Superserver, combined with a lack of consistency between the OS setup tools from distro to distro, make it difficult to help out with any useful "general rule".

Code has been added for Classic and Superserver on Linux to bypass these problems and automate generation of a core dump file when an abort() on BUGCHECK occurs. The Firebird server will make the required 'cwd' (change working directory) to an appropriate writable location (/tmp) and set the core file size limit so that the 'soft' limit equals the 'hard' limit.

Default setting is disabled (0) for production release versions and enabled (1) for debug versions.

If you need to enable the facility, don't forget that the server needs to be restarted to activate a parameter change.

The following new parameters were added to the configuration options in Firebird 2.0.

LegacyHash

This parameter enables you to configure Firebird 2 to reject an old DES hash in an older security database that has been restored into Firebird 2 and upgraded to the new security2.fdb structure.

The installation default value is 1 (true, i.e., always reject the DES hash when using), which is intended to be an interim value for installations that are migrating a security database from a Firebird 1.5.4 or older version. As soon as users have changed their passwords in the new security database structure (thus removing the old DES hash strings stored) the parameter must be set to 0, or commented out.

If you are not in the situation of needing to use the security database upgrade procedure, this parameter does not affect Firebird operation, since a DES hash cannot arrive in the new security2.fdb.

Refer to *Upgrading Your Security Database* in Chapter 1, *Installation*, for instructions on how to upgrade your existing security.fdb (from Firebird 1.5.x) or isc4.gdb (from Firebird 1.0.x or InterBase).

Redirection

Parameter for controlling redirection of remote requests. It controls the "server multi-hop" capability that was broken in InterBase 6 and is restored in Firebird 2. By default, it is disabled (set to 0, False).

For information about multi-hop, refer to the topic *Server Multi-hop Capability* 23 in Chapter 2, *Network Setup*.

For information about the risks of using redirection, refer to the *Vulnerabilities* 192 topic in Chapter 34, *Server Protection*.

GCPolicy

Sets the garbage collection policy for Superserver installations. The possible settings are **`cooperative`**, **`background`** and **`combined`**.

For information, see *Reworking of Garbage Collection* 78 in Chapter 15, *Creating and Maintaining a Database*.

This setting has no effect on a Classic server installation, since Classic supports only cooperative garbage collection.

OldColumnNaming

The parameter OldColumnNaming was introduced at Firebird 1.5.3 and was ported forward to Firebird 2.0. It allows the server to revert to pre-V1.5 column naming behaviour in SELECT expressions, whereby the engine would not attempt to supply run-time identifiers, e.g., CONCATENATION, for derived fields where the developer has neglected to provide identifiers.

The installation default is 0 (disabled).

This setting affects all databases on the server and will potentially produce exceptions or unpredicted results where mixed applications are implemented.

UsePriorityScheduler

Setting this parameter to zero now disables switching of thread priorities completely. Its default setting is 1 (True).

It affects only the Win32 SuperServer. If you have problems with computer response time when running Superserver stand-alone on a workstation, using this parameter to turn off the thread scheduler may be effective in gaining more frequent CPU slices for Superserver threads.

Changed Parameters

The following parameters have changed in usage or default value since Firebird 1.5.x.

TempDirectories

The parameter TempDirectories was partly broken in V.2.1. The correct usage of this parameter is to supply a semicolon-separated list of paths to locations that the server is to use in preferential order for storing the temporary intermediate sets for sort operations when RAM is too low to store them there.

During V.2.1.3 development, a regression was discovered, whereby the engine would no longer hand on to subsequent locations if it found insufficient disk space in the first (leftmost) location configured for *TempDirectories*. Because of the impact on other code in V.2.1.x, the problem is flagged as a "known issue" and is not to be fixed in the V.2.1 series.

For V.2.1.x servers it is therefore essential to ensure that the first (leftmost) location is always adequate to accommodate the largest intermediate sets for ORDER BY, GROUP BY and DISTINCT operations.

LockHashSlots

This parameter is used for tuning the lock hash list. Under heavy load, throughput might be improved by raising the number of hash slots to disperse the list in shorter hash chains. The value is integer; prime number values are recommended.

This parameter and the LockMemSize (see the next section) should be evaluated at the same time, using the Lock Print tool. If the lock hash chains are longer than 20 on average, the number of hash slots is too small. If you need to increase the number of hash slots, you should increase the lock table size by the same percentage.

- In Firebird 2.1 the default was raised from 101 to **1009**.
- The previous maximum number of hash slots available was 2048. From v.2.1 onward, it is 65,536. However, because the actual setting should be a prime number, the exact supported maximum is 65,521 (the biggest prime number below 65,536).
- The minimum is 101.

LockMemSize

This integer parameter represents the number of bytes of shared memory allocated to the memory table used by the lock manager. For a Classic server, the LockMemSize gives the initial allocation, which will grow dynamically until memory is exhausted (“Lock manager is out of room” does not mean somebody went for coffee!) This parameter’s value is related to the size of the database cache, since each page will require a separate lock in table. When the number of pages of database cache is set to a high value, it often causes problems with the lock memory table.

In Superserver, the memory allocated for the lock manager does not grow.

The previous default size on Linux and Solaris was 98,304 bytes (96K) and on Windows, 262,144 (256K). From v.2.1 onward, it is 1 MB on all platforms.

IpcName

For Windows local connections, this is the name of the shared memory area used as a transport channel. Note that the local protocol in v2.0 is not compatible with any previous version of Firebird or InterBase.

Value is a string. The default value (previously FirebirdIPI) has changed:

```
#IpcName = FIREBIRD
```

If your host operating system is Windows Vista, 2003 or XP with Terminal Services enabled and you need the local connection to work for a non-privileged user, you may need to uncomment the default entry (above) and add the prefix **`Global\`** to it. As the prefix is case-sensitive, it should be exactly **`Global\FIREBIRD`**. However, from v.2.1.3, with the correct version of the client library, the prefix will be applied automatically if it is needed.

ExternalFileAccess

Modified in Firebird 2, to allow the first path cited in *ExternalFileAccess* to be used as the default when a new external file is created.

TCPNoNagle

The default value for TcpNoNagle is now TCP_NODELAY enabled (1), not disabled (0) as it was in Firebird 1.5.x.

Removed or Deprecated Parameters

CreateInternalWindow

The option *CreateInternalWindow* is no longer required to run multiple server instances and it has been removed.

DeadThreadsCollection

The *DeadThreadsCollection* parameter is no longer used at all and Firebird 2.0 silently ignores it . Dead threads are now efficiently released "on the fly", making configuration unnecessary.

External Code Modules

The FBUDF external function library no longer depends on the Firebird client library to be built.

Operating Firebird Databases on a Raw Device

Since Firebird 2, on Linux it has been possible to place a Firebird database on a raw storage device and access it independently of the file system. For the Firebird 2.1 development cycle and release, the feature was rigorously tested and got the Seal of Approval from Firebird's Linux platform gurus.

Why Operate on a Raw Device?

File system I/O can degrade performance severely when a database in Forced Writes mode grows rapidly. On Linux, which lacks the appropriate system calls to grow the database efficiently, performance with Forced Writes can be as much as three times slower than with asynchronous writes.

When such conditions prevail, performance may be greatly enhanced by bypassing the file system entirely and restoring the database directly to a raw device.

Moving a Database to a Raw Device

A Firebird database can be recreated on any type of block device. Moving your database to a raw device can be as simple as restoring a backup directly to an unformatted partition in the local storage system. For example,

```
gbak -c my.fbk /dev/sda7
```

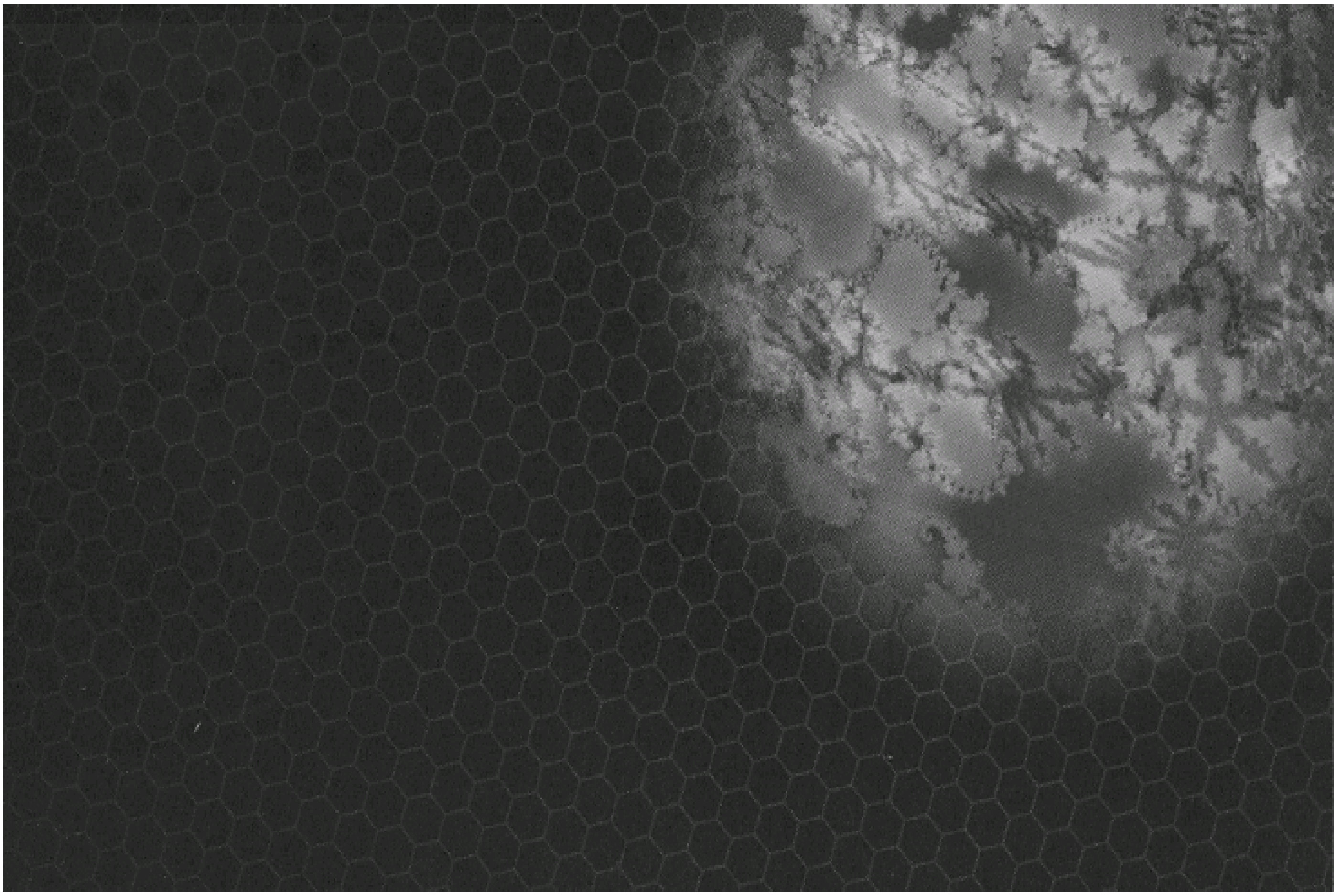
will restore your database on the third logical disk in the extended partition of your first SCSI or SATA hard-drive (disk0).

The database does not have a "database name" other than the device name itself. In the example given, the name of the database is '/dev/sda7'.

Further Advice

- There are some issues to pay attention to if you are using the *nbackup* utility to keep incremental backups of a database on a raw device. Refer to the topic *Database on a Raw Device* 217 in Chapter 38, *Database Backup and Restore*.

- This feature is not known to be possible on a Windows platform.
- Although no other specific issues are known at this point about the use of raw device storage for databases, keep in mind that the growth and potential growth of the database is less obvious to end-users than one that lives as a file within a file system. If control of the production system's environment is out of your direct reach, be certain to deploy adequate documentation for any monitoring that will be required!
- Maintain your raw devices in *aliases.conf*. That way, in the event of needing to reconfigure the storage hardware, there will be no need to alter any connection strings in your application code.



Part Nine

Tools

Topics in This Part

Chapter 37
Interactive SQL Utility (isql)

Work on *isql* has involved a lot of bug-fixing and the introduction of a few new, useful features.

One trick to note is that CHAR and VARCHAR types defined in character set OCTETS (alias BINARY) now display in hexadecimal format. Currently, this feature cannot be toggled off.

New Features

Output from a SELECT in an interactive isql session can now be stopped using Ctrl-C. Note, this merely stops fetching rows from the buffer, it does not cancel the query.

Some useful new switches and commands were added to *isql*.

Switches

-nodbtriggers

A new parameter was added to suppress database triggers 170 from running. It is available only to the database owner and SYSDBA.

-b[ail] “Bail out” Mode

The new command line switch **`-b`** instructs *isql* to bail out on error when it is running in non-interactive mode, returning an error code to the operating system.

When using scripts as input in the command line, it may be appropriate to make *isql* stop executing a batch of commands after an error has happened. The **`-b[ail]`** option will cause script execution to stop at the first error it detects. No further statements in the input script will be executed and isql will return an error code to the operating system.

Example

```
isql -b -i my_fb.sql -o results.log -m -m2
```

- Most cases have been covered, but if you find some error that is not recognised by *isql*, you should file a report in the Firebird Project's bug tracker, as this is a feature in progress.
- Currently there is no differentiation by error code—any non-zero return code should be interpreted as failure. Depending on other options (like **`-o`**, **`-m`** and **`-m2`**) , *isql* will show the error message on screen or

will send it to a file.

Some Features

- Even if isql is executing nested scripts, it will cease all execution and return to the operating system when it detects an error. Scripts are nested when a script A is used as -i[nput] in the command-line invocation and script A, in turn, contains an INPUT command to load script B...and so on.

isql itself has no capability to detect either direct or indirect recursion. If the programmer makes a mistake and script A loads itself or loads script B that, in turn, reloads script A, *isql* will run until it exhausts memory or an error is returned from the database. If *-bail* is active, all activity will stop as soon as an error is raised.

- DML errors will be caught when being prepared or executed, depending on the type of error.
- In many cases, *isql* will return the line number of a DML statement that fails during execution of a script.
- Under the isql default setting for auto-committing DDL statements (AUTODDL ON), DDL errors will be caught when being prepared or executed. However, if AUTO DLL is OFF, the server might not complain until the script does an explicit COMMIT call, which might involve several SQL statements.

SET BAIL Statement

"Bail-out mode" can be enabled/disabled inside a script by means of the statement

```
SET BAIL [[ON | OFF]]
```

As with other SET commands, simply using **`SET BAIL`** will toggle the bail-out mode between active and inactive.

In an interactive isql session, using **`SET`** on its own will display the state of the switch, along with all the other two-state switches supported by **`SET`** statements in *isql*. However, **`SET BAIL`** does not have any use inside an *isql* shell session, since it is designed to work only when isql is invoked non-interactively.

A non-interactive session happens when the user calls *isql* in batch mode, giving it a script as input.

If the user starts an isql shell session during which s/he executes a script with the INPUT command, this is considered an interactive session. Even though *isql* knows it is executing a script, it will not respond to a bail-out condition because it also knows that it is running the script inside an interactive session.

-m2 to Output Stats and Plans

The **`-M2`** switch is a command-line-only option to mix statistics and plans output with the other output sent to the specified **`-o[utput]`** file.

- Statistics are available if the input script contains an enabling SET STATS command
- Plans are available if the input script contains an enabling SET PLAN or SET PLANONLY command

The existing **-m** switch mixes error messages with the normal output, sending them as and when they happen to wherever the output is being redirected. The switches can be used alone or together to get the desired output.

Neither switch has an interactive SET command as a counterpart.

-r2 to Pass a Case-Sensitive Role Name

With this switch you can pass a case-sensitive role name in a command-line invocation of *isql*.

The original switch for passing a role name is **-r**. However, the role name must be one that has been declared in such a way that it is stored in the table RDB$ROLES in upper case because, whatever you supply as the role name argument for the **-r** switch, it will be converted to upper case.

With the new **-r2** switch, the role identifier passed will be exactly as you type it.

Example

1. Using the existing -r switch:

```
isql -r trusty localhost:/data/mydb.fdb -u cooldude -pas day2reme

Database:  localhost:/data/mydb.fdb, User: cooldude, Role: TRUSTY
SQL>
```

2. Using the new -r2 switch:

```
isql -r2 "trusty" localhost:/data/mydb.fdb -u cooldude -pas day2reme

Database:  localhost:/data/mydb.fdb, User: cooldude, Role: trusty
SQL>
```

Commands

The following commands have been added or enhanced.

SET HEAD[ing] toggle

Some people consider it useful to be able to do a SELECT inside isql and have the output sent to a file, for additional processing later, especially if the number of columns makes isql display impracticable. However, isql by default prints column headers and. in this scenario, they are a nuisance.

Therefore, printing the column headers—previously a fixed feature—can now be switched on or off interactively or from a script by means of the command

```
SET HEAD[ing] [ON | OFF]
```

in the *isql* shell or script.

As is the case with other SET commands, simply using **SET HEAD** will toggle the state between activated and deactivated.

There is no command line switch to toggle headings off.

Using **SET** on its own will display the state of **SET HEAD**, along with other switches that can be toggled on/off in the isql shell.

Enhancement for SHOW PROCEDURE & SHOW TRIGGER

On implementation of the capability to use domains for defining variables and arguments in PSQL code, a new field RDB$VALID_BLR was added in RDB$PROCEDURES and RDB$TRIGGERS to indicate whether the procedure/trigger is valid after an ALTER DOMAIN operation.

SHOW PROCEDURE and SHOW TRIGGER now print the value of this indicator.

SHOW SYSTEM Enhancements

The **SHOW <object_type>** command is meant to show user objects of that type. The **SHOW SYSTEM** command is meant to show system objects but, until now, it only showed system tables. Now it lists the predefined system UDFs incorporated into Firebird 2.

Use **SHOW SYSTEM COLLATIONS** to view all the Firebird-distributed character set/collation pairs that are available on the server.

SHOW COLLATIONS Command

Lists all the user-declared character set/collation pairs declared in the database.

SHOW DATABASE Enhancement

The on-disk structure (ODS) version is now returned in the SHOW DATABASE command.

```
SQL> SHOW DATABASE;
Database: localhost:c:\Program Files\Firebird\Fire~
           ~bird_2_1\examples\empbuild\employee.fdb
        Owner: SYSDBA
PAGE_SIZE 4096
Number of DB pages allocated = 260
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 190
Transaction - oldest active = 191
Transaction - oldest snapshot = 191
Transaction - Next = 194
ODS = 11.1
Default Character set: NONE
```

```
SQL>
```

SET SQLDA_DISPLAY ON/OFF

Previously available only in DEBUG builds, **SET SQLDA_DISPLAY** shows the input SQLDA parameters of INSERT, UPDATE and DELETE statements if it is active. It shows the information for raw SQLVARs. Each SQLVAR represents a field in the XSQLDA, the main structure used in the API to talk to clients transferring data into and out of the server.

Although, like many other **SET** switches, **SET SQLDA_DISPLAY** has ON/OFF states, its state is not reported in the list generate by the lone **SET** command.

SET TRANSACTION Enhancement

The SET TRANSACTION statement has been enhanced so that, now, all TPB options are supported.

For details, see Chapter 26, *Configuring Transactions* 153.

Retrieve Line Number of Script Error

In previous versions, there are ways to find out which line of a script caused an error but sometimes it can be very involved.

Now, the ability to signal the script-related line number of a failure enables the user to go to the script directly and find the offending statement. When the server provides line and column information, you will be told the exact line in the script that caused the problem. When the server only indicates a failure, you will be told the starting line of the statement that caused the failure, relative to the whole script.

This feature works even if there are nested scripts, since each file gets a separate line counter. For example, if script A includes script B (has an INPUT statement for it) and B causes a failure, the line number refers to B. When B has been read completely, *isql* continues executing A and resumes counting lines related to A.

Lines are counted according to what the underlying IO layer considers separate lines. For ports using EDITLINE, a line is what readline() provides in a single call. The line length limit of 32767 bytes remains unchanged.

Example

A script named blah.sql, with an error:

```
create table blah (
  id BigInt not null,
  item varchar(20),
  CHARACTER_LENGTH INTEGER); -- reserved word!
```

```
SQL> input d:\data\scripts\blah.sql;
Statement failed, SQLCODE = -104
Dynamic SQL Error
-SQL error code = -104
-Token unknown - line 4, column 3
-CHARACTER_LENGTH
At line 4 in file d:\data\scripts\blah.sql
SQL>
```

Enhanced Command-line Help

The help display that appears on the console when isql is invoked with unknown (or no) switches now shows all of the command-line switches, with descriptions, insteadof just a simple list of allowed switches.

```
opt/firebird/bin] isql -?
Unknown switch: ?
usage: isql [options] [<database>]
-a(all) extract metadata incl. legacy non-SQL tables
-b(ail) bail on errors (set bail on)
-c(ache) <num> number of cache buffers
-ch(arset) <charset> connection charset (set names)
-d(atabase) <database> database name to put in script creation
-e(cho) echo commands (set echo on)
-ex(tract) extract metadata
-i(nput) <file> input file (set input)
-m(erge) merge standard error
-m2 merge diagnostic
-n(oautocommit) no autocommit DDL (set autoddl off)
-now(arnings) do not show warnings
-o(utput) <file> output file (set output)
-pag(elength) <size> page length
-p(assword) <password> connection password
-q(uiet) do not show the message  “Use CONNECT... “
-r(ole) <role> role name
-r2 <role> role (uses quoted identifier)
-sqldialect <dialect> SQL dialect (set sql dialect)
-t(erminator) <term> command terminator (set term)
-u(ser) <user> user name
-x extract metadata
-z show program and server version
```

Chapter 38 Database Backup and Restore

Backup Tools

Firebird 2 brings enhancements and more options for backing up databases: the new on-line incremental backup utility ***NBackup*** and some improvements to the standard *gbak* utility.

NBackup & Nbak

(from original notes by Nickolay Samofatov)

The new on-line, page-level incremental backup engine comprises two parts:

- Nbak, the engine support module
- NBackup, the tool that does the actual backups

Nbak

Nbak is responsible for

1. redirecting writes to difference files when asked. The statement that initiates this is:

```
ALTER DATABASE BEGIN BACKUP
```

2. producing a GUID for the database snapshot and writing it into the database header before the statement call returns
3. merging differences into the database when asked. The statement that makes this happen is:

```
ALTER DATABASE END BACKUP
```

4. marking pages written by the engine with the current *page scan counter* value (SCN) for the data base
5. incrementing the SCN on each change of backup state

The Backup State Cycle

The backup state cycle is:

nbak_state_normal ⇒ nbak_state_stalled ⇒ nbak_state_merge ⇒ nbak_state_normal

Normal state: writes go directly to the main database files.

Stalled state: writes go to the difference file only and the main files are read-only.

Merge state: new pages are not allocated from difference files. Writes go to the main database files. Reads of mapped pages compare both page versions and return the version which is fresher, because it is not known whether it is merged.

This merge state logic has a quirk. Both Windows and Linux define the contents of file growth as "undefined", i.e., garbage, and both zero-initialize them. Because of this, mapped pages beyond the original end of the main database file are not read but are kept current in a difference file until the end of a merge. This is almost half of Nbak fetch and write logic, tested by using modified PIO on existing files containing garbage.

NBackup

NBackup, the tool interface for the Nbak engine, is responsible for

1. providing a convenient way to issue the **ALTER DATABASE BEGIN/END BACKUP** statement
2. getting the database back to normal after filesystem copying is done, by physically changing *nbak_state_diff* to *nbak_state_normal* in the database header
3. creating incremental backups and restoring from them

The incremental backups are multi-level. That means if you do a Level 2 backup every day and a Level 3 backup every hour, each Level 3 backup contains all pages changed from the beginning of the day till the hour when the Level 3 backup is made.

Backing Up

When NBackup creates incremental backups it follows this sequence:

1. Issues **ALTER DATABASE BEGIN BACKUP** to redirect writes to the difference file
2. Looks up the SCN and GUID of the most recent backup at the previous level
3. Streams to the backup file all database pages having a SCN larger than was found at step 2
4. Writes the GUID of the previous-level backup to the header, to enable the consistency of the backup chain to be checked during restore
5. Issues **ALTER DATABASE END BACKUP**
6. Adds a record of this backup operation to **RDB$BACKUP_HISTORY**, recording current level, SCN, snapshot GUID and some miscellaneous stuff for user consumption.

Restoring

Restoring is simply reconstructing the physical database image for the chain of backup files, checking that the *backup_guid* of each file matches *prev_guid* of the next one, then changing its state in the header to *nbak_state_normal*.

Usage

```
nbackup <options>
```

Valid Options

-L <*database*> Lock database for filesystem copy

-N <*database*> Unlock previously locked database

-F <*database*> Fixup database after filesystem copy

-B <*level*> <*database*> [<*filename*>] Create incremental backup

-R <*database*> [<*file0*> [<*file1*>...]] Restore incremental backup

-U <*user*> User name

-P <*password*> Password

-T <*database*> Suppress database triggers (requires SYSDBA privileges)

- <*database*> may specify a database alias
- **`stdout`** may be used as a value of <*filename*> for the -B option
- Incremental backups of multi-file databases are not supported yet

---

Database on a Raw Device

Not applicable to Windows platforms

When the database is operating from a raw device 203, *nbackup* must be supplied with an explicit file path and name for its difference file, in order to avoid this file being written into the /dev/ directory. You can achieve this with the following statement, using isql:

```
# isql /dev/sda7

SQL> alter database add difference file '/tmp/dev_sda7';
```

To keep the size of the *nbak* copy within reasonable bounds, it is of benefit to know how much storage on the device is actually occupied. Use the '-s' switch to return the size of the database in database pages:

```
# nbackup -s -l /dev/sda7

77173
```

Don't confuse the result here with the block size of the device. The figure returned—77173—is the number of pages occupied by the database. Calculate the physical size (in bytes) as (number of pages * page size). If you are unsure of the page size, you can query it from the database header using *gstat -h*:

```
# gstat -h /dev/sda7
```

```
Database "/dev/sda7"
Database header page information:
        Flags                   0
        Checksum                12345
        Generation              43
        Page size               4096  <——
        ODS version             11.1
 . . . . . . .
```

Examples of Usage with a Raw Device

1. A backup can be performed in a script, using the output from the '-s' switch directly. For example,

```
# DbFile=/dev/sda7
# DbSize=`nbackup -L $DbFile -S` || exit 1
# dd if=$DbFile ibs=4k count=$DbSize | # compress and record DVD
# nbackup -N $DbFile
```

2. A physical backup using nbackup directly from the command line:

```
# nbackup -B 0 /dev/sda7 /tmp/lvl.0
```

User Manual

A user manual for NBak/NBackup can be downloaded from the documentation area at the Firebird website: the URL for downloading the English version is http://firebirdsql.org/pdfmanual/Firebird-nbackup.pdf. It is viewable in HTML, too, in several languages besides English.

gbak Backup/Porting/Restore Utility

gbak is the standard tool for both online and offline backups of Firebird databases. It underwent some rigorous bug-fixing and cleanup during Firebird 2 development. Most visible is the changed behaviour of the restore switch *-r[eplace_database]* which, in the past, has caused grief for many unsuspecting folk, who used it without understanding that its first action is to delete the existing database file from disk!

New Restore Switch

The new gbak switch

```
-RECREATE_DATABASE [OVERWRITE]
```

is a separate switch designed to make harder for those unsuspecting people to overwrite a database accidentally, as could occur easily with the shortened form of the old switch:

```
-R[EPLACE_DATABASE]
```

Now, **`gbak -R`** (or **`gbak -r`**) applies to the new **`-R[ECREATE_DATABASE]`** switch and will never overwrite an existing database if the **`O[VERWRITE]`** argument is absent.

Change to -REPLACE_DATABASE

The short form of the old gbak **-R[EPLACE_DATABASE]** is now **-REP[LACE_DATABASE]**. This switch does not accept the **O[VERWRITE]** argument.

The **-REP[LACE_DATABASE]** switch should be considered as deprecated, i.e. it will become unavailable in some future Firebird release.

This change means that, if you have any legacy batch or cron scripts that rely on "gbak -r" or "gbak -R" without modification, then the operation will except if the database exists.

If you want to retain the ability of your script to overwrite your database unconditionally, you will need to modify the command to use either the new switch with the OVERWRITE argument or the new short form for the old -REPLACE_DATABASE switch.

Suppress Database Triggers

A new switch **-nodbtriggers** added suppress database triggers from running. It is available only to the database owner and SYSDBA

Some *gbak* Tweaks

:The latest evolution *gbak* can be used to restore a database on any version of Firebird.

:Some misbehaviours that could occur when the Services Manager was doing backup/restore operations and some parameter items were missing or in the wrong sequence were fixed for the v.2.1 release.

- The problem still affects v.2.0.x and lower versions, so care should be taken to specify all required switches and supply the database name and backup file spec in the correct order when using the **-se[rvice_mgr]** switch with these older versions.

On POSIX, gbak now changes param0 to prevent the user name and password being displayed in the output of **ps -axf**.

Chapter 38 Errata

Page	Erratum
819	In table 38-1, in the "SWITCH" column, the following is incorrect: `-pa[ssword] password` It should read `-pas[sword] password`
819 824	In the "SWITCH" column of both tables, the following is incorrect: `-v[erbose]` It should read `-v[erify]`
819 824	In each table, in the "Effect" text for the –y switch, the references to "the –v[erbose] switch" should be corrected to "the –v[**erify** switch]".
824	In table 38-2, in the "SWITCH" column, the following is incorrect: `-p[age_size] n` o It should read `-pa[ge_size] n` [[... continued ...]] and pa[ssword] should be **–pas**[sword] o The leading hyphen is missing here, too.
826	Under the heading "Single Volume Backup to Multiple-Volume Database" the examples are wrong. Replace the POSIX example with the following: `.gbak -c /backups/stocks.fbk data/stocks_trial.fdb 500000 /data/stocks/stocks_trial.fd1 -user SYSDBA -password millp0nd -v -y /logs/backups/stocks_r.20040703.log`

Replace the Windows example with :

```
.gbak -c e:\backups\stocks.fbk d:\data\stocks_trial.fdb 500000
  d:\data\stocks\stocks_trial.fd1 -user SYSDBA -password millp0nd
    -v -y e:\logs\backups\stocks_r.20040703.log
```

---

828 Under the heading "Using gbak with the Firebird Service Manager", the following is incorrect:

When you run gbak with the -service_mgr switch....

It should read

When you run gbak with the **-service** switch....

---

829 Under the heading "Using gbak with the Firebird Service Manager", the following code sample is incorrect:

```
gbak -b -se hotchicken:service_mgr
  hotchicken:d:\data\stocks.fdb
    f:\backups\stocks.20040705.fbk
      -v -y f:\backups\logs\stocks.20040715.log
```

It should read:

```
gbak -b -se hotchicken:service_mgr
  d:\data\stocks.fdb
    f:\backups\stocks.20040705.fbk
      -v -y f:\backups\logs\stocks.20040715.log
```

Chapter 39
Housekeeping Tools
(gfix & fbsvcmgr)

New Services Tool *fbsvcmgr*

Firebird 2.1 brings the new utility *fbsvcmgr*, a command-line interface enabling access to any service that is implemented in Firebird's Services API. It provides a much-needed front-end through which the Services API functions and parameters can pass, for when bare-bones access to services is needed using a text-only connection.

Using fbsvcmgr

This is not an end-user tool. The *fbsvcmgr* interface does not emulate the switches implemented in the traditional “g*” utilities. Users need to be familiar with the latest version of the Services API. The API header file, *ibase.h*, in the *../include* directory of your Firebird installation, should be regarded as the primary source of information about what is available, backed up by the *InterBase 6.0 beta API Guide.*

SPB Syntax

The SPB syntax that *fbsvcmgr* understands closely matches with what you would encounter in the *ibase.h* include file or the InterBase 6.0 API documentation, except that a slightly abbreviated form is used to reduce typing and shorten the command lines a little. Here's how it works.

Parameters

Specify the Services Manager

The first required parameter for a command line call is the Services Manager you want to connect to:

- For a local connection use the simple symbol service_mgr
- To attach to a remote host, use the format hostname:service_mgr

Specify subsequent service parameter blocks (SPBs)

Subsequent SPBs, with values if required, follow. In *ibase.h*, all SPB parameters have one of two forms:

1. isc_spb_VALUE
2. isc_VALUE1_svc_VALUE2.

You just need to pick out the VALUE, VALUE1 or VALUE2 part[s] when you supply your parameter. Accordingly, for (1) you would type simply the VALUE part, while for (2) you would type VALUE1_VALUE2.

For example:

isc_spb_dbname => dbname

isc_action_svc_backup => action_backup

isc_spb_sec_username => sec_username

isc_info_svc_get_env_lock => info_get_env_lock

and so on.

---

isc_spb_user_name can be specified as either *user_name* or simply *user*.

Making Commands Readable

Any SPB can be optionally prefixed with a single '-' symbol. For the long command lines that are typical for fbsvcmgr, use of the '-' improves the readability of the command line. Compare, for example, the following (each a single command line despite the line breaks printed here):

```
# fbsvcmgr service_mgr user sysdba password masterke
        action_db_stats dbname employee sts_hdr_pages
```

and

```
# fbsvcmgr service_mgr -user sysdba -password masterke
        -action_db_stats -dbname employee -sts_hdr_pages
```

Some Syntax Specifics

It is not realistic to attempt to describe all of the SPB parameters here. In the InterBase 6.0 beta documentation it takes about 40 pages! However, there are some known differences between the operation of *fbsvcmgr* and what you might otherwise infer from the old beta documentation.

- The formats described for some parameters in the InterBase 6 beta documentation are buggy. When in trouble, treat *ibase.h* as the primary source for the correct form.
- The function *isc_spb_rpr_list_limbo_trans* was omitted from the IB6 beta documentation. It is supported in *fbsvcmgr*.
- Everything to do with licensing was removed from the original InterBase 6 open source code and is therefore not supported either in Firebird or by *fbsvcmgr*.
- The old Config file view/modification functions have been unsupported since Firebird 1.5 and

are not implemented by *fbsvcmgr*.

Multiple Function Calls

With fbsvcmgr you can perform a single action—and get its results if applicable—or you can use it to retrieve multiple information items from the Services Manager. You cannot do both in a single command.

For example, to list all current users on the local Firebird server, you could do:

```
# fbsvcmgr service_mgr -user sysdba -password masterke
    -action_display_user
```

It could return output similar to this:

```
SYSDBA                        Sql Server Administrator      0    0
QA_USER1                                                    0    0
QA_USER2                                                    0    0
QA_USER3                                                    0    0
QA_USER4                                                    0    0
QA_USER5                                                    0    0
GUEST                                                       0    0
SHUT1                                                       0    0
SHUT2                                                       0    0
QATEST                                                      0    0
```

The following command will report both the server version and its implementation:

```
# fbsvcmgr service_mgr -user sysdba -password masterke
    -info_server_version -info_implementation

Server version: LI-T2.1.0.17798 Firebird 2.1 Final
Server implementation: Firebird/linux AMD64
```

But an attempt to mix all of this in single command line causes an error:

```
# fbsvcmgr service_mgr -user sysdba -password masterke
    -action_display_user -info_server_version -info_implementation

Unknown switch "-info_server_version"
```

Support for New Services API Items

Two new items that were added to the Services API in Firebird 2.1 are supported by *fbsvcmgr*:

- *isc_spb_trusted_auth* (type it as *trusted_auth*) applies only to Windows. It forces Firebird to use Windows trusted authentication.
- *isc_spb_dbname* (type as *dbname*) gives the ability to set a database name parameter in all service actions related to accessing the security database from a remote client. It is equivalent to supplying

the -*database* switch to the *gsec* utility.

For *gsec* the -*database* switch is mostly used to specify a remote server you want to administer. In *fbsvcmgr*, the name of the server is already given in the first parameter (via the *service_mgr* symbol) so the *dbname* parameter should be unnecessary.

Improvements to gfix

Changes to the *gfix* utility include:

Extension of Database Shutdown Modes

The options for gfix -shut[down] and gfix -o[nline]have been extended to include two extra states or modes—multi-user and full—to govern the shutdown.

Syntax Pattern

```
gfix <command> [<state>] [<options>]
<command>> ::= {-shut | -online}
<state> ::= {normal | multi | single | full}
<options> ::= {-force <timeout> | -tran | -attach}
```

- normal state = online database
- multi state = multi-user shutdown mode (the legacy one, unlimited attachments of SYSDBA/owner are allowed)
- single state = single-user shutdown (only one attachment is allowed, used by the restore process)
- full state = full/exclusive shutdown (no attachments are allowed)

"Multi" is the default state for -*shut*, "normal" is the default state for -*online*.

The modes can be switched sequentially:

normal <—> multi <—> single <—> full<—>normal

Examples

```
gfix -shut single -force 0
gfix -shut full -force 0
gfix -online single
gfix -online
```

You cannot use **-shut** to bring a database one level more "online" and you cannot use **-online** to make a database more protected (an error will be thrown).

These are prohibited:

```
gfix -shut single -force 0
gfix -shut multi -force 0
gfix -online
gfix -online full
gfix -shut -force 0
gfix -online single
```

Chapter 39 Errata

Page **Erratum**

---

839 Section "Shutting down a database"
The *gbak* utility is improperly referenced (twice) as the means of shutting down and restarting a database. This should be the **gfix** utility instead.

---

841 Section "Qualifying Arguments"

The second example reads

```
gfix -sh -f  800
```

Because gfix has two switches starting with the letter 'f', this example will cause an error. It should read

```
gfix -sh -force 800
```

---

843 The following text/code is incorrect:

Under the heading "Changing the Database Dialect", the abbreviation for the gfix -sql_dialect switch is wrong in both places it is used:

```
gfix -s[ql_dialect] n db_name
```

should read

```
gfix -sql[_dialect] n db_name
```

and, beneath it:

```
./gfix -s 3 /data/accounts.fdb
```

should read

```
./gfix -sql 3 /data/accounts.fdb
```

Chapter 40 Understanding the Lock Manager

Increased Lock Table Limits & Defaults

The V.2.1 release addresses some lock table capacity problems that have been getting progressively irksome for high-load deployments.

Attribute	***firebird.conf* Parameter**
Maximum number of hash slots raised from 2048 to 65,536. Because the actual setting should be a prime number, the exact supported maximum is 65,521 (the biggest prime number below 65,536). The minimum is 101. Default number of hash slots is now 1009 (previously 101).	LockHashSlots 201
Default lock table size has been increased to 1 Mb on all platforms (Previously 96 Kb on POSIX, 256 Kb on Windows)	LockMemSize 201

Chapter 40 Errata

Page	Erratum
860	In Table 40-1, the following text is incorrect: Prints out the lock table header block....lock resource type that you want to report on. Refer to Table 40-10 for the numbers. It should read: Prints out the lock table header block....lock resource type that you want to report on. Refer to Table **40-5** for the numbers

Chapter 41
Database Monitoring

Firebird 2.1 introduces the ability to monitor server-side activity happening inside a particular database. The engine can now deliver a set of so-called "virtual" tables on demand from a database of ODS 11.1 or higher, providing snapshots of the current activity within a database.

Virtual monitoring tables exist only in ODS 11.1 (and higher) databases. While a Firebird 2.1 server can attach to a database with a lower ODS, the internal structures of legacy databases do not have the metadata for generating the virtual tables that the monitoring routines populate. A full backup/restore migration, as described in Chapter 38 of *The Firebird Book* (p.815), is required in order to enable this feature.

How Monitoring Works

The monitoring feature delivers an *activity snapshot* representing the current state of the database. Outputs comprise a variety of information about the database itself, active attachments and users, transactions, prepared and running statements, and much more.

The tables are "virtual" insofar as no data are materialised in them until explicitly requested. Metadata for the virtual tables is persistent, however, from the schema of the databases. Relation names for these tables all begin with "MON$".

An activity snapshot is isolated inside one transaction. It is created as soon as the first SELECT.. request on any MON$ table is received and it is preserved until that transaction ends. Even if you sent your request in a READ COMMITTED or SNAPSHOT ("concurrency") transaction, internally the transaction's isolation is escalated to a level that behaves like a SNAPSHOT TABLE STABILITY ("consistency") transaction. In this way, multi-table queries such as master-detail joins and those reading dependent data via subqueries, will maintain a consistent view of database state.

Refreshing this stable snapshot requires ending the transaction and requerying the MON$ tables in a new transaction context.

Security and Scope

Only SYSDBA and the database owner have access to what is happening in all attachments to the database. Regular users are restricted to the information about their own attachments only and cannot see any data from other clients.

Monitoring Multiple Connections

Until sub-release 2.1.2, an ordinary user could monitor only the CURRENT_CONNECTION. From V.2.1.2, that ordinary user can monitor all connections logged in under the same user name.

Access to the MON$ tables is available not just in DSQL. The metadata definitions are available to PSQL as well, which means it is very feasible to delegate your frequently-used monitoring routines to stored procedures. For your convenience, the structures of the MON$ tables are described at the end of this chapter 234.

Using MON$

A valid database connection is required in order to retrieve the monitoring data. Although creation of a snapshot is usually quite a fast operation, some delay might be expected under high load, on a Classic server particularly.

The monitoring tables materialised in the attached database return only information about that database: there is no way at this point in the implementation for one database to garner MON$ data from other databases. If the server is serving multiple databases, each one has to be connected to and monitored separately.

"Monitoring tables" is a bit of a mouthful, especially if English isn't your native language! "MON-dollar" will do just fine.

Excluding the "Me" Connection

Usually you will want to exclude the MON$ connection and its own activities from the information you want to collect and analyse. The caller of the MON$ process can, of course, uniquely identify itself through the context variables **CURRENT_CONNECTION** and **CURRENT_TRANSACTION**. Since the same IDs will show up in the corresponding MON$ tables, the monitoring transaction can therefore use them to establish keys for excluding information about itself and its own connection.

Usage Examples

To give a feeling for the information you can retrieve, the following are some simple examples to get started with.

An Attachment Query

Suppose you want to retrieve the process IDs of all of the Classic server instances currently consuming CPU resources. We have a field in MON$ATTACHMENTS that can tell us which processes are busy and which are idle:

```
SELECT MON$SERVER_PID
  FROM MON$ATTACHMENTS
  WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION -- exclude Me
  AND MON$STATE = 1
```

Who is Doing What from Where?

You might want to know what clients are connected to the database and, if they are connected through a v.2.1 or higher client version, their network address and what applications they are using:

```
SELECT
  MON$USER,
  MON$REMOTE_ADDRESS,
  MON$REMOTE_PID,
  MON$TIMESTAMP
FROM MON$ATTACHMENTS
  WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION -- exclude Me
```

Clients that are v.2.0.x or lower cannot provide information about the application or their network address. The relevant fields will be NULL. In a different query you might like to use that situation to find out how many of the currently connected clients are using old versions of the client library! The time the connection started (MON$TIMESTAMP) and the user name and/or role name might help with this kind of forensic search.

What is My Isolation Level?

This time, instead of excluding the Me transaction, you explicitly want to know something about it:

```
SELECT MON$ISOLATION_MODE
  FROM MON$TRANSACTIONS
  WHERE MON$TRANSACTION_ID = CURRENT_TRANSACTION
```

What Statements are Active?

This query is a little more adventurous. Two MON$ tables are joined to extract quite detailed information about what statements are running and whose workstations requested them:

```
SELECT ATT.MON$USER,
       ATT.MON$REMOTE_ADDRESS,
       STMT.MON$SQL_TEXT,
       STMT.MON$TIMESTAMP
  FROM MON$ATTACHMENTS ATT
    JOIN MON$STATEMENTS STMT
      ON ATT.MON$ATTACHMENT_ID = STMT.MON$ATTACHMENT_ID
  WHERE ATT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION -- exclude Me
    AND STMT.MON$STATE = 1
```

What's Happening with Executable Code on the Server?

You can use a CTE to do a comprehensive query on all the PSQL modules that are executing:

```
WITH RECURSIVE HEAD AS
  (
     SELECT CALL.MON$STATEMENT_ID,
            CALL.MON$CALL_ID,
            CALL.MON$OBJECT_NAME,
            CALL.MON$OBJECT_TYPE
     FROM MON$CALL_STACK CALL
       WHERE CALL.MON$CALLER_ID IS NULL
     UNION ALL
       SELECT CALL.MON$STATEMENT_ID,
              CALL.MON$CALL_ID,
              CALL.MON$OBJECT_NAME,
              CALL.MON$OBJECT_TYPE
       FROM MON$CALL_STACK CALL
         JOIN HEAD
           ON CALL.MON$CALLER_ID = HEAD.MON$CALL_ID
  )
  SELECT MON$ATTACHMENT_ID,
         MON$OBJECT_NAME,
         MON$OBJECT_TYPE
  FROM HEAD
    JOIN MON$STATEMENTS STMT
      ON STMT.MON$STATEMENT_ID = HEAD.MON$STATEMENT_ID
    WHERE STMT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

Cancelling a Running Query

Firebird has no API method for a transaction to stop wild or long-running queries that result from its own requests. However, the MON$ system now enables such problem queries to be terminated from a separate, dedicated connection, i.e., one that is not engaged in any regular client activities. The SYSDBA or the database owner can make use of data available in the monitoring tables to isolate the offending query and devise an appropriate mechanism for reining it in.

Example

As a very rough example, the following statement will kill all statements currently running in the database, other than any that belong to the Me connection:

```
delete from mon$statements
  where mon$attachment_id <> current_connection
```

Naturally, this won't normally be what you want to do! Study the metadata carefully, along with the data you are seeing through MON$. You will be able to figure out the most effective ways to isolate the problem queries in your system and target them for cancellation.

Metadata of the MON$ Tables

Table 41-1. Structure of MON$ Tables

MON$DATABASE (connected database)		
MON$DATABASE_NAME	VARCHAR(253)	Database pathname or alias
MON$PAGE_SIZE	SMALLINT	Page size
MON$ODS_MAJOR	SMALLINT	Major ODS version
MON$ODS_MINOR	SMALLINT	Minor ODS version
MON $OLDEST_TRANSACTION	INTEGER	OIT number (oldest [interesting] transaction)
MON$OLDEST_ACTIVE	INTEGER	OAT number (Oldest active transaction)
MON $OLDEST_SNAPSHOT	INTEGER	OST number (Oldest Snapshot, i.e., the number of the OAT when the last garbage collection was done)
MON $NEXT_TRANSACTION	INTEGER	Next transaction number
MON$PAGE_BUFFERS	INTEGER	Number of pages allocated in the pagecache
MON$SQL_DIALECT	SMALLINT	SQL dialect of the database
MON$SHUTDOWN_MODE	SMALLINT	Current shutdown mode: 0: online 1: multi-user shutdown 2: single-user shutdown 3: full shutdown
MON$SWEEP_INTERVAL	INTEGER	Sweep interval
MON$READ_ONLY	SMALLINT	Read-only flag
MON$FORCED_WRITES	SMALLINT	Synchronous writes flag
MON$RESERVE_SPACE	SMALLINT	Reserve space flag
MON$CREATION_DATE	TIMESTAMP	Creation date/time, i.e., when the database was created or last restored.
MON$PAGES	BIGINT	Number of pages allocated on disk. Multiply by page size to estimate the on-disk size of the database at snapshot time. Note that a database on a raw device always returns 0.
MON$BACKUP_STATE	SMALLINT	Current state of physical backup (nbackup)

Table 41-1. Structure of MON$ Tables		
		0: normal 1: stalled 2: merge
MON$STAT_ID	INTEGER	Statistics ID
MON$ATTACHMENTS (connected attachments)		
MON$ATTACHMENT_ID	INTEGER	Attachment ID
MON$SERVER_PID	INTEGER	Server process ID
MON$STATE	SMALLINT	Attachment state 0: idle 1: active
MON $ATTACHMENT_NAME	VARCHAR (253)	Connection string
MON$USER	CHAR(93)	User name
MON$ROLE	CHAR(93)	Role name
MON $REMOTE_PROTOCOL	VARCHAR(8)	Remote protocol name
MON$REMOTE_ADDRESS	VARCHAR (253)	Remote address
MON$REMOTE_PID	INTEGER	Remote client process ID, contains non-NULL values only if the client library is version 2.1 or higher
MON$REMOTE_PROCESS	VARCHAR (253)	Remote client process pathname. Contains non-NULL values only if the client library is version 2.1 or higher. Can contain a non-pathname value if an application has specified a custom process name via the DPB.
MON $CHARACTER_SET_ID	SMALLINT	Attachment character set
MON$TIMESTAMP	TIMESTAMP	Connection date/time
MON $GARBAGE_COLLECTION	SMALLINT	Garbage collection flag, indicates whether GC is allowed for this attachment (as specified via the DPB in isc_attach_database).
MON$STAT_ID	INTEGER	Statistics ID

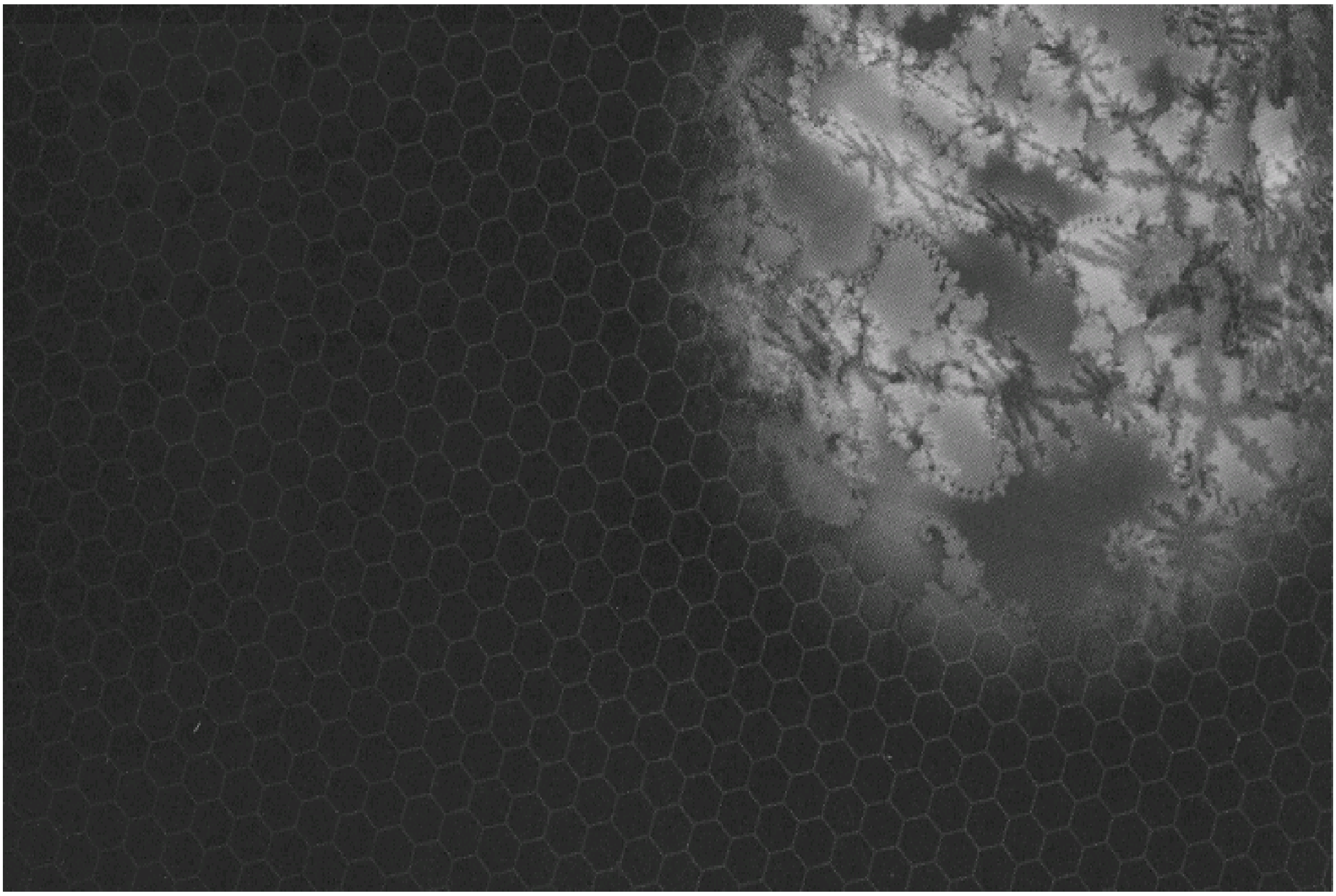
Table 41-1. Structure of MON$ Tables		
MON$TRANSACTIONS (started transactions)		
MON$TRANSACTION_ID	INTEGER	Transaction ID
MON$ATTACHMENT_ID	INTEGER	Attachment ID
MON$STATE	SMALLINT	Transaction state 0: Idle (transaction has one or more statements that are prepared and waiting to execute and no statements with open cursors) 1: Active (transaction has one or more statements executing or fetching or with pending inserts, updates or deletes)
MON$TIMESTAMP	TIMESTAMP	Transaction start date/time
MON$TOP_TRANSACTION	INTEGER	ID of Top transaction, the upper limit used by the sweeper transaction when advancing the global OIT. All transactions above this threshold are considered active. It is normally equivalent to the MON$TRANSACTION_ID but COMMIT RETAINING or ROLLBACK RETAINING will cause MON $TOP_TRANSACTION to remain unchanged ("stuck") when the transaction ID is incremented.
MON $OLDEST_TRANSACTION	INTEGER	Local OIT number (i.e., the OAT as known within the transaction's own isolation context).
MON$OLDEST_ACTIVE	INTEGER	Local OAT number
MON$ISOLATION_MODE	SMALLINT	Isolation level 0: consistency 1: concurrency 2: read committed record version 3: read committed no record version
MON$LOCK_TIMEOUT	SMALLINT	Lock timeout -1: infinite wait 0: no wait N: timeout configured as N seconds
MON$READ_ONLY	SMALLINT	Read-only flag
MON$AUTO_COMMIT	SMALLINT	Auto-commit flag
MON$AUTO_UNDO	SMALLINT	Auto-undo flag, indicates the auto-undo status set for the transaction, i.e., whether a transaction-level savepoint was created. The existence of the transaction-level savepoint allows changes to be undone if ROLLBACK is called and the transaction is then just committed. If this savepoint does not exist, or it does exist but the number of changes is very large, then an actual ROLLBACK is executed and the the

Table 41-1. Structure of MON$ Tables		
		transaction is marked in the transaction inventory (TIP) as “dead”.
MON$STAT_ID	INTEGER	Statistics ID
MON$STATEMENTS (prepared statements)		
MON$STATEMENT_ID	INTEGER	Statement ID
MON$ATTACHMENT_ID	INTEGER	Attachment ID
MON$TRANSACTION_ID	INTEGER	Transaction ID. contain valid values for active statements only
MON$STATE	SMALLINT	Statement state 0: Idle (state after prepare, until execution begins) 1: Active (state during execution and fetch. Idle state returns after cursor is closed)
MON$TIMESTAMP	TIMESTAMP	Statement start date/time. Values are valid for active statements only
MON$SQL_TEXT	BLOB TEXT	Statement text, if appropriate, contains NULL for GDML statements
MON$STAT_ID	INTEGER	Statistics ID
NOTE :: The execution plan and the values of parameters are not available.		
MON$CALL_STACK (call stack of active PSQL requests)		
MON$CALL_ID	INTEGER	Call ID
MON$STATEMENT_ID	INTEGER	Top-level DSQL statement ID, groups call stacks by the top-level DSQL statement that initiated the call chain. This ID represents an active statement record in the table MON $STATEMENTS.
MON$CALLER_ID	INTEGER	Caller request ID
MON$OBJECT_NAME	CHAR(93)	PSQL object name
MON$OBJECT_TYPE	SMALLINT	PSQL object type 2: trigger 5: procedure
MON$TIMESTAMP	TIMESTAMP	Request start date/time

Table 41-1. Structure of MON$ Tables

MON$SOURCE_LINE	INTEGER	SQL source line number, contain line/column information related to the PSQL statement currently being executed
MON$SOURCE_COLUMN	INTEGER	SQL source column number. contain line/column information related to the PSQL statement currently being executed
MON$STAT_ID	INTEGER	Statistics ID
MON$IO_STATS (I/O statistics)		
MON$STAT_ID		statistics ID
MON$STAT_GROUP	INTEGER	Statistics group 0: database 1: attachment 2: transaction 3: statement 4: call
MON$PAGE_READS	SMALLINT	Number of page reads
MON$PAGE_WRITES	BIGINT	Number of page writes
MON$PAGE_FETCHES	BIGINT	Number of page fetches
MON$PAGE_MARKS	BIGINT	Number of pages with changes pending
MON$RECORD_STATS (record-level statistics)		
MON$STAT_ID	INTEGER	Statistics ID
MON$STAT_GROUP	SMALLINT	Statistics group) 0: database 1: attachment 2: transaction 3: statement 4: call
MON $RECORD_SEQ_READS	BIGINT	Number of records read sequentially
MON $RECORD_IDX_READS	BIGINT	Number of records read via an index
MON$RECORD_INSERTS	BIGINT	Number of inserted records
MON$RECORD_UPDATES	BIGINT	Number of updated records
MON$RECORD_DELETES	BIGINT	Number of deleted records

Table 41-1. Structure of MON$ Tables		
MON $RECORD_BACKOUTS	BIGINT	Number of records where a new primary record version or a change to an existing primary record version is backed out due to rollback or savepoint undo
MON$RECORD_PURGES	BIGINT	Number of records where record version chain is being purged of versions no longer needed by OAT or younger transactions
MON $RECORD_EXPUNGES	BIGINT	Number of records where record version chain is being deleted due to deletions by transactions older than OAT



Appendices

Appendix I
Function Summary

Until Firebird 2, the majority of common string, arithmetic and trigonometric functions were not implemented internally but were available instead as external functions—popularly known as UDFs—that would be optionally declared to a database if needed. In Firebird 2.0, the process of internalising the functions distributed in the *ib_udf* external function library began with the implementation of LOWER(), TRIM() and the three string-size functions BIT_LENGTH, CHAR_LENGTH/CHARACTER_LENGTH and OCTET_LENGTH.

With Firebird 2.1 came the internalisation of dozens more erstwhile external functions from both *ib_udf* and other popular libraries.

External function support has not been abandoned. There are still many useful libraries out there with specialised functions and, in any case, people will always want the benefit of being able to write a custom function or to implement a common function in a non-standard way. If you need to continue using a UDF in preference to the corresponding internal functions, drop its declaration and redeclare it with a double-quoted function name.

Indeed, some of the functions in the existing libraries deployed with Firebird have been improved, including the new capability for some traditional string UDFs to take NULL arguments.

! The choice between UDF and built-in function is decided when compiling the statement. If the statement is compiled in a PSQL module whilst the UDF is available in the database, then the module will continue to require the UDF declaration to be present until it is next recompiled.

External Function Improvements

UDFs added or enhanced in Firebird 2.0's supplied libraries are:

IB_UDF	**srand()**
Linux, Win32	Returns a random number between 0 and 1
Arguments	No arguments, but parentheses are required
Return value	A FLOAT number
Notes	In previous versions, the external function *rand()* sets the random number generator's starting point based on the current time and then generates the pseudo-random value. The problem with this algorithm is that it will return the same value for two calls done within a second. To work around this issue, rand() was changed in Firebird 2.0 so that the starting point is not set explicitly. This ensures that different values will always be returned. In order to keep the legacy behaviour available in case somebody needs it, srand() has been introduced. It does exactly the same as the old *rand()* did.
Example	`SELECT SRAND() AS RandomNumber`

```
from RDB$DATABASE;
```

Related or similar functions	See also *RAND()*

IB_UDF	**"LOWER"(value)**
Linux, Win32	Returns the input string as lower-case characters. It works only with ASCII characters.
Arguments	value is a column or expression that evaluates to an ASCII string of 32,765 bytes or less.
Return value	A CHAR(n) or VARCHAR(n) of the same size as the input string.
Notes	Substitutes only the declared function name—by enclosing it in double quotes— for the previous LOWER() function (entry_point IB_UDF_lower) which could conflict with the new internal function *lower()*, if you try to declare it in a database using the **ib_udf.sql** script.

```
/* New declaration in ib_udf2.sql */
DECLARE EXTERNAL FUNCTION "LOWER"
CSTRING(255) NULL
RETURNS CSTRING(255) FREE_IT
ENTRY_POINT 'IB_UDF_lower' MODULE_NAME 'ib_udf';
```

Example	The following string will return the string 'come and sit at my table':

```
SELECT "LOWER" ('Come and sit at MY TABLE')
  as L_STRING
FROM RDB$DATABASE;
```

Related or similar functions	It is recommended to avoid using the UDF functions and to use the new internal function LOWER(). Amongst other benefits, the internal function behaves correctly with non-ASCII character sets and follows the declared COLLATE rules if they are in the picture.

IB_UDF	**ASCII_CHAR, LOWER, "LOWER", LPAD, LTRIM, RPAD, RTRIM, SUBSTR and SUBSTRLEN**
Linux, Win32	
Arguments	Modified to accept NULL signalling.
Return value	As before (refer to the corresponding Appendix in *The Firebird Book*).
Notes	These string functions have been enhanced to take a NULL signal on their inputs and have it interpreted correctly. Refer to the topic *NULL Signalling* in Chapter 21, *Expressions and Predicates*, for details of the changes and how to upgrade legacy metadata to use the Firebird 2 enhancements.
Example	Now, to behave correctly when passing NULL, these functions must be declared as in this example for ASCII_CHAR():

IB_UDF — **ASCII_CHAR, LOWER, "LOWER", LPAD, LTRIM, RPAD, RTRIM, SUBSTR and SUBSTRLEN**

```
DECLARE EXTERNAL FUNCTION ascii_char
       INTEGER NULL
       RETURNS CSTRING(1) FREE_IT
       ENTRY_POINT 'IB_UDF_ascii_char'
       MODULE_NAME 'ib_udf';
```

New or Enhanced Internal Functions

The new internally implemented functions, along with existing functions that have been enhanced, are:

ABS(value)

Returns the absolute value of a number.

Arguments: *value* is any value or expression that resolves to a signed or unsigned number

Return value: An unsigned number of the same type

Notes

Example

```
SELECT ABS(SUM(ASSET_VALUE)) AS LIABILITY
  FROM ASSET_REGISTER
  WHERE ASSET_VALUE < 0;
```

Related or similar functions: See also SIGN().

ACOS(value)

Returns the arccosine (inverse of a cosine) of *value*.

Arguments: *value* is a number or expression that resolves to a signed or unsigned DOUBLE PRECISION number between -1 and 1.

Return value: Is in radians: a DOUBLE PRECISION number in the range 0 to *pi*.

Example

```
IF (NEW.X IS NOT NULL) THEN
  NEW.Y = ACOS(NEW.X);
```

Related or similar functions: See also COS(), COSH() and other trigonometric functions.

ASCII_CHAR(value)

Returns the ASCII character with the specified code.

Arguments — *value* must be an integer within the range 0 to 255.

Return value — The result is returned as a single-byte character in character set NONE.

Example

```
UPDATE EXT_TABLE
  SET EOL = ASCII_CHAR(13) || ASCII_CHAR(11);
```

Related or similar functions — See also ASCII_VAL().

ASCII_VAL(value)

Returns the ASCII code of the first character of the specified string.

Arguments — *value* must be a string of 0 to 32,767 bytes.

Return value — Returns an integer in the range 0 to 255—an ASCII decimal code.

Notes — Returns 0 if the string is empty; throws an exception if the first character is multi-byte.

Example

```
IF (ASCII_VAL(A || B) IS NULL) THEN
  EXCEPTION NameElementIsMissing;
```

Related or similar functions — See also ASCII_CHAR().

ASIN(value)

Returns the arcsine of *value*.

Arguments — *value* must be a number or numeric expression that resolves to a number in the range -1 to 1.

Return value — Returns a number in the range -(*pi*/2) to (*pi*/2).

Example

```
IF (NEW.X IS NOT NULL) THEN
  NEW.READING1 = ASIN(NEW.X);
```

Related or similar functions — See also SIN() and other trigonometric functions.

ATAN(value)

Returns the arctangent of *value*.

Arguments — *value* must be a number or numeric expression.

Return value — Returns a number in the range -(*pi*/2) to (*pi*/2).

Example

```
IF (NEW.X IS NOT NULL) THEN
  NEW.READING1 = ATAN(NEW.X);
```

Related or similar functions — See also ATAN2(), TAN() and other trignonometric functions.

2.1

ATAN2(value1,value2)

Returns the arctangent of *value1* divided by *value2*.

Arguments — *value1* and *value2* must be numbers or numeric expressions. *value2* must not resolve to zero.

Return value — Returns a number in the range -*pi* to *pi*.

Example

```
IF (NEW.A IS NOT NULL AND NEW.B IS NOT NULL) THEN
  NEW.READING1 = ATAN2(NEW.A,NEW.B);
```

Related or similar functions — See also ATAN(), TAN() and other trigonometric functions.

2.1

BIN_AND(value1, value2 [,...valueN])

Performs a binary AND operation on two or more numbers.

Arguments — *value1*, *value2*, etc. must be numbers or numeric expressions.

Return value — Returns the result of the binary AND operation.

Notes — After the v.2.1 release, the function was modified to disallow a list of fewer than two arguments.

Example

```
SELECT BIN_AND(current_flags, old_flags, 1) AS new_flags
  FROM aTable;
```

Related or similar functions — See also BIN_OR(), BIN_SHL(), BIN_SHR(), BIN_XOR(), BIN_NOT().

To be implemented in v.2.1.1

BIN_NOT(value)

Performs a binary NOT operation on *value*.

Arguments — *value* must be a number or numeric expression.

Return value — Returns the result of reversing the bit-order of *value*.

Notes	Not implemented in v.2.1.
Example	`SELECT BIN_NOT(current_flags) AS new_flags FROM aTable;`
Related or similar functions	See also BIN_AND(), BIN_OR(), BIN_SHL(), BIN_SHR(), BIN_XOR().

BIN_OR(value1, value2 [,...valueN])

Performs a binary OR operation on two or more numbers.

Arguments	*value1*, *value2*, etc. must be numbers or numeric expressions.
Return value	Returns the result of the binary OR operation.
Notes	After the v.2.1 release, the function was modified to disallow a list of fewer than two arguments.
Example	`SELECT BIN_OR(current_flags, 1) AS new_flags FROM aTable;`
Related or similar functions	See also BIN_AND(), BIN_SHL(), BIN_SHR(), BIN_XOR(), BIN_NOT().

BIN_SHL(value1, value2)

Performs a binary shift-left operation on two numbers.

Arguments	*value1*, *value2*, etc. must be numbers or numeric expressions.
Return value	Returns the result of *value1* << *value2*.
Example	`SELECT BIN_SHL(current_flags, 1) AS new_flags FROM aTable;`
Related or similar functions	See also BIN_AND(), BIN_OR(), BIN_SHR(), BIN_XOR(), BIN_NOT().

BIN_SHR(value1, value2)

Performs a binary shift-right operation on two numbers.

Arguments	*value1*, *value2*, etc. must be numbers or numeric expressions.
Return value	Returns the result of *value1* >> *value2*.
Example	`SELECT BIN_SHR(current_flags, 1) AS new_flags FROM aTable;`

Related or similar functions	See also BIN_AND(), BIN_OR(), BIN_SHL(), BIN_XOR(), BIN_NOT().

BIN_XOR(value1, value2 [,...valueN])

Performs a binary XOR operation on two or more numbers.

Arguments	*value1*, *value2*, etc. must be numbers or numeric expressions.
Return value	Returns the result of the binary XOR operation.
Notes	After the v.2.1 release, the function was modified to disallow a list of fewer than two arguments.
Example	`SELECT BIN_OR(current_flags, old_flags) AS new_flags FROM aTable;`
Related or similar functions	See also BIN_AND(), BIN_OR(), BIN_SHL(), BIN_SHR(), BIN_NOT().

BIT_LENGTH(value)

Returns the length of a string in bits.

Arguments	*value* is a string or expression that resolves to a string.
Return value	An integer in the range 0 to 262,136

Example

```
select
  rdb$relation_name,
  bit_length(rdb$relation_name),
  bit_length(trim(rdb$relation_name))
  from rdb$relations;
```

Related or similar functions	See also CHAR_LENGTH()/CHARACTER_LENGTH(), OCTET_LENGTH().

CEIL(value) | CEILING(value)

Returns the smallest integer number that is greater than or equal to *value*.

Arguments	*value* is any number or numeric expression.
Return value	A number of an appropriate integer type.
Notes	CEIL() and CEILING are exact synonyms.
Example	`SELECT CEILING(12.345) AS WHATEVER`

```
 from RDB$DATABASE; -- returns 13

SELECT CEIL(PI()) AS HI_PI
  from RDB$DATABASE; -- returns 4
```

Related or similar functions — See also FLOOR().

`CHAR_LENGTH(value) | CHARACTER_LENGTH(value)`

Returns the length of a string in characters.

Arguments — *value* is a well-formed string or expression that resolves to a well-formed string.

Return value — An integer in the range 0 to 32,767

Notes — CHAR_LENGTH() and CHARACTER_LENGTH() are exact synonyms. Multi-byte character set strings are treated correctly. A malformed string will cause an exception.

Example

```
select
  rdb$relation_name,
  char_length(rdb$relation_name),
  character_length(trim(rdb$relation_name))
  from rdb$relations;
```

Related or similar functions — See also BIT_LENGTH(), OCTET_LENGTH().

`COS(value)`

Returns the natural cosine of *value*.

Arguments — *value* is a real number or expression that resolves to a real number, representing the size of the angle in radians. (1 radian ≈ 57.2958 degrees).

Return value — A DOUBLE PRECISION number in the range between -1 and 1.

Example

```
IF (NEW.RADIANS IS NOT NULL) THEN
  NEW.Y = COS(NEW.RADIANS);
```

Related or similar functions — See also ACOS(), COSH() and other trigonometric functions.

`COSH(value)`

Returns the hyperbolic cosine of *value*.

Arguments

value is a real number or expression that resolves to a real number, representing the size of the angle in radians. (1 radian ≈ 57.2958 degrees).

Return value

A DOUBLE PRECISION number in the range between -1 and 1.

Example

```
IF (NEW.RADIANS IS NOT NULL) THEN
  NEW.Y = COSH(NEW.RADIANS);
```

Related or similar functions

See also COS(), ACOS() and other trigonometric functions.

COT(value)

Returns the cotangent of *value*.

Arguments

value is a real number or expression that resolves to a real number, representing the size of the angle in radians. (1 radian ≈ 57.2958 degrees).

Return value

A DOUBLE PRECISION number in the range between 0 and pi/2.

Example

```
IF (NEW.RADIANS IS NOT NULL) THEN
  NEW.Y = COT(NEW.RADIANS);
```

Related or similar functions

See also TAN(), ATAN(), ATAN2() and other trigonometric functions.

DATEADD(value2 value1 TO value3) |
DATEADD(value1, value2, value3)

Returns a date/time/timestamp value incremented or decremented by the specified measure of time.

Arguments

value1 is a string symbol representing what unit of time is to be used. It can be one of YEAR | MONTH | DAY | WEEK | HOUR | MINUTE | SECOND | MILLISECOND. It is case-insensitive.

value2 is a signed number specifying how many of the specified units of time are to be added (if positive) or subtracted (if negative).

value3 is a date/time expression, value or literal, being the value to be operated on

Return value

A date/time value of the same type as *value3*.

Notes

- The two format variants are equivalent. Note the differences in argument order and syntactical symbols.
- The MONTH symbol will cause the date to advance or retreat by calendar months.
- YEAR, MONTH and DAY cannot be used with a *value3* argument of type TIME.
- HOUR, MINUTE, SECOND and MILLISECOND cannot be used with a *value3* argument

of type DATE.

- Any *value1* argument is valid for a *value3* argument of type TIMESTAMP.
- Notice the use of the date/time type hint in the first example below. This little-known convention is available wherever you use a date/time literal in dialect 3 SQL.

Examples

```
SELECT
  DATEADD(WEEK, 1, DATE 'YESTERDAY')
  from RDB$DATABASE;
```

```
SELECT
  DATEADD((-1 YEAR TO CURRENT_DATE) as LAST_YEAR
  from RDB$DATABASE;
```

Related or similar functions

See also DATEDIFF()

DATEDIFF(value1 FROM value2 TO value3) |

DATEDIFF(value1, value2, value3)

Returns an exact numeric value representing the number of units of time elapsed between one date/time value and another.

Arguments

value1 is a string symbol representing what unit of time is to be used. It can be one of YEAR | MONTH | DAY | WEEK | HOUR | MINUTE | SECOND | MILLISECOND. It is case-insensitive.

value2 is a date/time expression, value or literal, being the first operand

value3 is a date/time expression, value or literal, being the second operand

Return value

An exact numeric representing the number of *value1* units of time, positive if *value2* is earlier than *value3*, zero if there is no calculated difference, negative otherwise.

Notes

- The two format variants are equivalent. Note the differences in syntactical symbols.
- value2 and value3 must be compatible date/time types
- The MONTH symbol will operate in calendar months.
- A DATE type cannot be compared with a TIME type
- YEAR, MONTH and DAY cannot be used with *value2* or *value3* arguments of type TIME.
- HOUR, MINUTE, SECOND and MILLISECOND cannot be used with a *value3* argument of type DATE.
- Any *value1* argument is valid where both *value2* and *value3* arguments are of type TIMESTAMP.
- Use common sense about other combinations
- Notice the use of the date/time type hint in the examples below. This little-known

convention is available wherever you use a date/time literal in dialect 3 SQL.

Examples

```
SELECT
  DATEDIFF(WEEK FROM TIMESTAMP 'NOW'
    TO TIMESTAMP '25.12.2009') AS 'WEEKS_TILL_CHRISTMAS'
  FROM RDB$DATABASE;
```

```
SELECT
  DATEDIFF(DAY, DATE 'TODAY', DATE '08.08.2008')
    AS 'DAYS_TO_OR_SINCE_OLYMPIAD_XXIX'
  FROM RDB$DATABASE;
```

Related or similar functions

See also DATEADD()

`DECODE(expr,search1,result1[,search2,result2...][,default])`

Offers a way to "power-code" a CASE ... WHEN ... ELSE expression

Arguments

expr is a required argument, a column name or expression representing the condition to test.

search1 is the target value to be tested by *expr*.

result1 is the value to return when *search1* is found.

To test multiple cases, add more search/result pairs (*search2*, *result2*, *search3*, *result3*, and so on).

default, which is optional, handles the ELSE case.

Return value

The *resultN* value that is paired with the *searchN* value, or *default* if none of the search cases is matched.

Notes

Count the arguments carefully!

Example

```
SELECT DECODE(SERVER_TYPE,
  1, 'Firebird',
  2, 'InterBase',
  'Unspecified') AS "Server Type"
FROM TARGET_SERVERS;
```

Related or similar functions

See also IIF(). Conditional expressions are discussed in *Using CASE() and Friends* in Chapter 21 of *The Firebird Book* (p. 409).

`EXP(value)`

Returns the exponential of *value*. Sometimes called the "anti-logarithm".

Arguments

value should be a real number.

Return value

A positive real number that is ***e*** (Euler's number, 2.718281828 approx.) raised to the power of *value*.

Example

```
SELECT EXP(X_FACTOR) AS ANTI_LOG_X
FROM RDB$DATABASE;
```

Related or similar functions

See also LN(), LOG(), LOG10().

EXTRACT(WEEK FROM value1)

Returns the week of the year for a date.

Arguments

value1 is a DATE or TIMESTAMP value or expression.

Return value

An integer in the range 1 to 53.

Notes

The WEEK option was added to the EXTRACT() function options at v.2.1.

Make sure you understand what you are getting when submitting a date that occurs around the turn of the year. Contrary to what your diary might tell you, under the ISO 8601 standards, dates may not overlap weeks in adjoining years and no date can fall into a gap between weeks. The "ISO year" starts at the first Monday of Week 1 and ends at the Sunday before the new ISO year. If 1 January is on a Monday, Tuesday, Wednesday or Thursday, it is in week 01. If 1 January is on a Friday, Saturday or Sunday, it is in week 52 or 53 of the previous year. You can identify the years that have a "Week 53" by counting the number of Thursdays in the calendar year.

Example

```
SELECT
  ...
  EXTRACT(WEEK FROM ADATE) AS WEEKOFYEAR,
  ...
FROM ATABLE;
```

Related or similar functions

See the full options for EXTRACT() in Chapter 10 of *The Firebird Book*, p.157 ff., *The EXTRACT() Function*.

FLOOR(value)

Returns the greatest integer number that is less than or equal to *value*.

Arguments

value is any number or numeric expression.

Return value

A number of an appropriate integer type.

Example

```
SELECT FLOOR(15.678) AS WHATEVER
 from RDB$DATABASE; -- returns 15
```

```
SELECT FLOOR(PI()) AS LO_PI
  from RDB$DATABASE; -- returns 3
```

Related or similar functions: See also CEIL()/CEILING().

GEN_UUID()

Returns a number that should be universally unique—a "Universally Unique Identifier".

Arguments: None.

Return value: A string of numerals, CHAR(16) CHARACTER SET OCTETS.

Example:

```
INSERT INTO ATABLE (PK, DATA)
  VALUES(GEN_UUID(), 'Some data');
```

HASH(value)

Returns the hash of a string.

Arguments: *value* is a string or string expression.

Return value: BIGINT

Example:

```
IF (NEW.ASTRING IS NOT NULL) THEN
  NEW.ASTRING_HASH = HASH(NEW.ASTRING);
```

IIF(expr, value1, value2)

Offers a way to "power-code" a CASE ... WHEN ... ELSE expression when performing a binary test.

Arguments: *expr* is a predicate representing the condition to test.

value1 is the value to return when *expr* is true; *value2* is the value to return in the ELSE case.

Return value: A result of the data type determined by *value1*.

Notes:

- All arguments are required.
- IIF() cannot be used to set conditional search or sorting criteria (WHERE, ORDER BY, GROUP BY).

Example:

```
SELECT IIF(SERVER_TYPE=1, 'Firebird', 'Other')
  AS "Server Type"
FROM TARGET_SERVERS;
```

Related or similar functions — See also DECODE(). Conditional expressions are discussed in *Using CASE() and Friends* in Chapter 21 of *The Firebird Book* (p. 409).

`LEFT(value1, value2)`

Returns a substring that is the first n characters of a specified string.

Arguments — *value1* is a string, string column or expression that resolves to a string.

value2 is an integer or integer expression specifying the number of characters to return.

Return value — A string that is *value2* characters in length.

Notes

- The first position in a string is 1, not 0.
- If the *value2* argument evaluates to a non-integer, banker's rounding is applied.

Example

```
SELECT LEFT(MEMO, 25) || '..' AS OPENING_TEXT
FROM ATABLE;
```

Related or similar functions — See also RIGHT(), SUBSTRING().

`LIST(value1 [, value2])`

Aggregate function that returns a delimited list of just about anything.

Arguments — *value1* can be any column or parameter expression, the data to be operated on

value2 is an optional string or string constant expression to be used as the delimiter if the default comma is not desired.

Return value — A TEXT BLOB in most cases; a BLOB of another subtype if the *value1*column is a BLOB of another subtype.

Notes — More completely documented in Chapter 21 in the topic *The LIST() Function* 126.

Example

```
SELECT LIST(CUST_NO, '; ') AS CLIST
FROM CUSTOMER
```

`LN(value)`

Returns the natural logarithm of a number.

Arguments — *value* is a number or numeric expression.

Return value — A real number.

Example

```
SELECT LN(aNumber) AS LOG_A
from RDB$DATABASE;
```

Related or similar functions — See also EXP(), LOG(), LOG10().

`LOG(base, value)`

Returns the logarithm of a number given a specified base.

Arguments — *base* is the base on which the logarithm calculation is to be done.

value is a number or numeric expression.

Return value — A real number.

Example

```
SELECT LOG(8, aNumber) AS LOG8_A
from RDB$DATABASE;
```

Related or similar functions — See also EXP(), LN(), LOG10().

`Log10(value)`

Returns the base 10 logarithm of a number.

Arguments — *value* is a number or numeric expression.

Return value — A real number.

Example

```
SELECT LOG10(aNumber) AS LOG10_A
from RDB$DATABASE;
```

Related or similar functions — See also EXP(), LN(), LOG().

`LOWER(value)`

Returns value converted to all lower-case characters.

Arguments — *value* must be a well-formed string or string expression of no more than 32,767 bytes.

Return value — A string of the same character set and collation as *value*.

Notes — The internal function, unlike its external counterpart "LOWER" in ib_udf, can correctly process both single-byte and multi-byte character strings.

Example

```
SELECT LOWER(RDB$RELATION_NAME)
FROM RDB$RELATIONS;
```

Related or similar functions

This function is the exact alter ego of the UPPER() function that Firebird has always had.

LPAD(value1, value2 [, value3])

Returns a string of a specified length that consists of a given string that has been left-padded with either SPACE characters (default) or a specified character.

Arguments

value1 is a string.

value2 is an integer number, being the required char_length of the string after padding.

value3 (optional) specifies the padding character[s] if the padding is to be other than a single SPACE character.

Return value

A left-padded string of *value2* characters.

Notes

The char_length of the *value1* argument should usually not be longer than the length specified by value2. If it is, the result string will be truncated to a length of *value2* characters.

Example

```
UPDATE MEMBERS
  SET MEMBERSHIP_NO = LPAD(CAST(MEM_NUM AS VARCHAR(5)), 5 ,'0');
```

Related or similar functions

See also RPAD().

MAXVALUE(value1, value2 [, ... valueN])

Returns the highest value from a list of values of comparable types.

Arguments

value1, *value2*, etc. are values of comparable types. At least two arguments are required.

Return value

The list member having the highest value, according to the data type of *value1*.

Example

```
SELECT CASE
WHEN MAXVALUE(AFIELD, BFIELD, 0.02) = AFIELD THEN 'Sample A is suspect'
WHEN MAXVALUE(AFIELD, BFIELD, 0.02) = BFIELD THEN 'Sample B is suspect'
ELSE 'Both samples OK' END AS RESULT
FROM ATABLE;
```

Related or similar functions

See also MINVALUE().

MINVALUE(value1, value2 [, ... valueN])

Returns the lowest value from a list of values of comparable types.

Arguments: *value1*, *value2*, etc. are values of comparable types. At least two arguments are required.

Return value: The list member having the lowest value, according to the data type of *value1*.

Example:

```
SELECT CASE
  WHEN MINVALUE(AFIELD, BFIELD, 0.02) = 0.02 THEN 'Both samples failed'
  WHEN MINVALUE(AFIELD, BFIELD, 0.02) = AFIELD THEN 'Sample A is OK'
  ELSE 'Sample B is OK' END AS RESULT
FROM ATABLE;
```

Related or similar functions: See also MAXVALUE().

MOD(value1, value2)

Modulo function: returns the remainder part of the division of *value1* by *value2*.

Arguments: *value1* and *value2* are numbers or numeric expressions.

Return value: A number of the type resulting from the division, it represents the result of *value1* modulo *value2*.

Example:

```
IF (MOD(TOTAL_QTY,TOTAL_KIDS) = 0) THEN
  SPARES = 'F';
```

OCTET_LENGTH(value)

Returns the length of a string in bytes.

Arguments: *value* is a string or expression that resolves to a string.

Return value: An integer in the range 0 to 32,767 or less.

Notes: The maximum size of the return value is indeterminate for some multi-byte character sets.

Example:

```
select
  rdb$relation_name,
  octet_length(rdb$relation_name),
  octet_length(trim(rdb$relation_name))
  from rdb$relations;
```

Related or similar functions: See also BIT_LENGTH, CHAR_LENGTH()/CHARACTER_LENGTH().

`OVERLAY(value1 PLACING value2 FROM value3 [FOR value4])`

Substitutes part of a string with another string.

Arguments

value1 is the string that is to be operated on.

value2 is the substitute string.

value3 is a number representing the position in value1 from where the substitution is to begin.

value4 is an optional argument enabling you to specify how many characters of *value2* to use for the substitution. If it is not specified, CHARACTER_LENGTH(*value2*) is the default.

Return value

The modified string, in effect, will be:

```
SUBSTRING(value1, 1 FOR (value3 - 1)) || value2 ||
SUBSTRING(value1, (value3 + value4))
```

Notes

- Both strings must be well-formed.
- The first position in a string is 1, not 0.
- If a *value3* or *value4* argument evaluates to a non-integer, banker's rounding is applied.

Example

```
SELECT OVERLAY ('This is an example of an upper-case sequence.'
  PLACING 'UPPER-CASED' FROM 26 FOR 10)
from RDB$DATABASE;
```

```
yields 'This is an example of an UPPER-CASE sequence.'
```

Related or similar functions

See also POSITION(), REPLACE(), SUBSTRING().

`PI()`

Returns the value of *pi*.

Arguments

None

Return value

3.1459... as a DOUBLE PRECISION number.

Example

```
UPDATE ATABLE
  SET AREA = PI() * RADIUS POWER 2;
```

`POSITION( value1 IN value2 ) |`
`POSITION( value1, value2 [, value3] )`

Returns the start position of one string within another.

Arguments

value1 is a string, being the string that the function is searching for.

value2 is a string, being the string that is being searched.

In the second syntax, *value3* is an optional integer argument to specify an offset position in *value2* so that the function returns the position of *value1* relative to that offset position.

Return value — A 4-bit integer.

Notes

- Both strings must be well-formed.
- The first position in a string is 1, not 0.

Examples

```
SELECT RDB$RELATION_NAME
  FROM RDB$RELATIONS
  WHERE POSITION('RDB$' IN RDB$RELATION_NAME) = 1;

SELECT RDB$RELATION_NAME
  FROM RDB$RELATIONS
  WHERE POSITION('$', RDB$RELATION_NAME, 4) = 1;
```

Related or similar functions — See also OVERLAY(), REPLACE().

POWER(value1, value2)

Raises a number to a specified power..

Arguments — *value1* is the number that is to be operated on.

value2 is the power to which value1 is to be raised.

Return value — A number.

Example

```
UPDATE ATABLE
  SET AREA = PI() * RADIUS POWER 2;
```

Related or similar functions — See also SQRT(), EXP()

RAND()

Returns a random number.

Arguments — None.

Return value — A DOUBLE PRECISION number in the range 0 to 1.

Example

```
SELECT RAND() AS WHATEVER
from RDB$DATABASE;
```

`REPLACE(value1, value2, value3)`

Replaces all occurrences of a substring within a given string with another string.

Arguments

value1 is the string on which the search is to be executed.

value2 is the substring to search for.

value3 is the replacement string.

Return value

The modified string.

Notes

- All arguments must be well-formed strings.
- With long strings, take care that the replacement operation will not overrun the maximum OCTET_LENGTH of strings for the character set.

Example

```
SELECT
  REPLACE(PHONE_NUMBER, '-', ' ') AS MOD_PHONE
  FROM ADDRESS_BOOK;
```

Related or similar functions

See also OVERLAY().

`REVERSE(value)`

Reverses the order of the characters in a string..

Arguments

value is a string.

Return value

A string.

Notes

Useful function for creating an expression index that indexes strings from right to left.

value must be a well-formed string.

Example

```
CREATE INDEX REV_EMAIL ON ADDRESS_BOOK
COMPUTED BY (REVERSE(EMAIL));

SELECT * FROM ADDRESS_BOOK
  WHERE REVERSE(EMAIL) STARTING WITH REVERSE('.uk');
```

Related or similar functions

See also OVERLAY(), REPLACE().

`RIGHT(value1, value2)`

Returns a substring that is the last n characters of a specified string.

Arguments

value1 is a string, string column or expression that resolves to a string.

value2 is an integer or integer expression specifying the number of characters to return.

Return value — A string that is *value2* characters in length.

Notes

- The first position in a string is 1, not 0.
- If the *value2* argument evaluates to a non-integer, banker's rounding is applied.

Example

```
SELECT RIGHT(MEMO, 25) || '..' AS CLOSING_TEXT
FROM ATABLE;
```

Related or similar functions — See also LEFT(), SUBSTRING().

ROUND(value1, value2)

Rounds a number to a specified scale.

Arguments — *value1* is a number or numeric expression.

value2 is the scale of the rounded number.

Return value — A number of fixed precision.

Notes — If *value2* is negative or is omitted, the decimal part of *value1* is zeroed and the integer part is rounded. So, for example, ROUND(123.456, -1) would return 120.000.

Example

```
SELECT
  EMPLOYEE_NAME,
  SALARY,
  ROUND (SALARY * 1.1) AS ADJUSTED_SALARY
from STAFF;
```

Related or similar functions — See also TRUNCATE().

RPAD(value1, value2 [, value3])

Returns a string of a specified length that consists of a given string that has been right-padded with either SPACE characters (default) or a specified character.

Arguments — *value1* is a string.

value2 is an integer number, being the required char_length of the string after padding.

value3 (optional) specifies the padding character[s] if the padding is to be other than a single SPACE character.

Return value — A right-padded string of *value2* characters.

Notes — The char_length of the *value1* argument should usually not be longer than the length specified by value2. If it is, the result string will be truncated to a length of *value2* characters.

Example

```
UPDATE MEMBERS
  SET MEMBERSHIP_NO = RPAD(CAST(MEM_NUM AS CHAR(5)), 5 ,'-');
```

Related or similar functions: See also LPAD().

SIGN(value)

Reports the sign of a number.

Arguments: *value* is a signed number or numeric expression.

Return value: Returns a SMALLINT, 1=positive, -1=negative, or zero if the input is zero.

Example

```
SELECT
  ...,
  CASE SIGN(INCOME - EXPENSES)
    WHEN 0 THEN 'Break even'
    WHEN 1 THEN 'Profit'
    ELSE 'Loss'
  END AS RESULT,
  ...
from ATABLE;
```

Related or similar functions: See also ABS().

SIN(value)

Returns the natural sine of *value*.

Arguments: *value* is a real number or expression that resolves to a real number, representing the size of the angle in radians. (1 radian ≈ 57.2958 degrees).

Return value: A DOUBLE PRECISION number in the range between -1 and 1.

Example

```
IF (NEW.RADIANS IS NOT NULL) THEN
  NEW.Y = SIN(NEW.RADIANS);
```

Related or similar functions: See also ASIN(), SINH() and other trigonometric functions.

`SINH(value)`

Returns the hyperbolic sine of *value*.

Arguments: *value* is a real number or expression that resolves to a real number, representing the size of the angle in radians. (1 radian ≈ 57.2958 degrees).

Return value: A DOUBLE PRECISION number in the range between -1 and 1.

Example:

```
IF (NEW.RADIANS IS NOT NULL) THEN
  NEW.Y = SINH(NEW.RADIANS);
```

Related or similar functions: See also SIN(), ASIN() and other trigonometric functions.

`SQRT(value)`

Returns the square root of a number.

Arguments: *value* is a number or numeric expression.

Return value: A number.

Example:

```
SELECT SQRT(PI()) AS ANUMBER
from RDB$DATABASE;
```

Related or similar functions: See also POWER().

`TAN(value)`

Returns the natural tangent of *value*.

Arguments: *value* is a real number or expression that resolves to a real number, representing the size of the angle in radians. (1 radian ≈ 57.2958 degrees).

Return value: A DOUBLE PRECISION number in the range between 0 and *pi*/2.

Example:

```
IF (NEW.RADIANS IS NOT NULL) THEN
  NEW.Y = TAN(NEW.RADIANS);
```

Related or similar functions: See also TANH(), ATAN(), ATAN2() and other trigonometric functions.

`TANH(value)`

Returns the hyperbolic tangent of *value*.

Arguments: *value* is a real number or expression that resolves to a real number, representing the size of the angle in radians. (1 radian ≈ 57.2958 degrees).

Return value: A DOUBLE PRECISION number in the range between 0 and *pi*/2.

Example:

```
IF (NEW.RADIANS IS NOT NULL) THEN
  NEW.Y = TANH(NEW.RADIANS);
```

Related or similar functions: See also TAN(), ATAN(), ATAN2() and other trigonometric functions.

TRIM([value1] [value2] [FROM] value3)

Trims unwanted characters from right, left or both sides of a string..

Arguments: *value1* is an optional keyword specifying the trimming required:

- LEADING to trim only from the left side
- TRAILING to trim only from the right side
- BOTH to trim both sides

The default is BOTH and may be omitted.

value2 is an optional argument (string or string expression) to specify which character[s] to strip off. The default is the SPACE character and may be omitted.

value3 is the input string or string expression.

Return value: A string.

Notes:

- Include the keyword FROM if either *value1* or *value2* is explicitly specified; otherwise, omit it.
- All arguments must be well-formed strings.

Examples:

```
select
  rdb$relation_name,
  trim(leading 'RDB$' from rdb$relation_name)
from rdb$relations
  where rdb$relation_name starting with 'RDB$';

select
  trim(rdb$relation_name) || ' is a system table'
  from rdb$relations
where rdb$system_flag = 1;
```

TRUNC(value1 [, value2])

Returns the integral part of a number, optionally truncated to a specified scale.

Arguments

value1 is a number or numeric expression.

value2 is a signed integer, optionally specifying a scaled truncation.

Return value

A number of the same type as *value1*.

Notes

The scale argument (*value2*) was added at v.2.1.

Examples

```
SELECT
  TRUNC(-10,32) AS TRUNC1,
  TRUNC(10.32) AS TRUNC2
FROM RDB$DATABASE
 -- returns -2, 2

SELECT
  TRUNC(987.65, 1) AS TRUNC1,
  TRUNC(987.65, -1) AS TRUNC2
FROM RDB$DATABASE
 -- returns 987.60, 980.00
```

Related or similar functions

See also ROUND().

Appendix I Errata

Page	Erratum
887 891 892	On a few pages, “Linux” is misspelt as “Linus”.

Appendix II
Solving Network Problems

No supplementary information.

Appendix III
Application Interfaces

Firebird 2.0 was a major release that changed or added many features that could affect application interfaces. Many of the most popular driver layers are known to have been updated to support the changes (Jaybird, the Firebird .NET Provider, the Firebird ODBC driver, IB Objects and FIB Plus for Delphi, and more).

If you are migrating your applications to Firebird 2.x, now would be a highly appropriate time to test your existing applications in the lab and prepare to upgrade your interface layers to resolve incompatibilities.

Appendix IV
Database Repair
How-to

No supplementary information.

Appendix V Administration Tools

Many of the more popular database administration tools for Firebird have released in new versions to support changes and enhancements in version 2.x. If you are using an older version of your favourite, it is recommended that you prepare to upgrade in order to get access to the new features.

The cross-platform, open source database admin utility for Firebird, FlameRobin, has matured much during Firebird 2 development. If it has been a while since you tried it, give it a whirl now with the newest Firebird release.

URL: http://www.flamerobin.org/

Appendix VI
The Sample Database

In Firebird 2, the sample database, **`employee.fdb`**, is installed in a different place from where it was in previous versions.

Taking $firebird as the root directory of the installation, you will find the database in **`$firebird/examples/empbuild`**. It is the same old **`employee.fdb`** that we know and hate and it has been upgraded to ODS 11.

Appendix VII
Firebird Limits

Some limits have increased for ODS-11+ databases.

Page Cache

The maximum number of pages (buffers) that can be allocated to the page cache was increased to 128,000 pages (2Gb when the page size is 16Kb).

Index Size

The old aggregate key length limit of 252 bytes is removed. Now the limit depends on page size: the maximum size of the key in bytes is one-quarter of the page size, extending the limit to 1,024 bytes with a 4Kb page size, 2,048 bytes on 8Kb pages, and so on.

Rows in a Table

The previous table size limit of around 30 Gb has been removed by the introduction of 40-bit (internally, 64-bit) record enumerators.

Appendix VIII Character Sets and Collations

TABLE VIII-1 LISTS THE NEW CHARACTER SETS AND COLLATIONS implemented in Firebird 2.0. The table will be updated as additions are made throughout the Firebird 2.x to 3.0 development process.

Table VIII-1. New Character Sets and Collations, Firebird 2.1

NAME	COLLATION	LANGUAGE
CP943C	CP943C_UNICODE	Japanese character set
ISO8859_1	ES_ES_CI_AI	Spanish language case- and accent-insensitive collation
	FR_FR_CI_AI	French language case- and accent-insensitive collation
	PT_BR	Brazil Portuguese
ISO8859_2	ISO_PLK	Polish language collation
KOI8R	KOI8-RU	Russian language character set and dictionary collation.
KOI8U	KOI8-UA	Ukrainian language character set and dictionary collation.
TIS620	TIS620_UNICODE	Thai character set, single byte
UTF8 (Unicode 4.0)	UCS_BASIC	Default. Sorts are performed in UNICODE code-point order
	UNICODE_CI	Case-insensitive collation
	UNICODE	Sorts using UCA (Unicode Collation Algorithm)
WIN1250	BS_BA	Bosnian collation
	WIN_CZ	Case-insensitive Czech language collation
	WIN_CZ_CI_AI	Case-insensitive, accent-insensitive Czech language collation
WIN1252	WIN_PTBR	Brazil Portuguese
WIN1257	WIN1257_LV	Latvian dictionary collation.
	WIN1257_LT	Lithuanian dictionary collation
	WIN1257_EE	Estonian dictionary collation
WIN1258	WIN1258	Vietnamese character set and collation

A bug, whereby UPPER did not convert aacute to Aacute for ISO8859_1, has been fixed.

Table VIII-2. Installed Character Sets and Collations, Firebird 2.1

Char. Set	Collation	Description	Attributes
ASCII	ASCII	U.S. ASCII	PAD SPACE, SYSTEM
BIG_5	BIG_5	Chinese	PAD SPACE, SYSTEM
CP943C	CP943C	Japanese, uppercase collation mapping to CP943C ibm-943_P15A-2003	PAD SPACE, SYSTEM
	CP943C_UNICODE	Same with Unicode encoding	PAD SPACE, SYSTEM
CYRL	CYRL	Cyrillic	PAD SPACE, SYSTEM
	DB_RUS	dBase Russian collation	PAD SPACE, SYSTEM
	PDOX_CYRL	Paradox Cyrillic collation	PAD SPACE, SYSTEM
DOS437	DOS437	English-USA	PAD SPACE, SYSTEM
	DB_DEU437	German	PAD SPACE, SYSTEM
	DB_ESP437	Spanish	PAD SPACE, SYSTEM
	DB_FRA837	French (France)	PAD SPACE, SYSTEM
	DB_FRC837	French (Canada	PAD SPACE, SYSTEM
	DB_FIN437	Finnish	PAD SPACE, SYSTEM
	DB_ITA437	Italian	PAD SPACE, SYSTEM
	DB_NLD437	Dutch	PAD SPACE, SYSTEM
	DB_SVE437	Swedish	PAD SPACE, SYSTEM
	DB_UK437	English (U.K.)	PAD SPACE, SYSTEM
	DB_US437	English (U.S.)	PAD SPACE, SYSTEM
	PDOX_ASCII	Paradox ASCII codepage	PAD SPACE, SYSTEM
	PDOX_INTL	Paradox international English code pages	PAD SPACE, SYSTEM
	PDOX_SWEDFIN	Swedish/Finnish	PAD SPACE, SYSTEM
DOS737	DOS737	Greek encoding	PAD SPACE, SYSTEM
DOS775	DOS775	Baltic encoding	PAD SPACE, SYSTEM
DOS850	DB_DEU850	German	PAD SPACE, SYSTEM
	DB_ESP850	Spanish	PAD SPACE, SYSTEM

Table VIII-2. Installed Character Sets and Collations, Firebird 2.1

	DB_FRA850	French (France)	PAD SPACE, SYSTEM
	DB_FRC850	French (Canada)	PAD SPACE, SYSTEM
	DB_ITA850	Italian	PAD SPACE, SYSTEM
	DB_NLD850	Dutch	PAD SPACE, SYSTEM
	DB_PTB850	Portuguese (Brazil)	PAD SPACE, SYSTEM
	DB_PTG850	Portuguese	PAD SPACE, SYSTEM
	DB_SVE850	Swedish	PAD SPACE, SYSTEM
	DB_UK850	English (U.K.)	PAD SPACE, SYSTEM
	DB_US850	English (U.S.)	PAD SPACE, SYSTEM
DOS852	DOS852	Latin II	PAD SPACE, SYSTEM
	DB_CSY	dBase Czech codepages	PAD SPACE, SYSTEM
	DB_SLO	dBase Slovakian codepages	PAD SPACE, SYSTEM
	DB_PLK	dBase Polish codepages	PAD SPACE, SYSTEM
	PDOX_CSY	Paradox Czech codepages	PAD SPACE, SYSTEM
	PDOX_HUN	Paradox Hungarian codepages	PAD SPACE, SYSTEM
	PDOX_PLK	Paradox Polish codepages	PAD SPACE, SYSTEM
	PDOX_SLO	Paradox Slovakian codepages	PAD SPACE, SYSTEM
DOS857	DOS857	Turkish codepages	PAD SPACE, SYSTEM
	DB_TRK	DBase collation for Turkish codepages	PAD SPACE, SYSTEM
DOS858	DOS858	Latin I + Euro symbol	PAD SPACE, SYSTEM
DOS860	DOS860	Portuguese codepages	PAD SPACE, SYSTEM
	DB_PTG860	Portuguese dictionary collation	PAD SPACE, SYSTEM
DOS861	DOS861	Icelandic codepages	PAD SPACE, SYSTEM
	PDOX_ISL	Iceland dictionary collation	PAD SPACE, SYSTEM
DOS862	DOS862	Hebrew	PAD SPACE, SYSTEM
DOS863	DOS863	French (Canada) codepages	PAD SPACE, SYSTEM
	DB_FRC863	French (Canada) dBase collation	PAD SPACE, SYSTEM
DOS864	DOS864	Arabic codepages	PAD SPACE, SYSTEM
DOS865	DOS865	Nordic codepages	PAD SPACE, SYSTEM
	DB_DAN865	dBase Danish collation	PAD SPACE, SYSTEM
	DB_NOR865	dBase Norwegian collation	PAD SPACE, SYSTEM
	PDOX_NORDAN4	Paradox Norwegian/Danish	PAD SPACE, SYSTEM

Table VIII-2. Installed Character Sets and Collations, Firebird 2.1

		collation	
DOS866	DOS866	Russian codepages	PAD SPACE, SYSTEM
DOS869	DOS869	Modern Greek codepages	PAD SPACE, SYSTEM
EUCJ_0208	EUCJ_0208	EUC Japanese encoding	PAD SPACE, SYSTEM
GBK	GBK	Simplified Chinese	PAD SPACE, SYSTEM
	GBK_UNICODE	Same but with Unicode mappings	PAD SPACE, SYSTEM
GB_2312	GB_2312	Simplified Chinese, 2-byte encodings (PRC and Hong Kong), a.k.a WIN_936	PAD SPACE, SYSTEM
ISO8859_1	ISO8859_1	Latin I codepages	PAD SPACE, SYSTEM
	DA_DA	Danish codepages	PAD SPACE, SYSTEM
	DE_DE	German codepages	PAD SPACE, SYSTEM
	DU_NL	Dutch codepages	PAD SPACE, SYSTEM
	EN_UK	U.K. English codepages	PAD SPACE, SYSTEM
	EN_US	U.S. English codepages	PAD SPACE, SYSTEM
	ES_ES	Spanish encoding, refer to attributes	PAD SPACE, 'DISABLE-COMPRESSIONS=1;SPECIALS-FIRST=1', SYSTEM
	ES_ES_CI_AI	Spanish encoding, refer to attributes	PAD SPACE, CASE INSENSITIVE, ACCENT INSENSITIVE, 'DISABLE-COMPRESSIONS=1;SPECIALS-FIRST=1', SYSTEM
	FI_FI	Finnish	PAD SPACE, SYSTEM
	FR_CA	French (Canada)	PAD SPACE, SYSTEM
	FR_FR	French (France)	PAD SPACE, SYSTEM
	FR_FR_CI_AI	French (France), refer to attributes	FROM EXTERNAL ('FR_FR'), PAD SPACE, CASE INSENSITIVE, ACCENT INSENSITIVE, 'SPECIALS-FIRST=1', SYSTEM
	IS_IS	Icelandic	PAD SPACE, SYSTEM
	IT_IT	Italian	PAD SPACE, SYSTEM
	NO_NO	Norwegian	PAD SPACE, SYSTEM
	PT_BR	Portuguese (Brazil), refer to attributes	PAD SPACE, CASE INSENSITIVE, ACCENT INSENSITIVE, SYSTEM
	PT_PT	Portuguese	PAD SPACE, SYSTEM
	SV_SV	Swedish	PAD SPACE, SYSTEM

Table VIII-2. Installed Character Sets and Collations, Firebird 2.1

ISO8859_2	ISO8859_2	Latin II - Central European language encodings	
	CS_CZ	Czech	CASE-SENSITIVE, PAD SPACE, SYSTEM
	ISO_HUN	Hungarian	PAD SPACE, SYSTEM
	ISO_PLK	Polish	PAD SPACE, SYSTEM
ISO8859_3	ISO8859_3	Latin III - Southern European language encodings (Maltese, Esperanto)	PAD SPACE, SYSTEM
ISO8859_4	ISO8859_4	Latin IV - Norther Europen language encodings (Estonian, Latvian, Lithuanian, Greelandic, Lappish)	PAD SPACE, SYSTEM
ISO8859_5	ISO8859_5	Cyrillic encodings	PAD SPACE, SYSTEM
ISO8859_6	ISO8859_6	Arabic encodings	PAD SPACE, SYSTEM
ISO8859_7	ISO8859_7	Greek encodings	PAD SPACE, SYSTEM
ISO8859_8	ISO8859_8	Hebrew encodings	PAD SPACE, SYSTEM
ISO8859_9	ISO8859_9	Latin V encodings	PAD SPACE, SYSTEM
ISO8859_13	ISO8859_13	Latin VII encodings - Baltic Rim	PAD SPACE, SYSTEM
	LT_LT	Lithuanian codepages	PAD SPACE, SYSTEM
KOI8R	KOI8R	Russian codepages	PAD SPACE, SYSTEM
	KOI8R_RU	Russian dictionary collation	PAD SPACE, SYSTEM
KOI8U	KOI8U	Ukrainian codepages	PAD SPACE, SYSTEM
	KOI8U_UA	Ukrainian dictonary collation	PAD SPACE, SYSTEM
KSC_5601	KSC_5601	Korean (Unified Hangeul)	PAD SPACE, SYSTEM
	KSC_DICTIONARY	Korean - dictionary order	PAD SPACE, SYSTEM
NEXT	NEXT	NeXTSTEP encoding	PAD SPACE, SYSTEM
	NXT_DEU	German	PAD SPACE, SYSTEM
	NXT_ESP	Spanish	PAD SPACE, SYSTEM
	NXT_FRA	French (France)	PAD SPACE, SYSTEM
	NXT_ITA	Italian	PAD SPACE, SYSTEM
	NXT_US	U.S. English	PAD SPACE, SYSTEM
NONE	NONE	Language-neutral single byte encodings	PAD SPACE, SYSTEM

Table VIII-2. Installed Character Sets and Collations, Firebird 2.1

OCTETS	OCTETS	Language-neutral single byte encodings	PAD ZERO, SYSTEM
SJIS_0208	SJIS_0208	Japanese 2-byte encoding	PAD SPACE, SYSTEM
TIS620	TIS620	Thai 2-byte encoding	PAD SPACE, SYSTEM
	TIS620_UNICODE	Thai Unicode encoding	PAD SPACE, SYSTEM
UTF8	UTF8	Unicode encoding	PAD SPACE, SYSTEM
	UCS_BASIC	Basic Unicode collation	PAD SPACE, SYSTEM
	UNICODE		PAD SPACE, SYSTEM
	UNICODE_CI	Case-insensitive Unicode collation	FROM EXTERNAL ('UNICODE'), PAD SPACE, CASE INSENSITIVE, SYSTEM
UNICODE_FSS	UNICODE_FSS	Deprecated fixed 3-byte Unicode encoding	PAD SPACE, SYSTEM
WIN1250	WIN1250	ANSI encoding, Central European languages	PAD SPACE, SYSTEM
	BS_BA	Bosnian	PAD SPACE, SYSTEM
	PXW_CSY	Paradox Czech codepages	PAD SPACE, SYSTEM
	PXW_HUN	Paradox Hungarian codepages	PAD SPACE, SYSTEM
	PXW_HUNDC	Paradox Hungarian, case-insensitive collation	PAD SPACE, SYSTEM
	PXW_PLK	Paradox Polish codepages	PAD SPACE, SYSTEM
	PXW_SLOV	Paradox Slovakian codepages	PAD SPACE, SYSTEM
	WIN_CZ	Czech codepages	PAD SPACE, SYSTEM
	WIN_CZ_CI_AI	Czech codepages, see attributes	PAD SPACE, CASE INSENSITIVE, ACCENT INSENSITIVE, SYSTEM
WIN1251	WIN1251	ANSI encoding, Cyrillic languages	PAD SPACE, SYSTEM
	PXW_CYRL	Paradox Cyrillic collation	PAD SPACE, SYSTEM
	WIN1251_UA	Ukrainian	PAD SPACE, SYSTEM
WIN1252	WIN1252	ANSI Latin I encoding	
	PXW_INTL850	Paradox multi-lingual Latin I	PAD SPACE, SYSTEM
	PXW_INTL	Paradox English (International)	PAD SPACE, SYSTEM
	WIN_PTBR	Portuguese (Brazil), see attributes	PAD SPACE, CASE INSENSITIVE, ACCENT INSENSITIVE, SYSTEM
WIN1253	WIN1253	ANSI Greek encoding	PAD SPACE, SYSTEM
	PXW_GREEK	Paradox Greek collation	PAD SPACE, SYSTEM
WIN1254	WIN1254	ANSI Turkish encoding	PAD SPACE, SYSTEM

Table VIII-2. Installed Character Sets and Collations, Firebird 2.1

	PXW_TURK	Paradox Turkish collation	PAD SPACE, SYSTEM
	PXW_NORDAN4	Paradox Nordic collation	PAD SPACE, SYSTEM
	PXW_SPAN	Paradox Spanish collation	PAD SPACE, SYSTEM
	PXW_SWEDFIN	Paradox Swedish and Finnish	PAD SPACE, SYSTEM
WIN1255	WIN1255	ANSI Hebrew encoding	PAD SPACE, SYSTEM
WIN1256	WIN1256	ANSI Arabic encoding	PAD SPACE, SYSTEM
WIN1257	WIN1257	ANSI Baltic encoding	PAD SPACE, SYSTEM
	WIN1257_EE	Estonian collation	PAD SPACE, SYSTEM
	WIN1257_LT	Lithuanian collation	PAD SPACE, SYSTEM
	WIN1257_LV	Latvian collation	PAD SPACE, SYSTEM
WIN1258	WIN1258	Vietnamese encoding	PAD SPACE, SYSTEM

Appendix IX System Tables and Views

System Table Structures Extended

RDB$PROCEDURES stores definitions of stored procedures. From v.2.1, ODS 11.1, it also stores a type identifier column to distinguish selectable from executable procedures.

COLUMN IDENTIFIER	TYPE	IDX	UQ	DESCRIPTION
RDB$PROCEDURE_TYPE	SMALLINT	N	N	Stores the type of stored procedure. Possible values are: 0 or NULL: legacy procedure (no validation checks are performed) 1: selectable procedure (one that contains a SUSPEND statement) 2: executable procedure (no SUSPEND statement, cannot be selected from)

RDB$TRIGGERS stores definitions of all triggers. From v.2.1, ODS 11.1, it also stores an indicator column to signal whether the trigger is valid after an ALTER DOMAIN operation.

COLUMN IDENTIFIER	TYPE	IDX	UQ	DESCRIPTION
RDB$VALID_BLR	SMALLINT	N	N	Indicates whether the trigger's BLR is still valid after an ALTER DOMAIN operation. Possible values are: 0 or NULL: BLR is invalid 1: BLR is valid

RDB$PROCEDURES stores definitions of stored procedures. From v.2.1, ODS 11.1, it also stores some new indicator columns.

COLUMN IDENTIFIER	TYPE	IDX	UQ	DESCRIPTION
RDB$PROCEDURE_	SMALLINT	N	N	Stores the type of stored procedure. Possible values are:

TYPE				0 or NULL: legacy procedure (no validation checks are performed) 1: selectable procedure (one that contains a SUSPEND statement) 2: executable procedure (no SUSPEND statement, cannot be selected from)
RDB$VALID_BLR	SMALLINT	N	N	Indicates whether the trigger's BLR is still valid after an ALTER DOMAIN operation. Possible values are: 0 or NULL: BLR is invalid 1: BLR is valid

RDB$INDEX_SEGMENTS stores the segments and positions of multi-segment indexes. Now it also stores individual segment statistics (selectivity) for each key in a multi-key index.

A new column RDB$STATISTICS has been added to the system table RDB$INDEX_SEGMENTS to store the per-segment selectivity values for multi-key indexes.

The column of the same name in RDB$INDICES is kept for compatibility and still represents the total index selectivity, that is used for a full index match.

COLUMN IDENTIFIER	TYPE	IDX	UQ	DESCRIPTION
RDB$STATISTICS	DOUBLE PRECISION	N	N	Stores the per-segment selectivity value for a key in a multi-key index. It is updated by SET STATISTICS.

RDB$GENERATORS stores names and IDs of generators. Now it also stores a description column.

COLUMN IDENTIFIER	TYPE	IDX	UQ	DESCRIPTION
RDB$DESCRIPTION	BLOB SUB_TYPE 1	N	N	Stores optional descriptive text

Example of Use

```
CREATE GENERATOR GEN_ACCOUNT_ID;
COMMIT;
UPDATE RDB$GENERATORS
  SET RDB$DESCRIPTION = 'Something informative'
  WHERE RDB$GENERATOR_NAME = 'GEN_ACCOUNT_ID';
COMMIT;
```

RDB$ROLES stores role definitions. Now it also stores a a system flag to flag user-defined roles (value 0) and a description column.

COLUMN IDENTIFIER	TYPE	IDX	UQ	DESCRIPTION
RDB$SYSTEM_FLAG	SMALLINT	N	N	Stores 0 for a user-defined role, 1 for a system-defined one
RDB$DESCRIPTION	BLOB SUB_TYPE 1	N	N	Stores optional descriptive text

Example of Use

```
CREATE ROLE AUDITOR;
COMMIT;
UPDATE RDB$ROLES
  SET RDB$DESCRIPTION = 'Something informative'
  WHERE RDB$ROLE_NAME = 'AUDITOR';
COMMIT;
```

Appendix X
Error Codes

THE ERROR CODES RETURNED TO CLIENTS or PSQL modules by Firebird 2.0 are listed in Table X-1. A few codes are unavailable to lower versions of Firebird. It is important to ensure that both server and client have the correct version of firebird.msg (interbase.msg for Firebird 2.x) stored in the Firebird root directory. It is not mandatory to install the *.msg file on clients but, if its present, it must be the correct one.

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
0	335544875	bad_debug_format	Bad debug info format
-84	335544554	nonsql_security_rel	Table/procedure has non-SQL security class defined
-84	335544555	nonsql_security_fld	Column has non-SQL security class defined
-84	335544668	dsql_procedure_use_err	Procedure %s does not return any values
-85	335544747	usrname_too_long	The username entered is too long.Maximum length is 31 bytes.
-85	335544748	password_too_long	The password specified is too long.Maximum length is 8 bytes.
-85	335544749	usrname_required	A username is required for this operation.
-85	335544750	password_required	A password is required for this operation
-85	335544751	bad_protocol	The network protocol specified is invalid
-85	335544752	dup_usrname_found	A duplicate user name was found in the security database
-85	335544753	usrname_not_found	The user name specified was not found in the security database
-85	335544754	error_adding_sec_record	An error occurred while attempting to add the user.
-85	335544755	error_modifying_sec_record	An error occurred while attempting to modify the user record.
-85	335544756	error_deleting_sec_record	An error occurred while attempting to delete the user record.
-85	335544757	error_updating_sec_db	An error occurred while updating the security database.
-103	335544571	dsql_constant_err	Data type for constant unknown

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-104	336003075	dsql_transitional_numeric	Precision 10 to 18 changed from DOUBLE PRECISION in SQL dialect 1 to 64-bit scaled integer in SQL dialect 3
-104	336003077	sql_db_dialect_dtype_unsupport	Database SQL dialect %d does not support reference to %s datatype
-104	336003087	dsql_invalid_label	Label %s %s in the current scope
-104	336003088	dsql_datatypes_not_comparable	Datatypes %sare not comparable in expression %s
-104	335544343	invalid_blr	Invalid request BLR at offset %ld
-104	335544390	syntaxerr	BLR syntax error: expected %s at offset %ld, encountered %ld
-104	335544425	ctxinuse	Context already in use (BLR error)
-104	335544426	ctxnotdef	Context not defined (BLR error)
-104	335544429	badparnum	Bad parameter number
-104	335544440	bad_msg_vec	
-104	335544456	invalid_sdl	Invalid slice description language at offset %ld
-104	335544570	dsql_command_err	Invalid command
-104	335544579	dsql_internal_err	Internal error
-104	335544590	dsql_dup_option	Option specified more than once
-104	335544591	dsql_tran_err	Unknown transaction option
-104	335544592	dsql_invalid_array	Invalid array reference
-104	335544608	command_end_err	Unexpected end of command
-104	335544612	token_err	Token unknown
-104	335544634	dsql_token_unk_err	Token unknown- line %ld, column %ld
-104	335544709	dsql_agg_ref_err	Invalid aggregate reference
-104	335544714	invalid_array_id	Invalid blob id
-104	335544730	cse_not_supported	Client/Server Express not supported in this release
-104	335544743	token_too_long	Token size exceeds limit
-104	335544763	invalid_string_constant	A string constant is delimited by double quotes
-104	335544764	transitional_date	DATE must be changed to TIMESTAMP
-104	335544796	sql_dialect_datatype_unsupport	Client SQL dialect %d does not support reference to

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
			%s datatype
-104	335544798	depend_on_uncommitted_rel	You created an indirect dependency on uncommitted metadata. You must roll back the current transaction.
-104	335544821	dsql_column_pos_err	Invalid column position used in the %s clause
-104	335544822	dsql_agg_where_err	Cannot use an aggregate function in a WHERE clause, use HAVING instead
-104	335544823	dsql_agg_group_err	Cannot use an aggregate function in a GROUP BY clause
-104	335544824	dsql_agg_column_err	Invalid expression in the %s (not contained in either an aggregate function or the GROUP BY clause)
-104	335544825	dsql_agg_having_err	Invalid expression in the %s (neither an aggregate function nor a part of the GROUP BY clause)
-104	335544826	dsql_agg_nested_err	Nested aggregate functions are not allowed
-104	335544849	malformed_string	Malformed string
-104	335544851	command_end_err2	Unexpected end of command- line %ld, column %ld
-104	336397231	dsql_cte_wrong_clause	Recursive member of CTE '@1' has @2 clause
-104	336397232	dsql_cte_union_all	Recursive members of CTE (@1) must be linked with another members via UNION ALL
-104	336397233	dsql_cte_miss_nonrecursive	Non-recursive member is missing in CTE '@1'
-104	336397235	dsql_col_more_than_once_using	Column @1 appears more than once in USING clause
-104	336397237	dsql_cte_not_used	CTE "@1" is not used in query
-104	336397234	dsql_cte_nested_with	WITH clause can't be nested
-104	336397215	dsql_max_sort_items	Cannot sort on more than 255 items
-104	336397216	dsql_max_group_items	Cannot group on more than 255 items
-104	336397217	dsql_conflicting_sort_field	Cannot include the same field (@1.@2) twice in the ORDER BY clause with conflicting sorting options
-104	336397218	dsql_derived_table_more_columns	Column list from derived table @1 has more columns than the number of items in its SELECT statement
-104	336397219	dsql_derived_table_less_columns	Column list from derived table @1 has less columns than the number of items in its SELECT statement
-104	336397220	dsql_derived_field_unnamed	No column name specified for column number @1 in derived table @2

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-104	336397221	dsql_derived_field_dup_name	Column @1 was specified multiple times for derived table @2
-104	336397222	dsql_derived_alias_select	Internal dsql error: alias type expected by pass1_expand_select_node
-104	336397223	dsql_derived_alias_field	Internal dsql error: alias type expected by pass1_field
-104	336397224	dsql_auto_field_bad_pos	Internal dsql error: column position out of range in pass1_union_auto_cast
-104	336397225	dsql_cte_wrong_reference	Recursive CTE member (@1) can refer itself only in FROM clause
-104	336397226	dsql_cte_cycle	CTE '@1' has cyclic dependencies
-104	336397227	dsql_cte_outer_join	Recursive member of CTE can't be member of an outer join
-104	336397228	dsql_cte_mult_references	Recursive member of CTE can't reference itself more than once
-104	336397229	dsql_cte_not_a_union	Recursive CTE (@1) must be an UNION
-104	336397230	dsql_cte_nonrecurs_after_recurs	CTE '@1' defined non-recursive member after recursive
-105	335544702	like_escape_invalid	Invalid ESCAPE sequence
-105	335544789	extract_input_mismatch	Specified EXTRACT part does not exist in input datatype
-150	335544360	read_only_rel	Attempted update of read-only table
-150	335544362	read_only_view	Cannot update read-only view %s
-150	335544446	non_updatable	Not updatable
-150	335544546	constaint_on_view	Cannot define constraints on views
-151	335544359	read_only_field	Attempted update of read-only column
-155	335544658	dsql_base_table	%s is not a valid base table of the specified view
-157	335544598	specify_field_err	Must specify column name for view select expression
-158	335544599	num_field_err	Number of columns does not match select list
-162	335544685	no_dbkey	Dbkey not available for multi-table views
-170	335544512	prcmismat	Input parameter mismatch for procedure %s
-170	335544619	extern_func_err	External functions cannot have more than 10 parameters

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-170	335544850	prc_out_param_mismatch	Output parameter mismatch for procedure %s
-171	335544439	funmismat	Function %s could not be matched
-171	335544458	invalid_dimension	Column not array or invalid dimensions (expected %ld, encountered %ld)
-171	335544618	return_mode_err	Return mode by value not allowed for this data type
-171	335544873	array_max_dimensions	Array data type can use up to @1 dimensions
-172	335544438	funnotdef	Function %s is not defined
-203	335544708	dyn_fld_ambiguous	Ambiguous column reference.
-204	336003085	dsql_ambiguous_field_name	Ambiguous field name between %s and %s
-204	335544463	gennotdef	Generator %s is not defined
-204	335544502	stream_not_defined	Reference to invalid stream number
-204	335544509	charset_not_found	CHARACTER SET %s is not defined
-204	335544511	prcnotdef	Procedure %s is not defined
-204	335544515	codnotdef	Status code %s unknown
-204	335544516	xcpnotdef	Exception %s not defined
-204	335544532	ref_cnstrnt_notfound	Name of Referential Constraint not defined in constraints table.
-204	335544551	grant_obj_notfound	Could not find table/procedure for GRANT
-204	335544568	text_subtype	Implementation of text subtype %d not located.
-204	335544573	dsql_datatype_err	Data type unknown
-204	335544580	dsql_relation_err	Table unknown
-204	335544581	dsql_procedure_err	Procedure unknown
-204	335544588	collation_not_found	COLLATION %s for CHARACTER SET %s is not defined
-204	335544589	collation_not_for_charset	COLLATION %s is not valid for specified CHARACTER SET
-204	335544595	dsql_trigger_err	Trigger unknown
-204	335544620	alias_conflict_err	Alias %s conflicts with an alias in the same statement
-204	335544621	procedure_conflict_error	Alias %s conflicts with a procedure in the same statement

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-204	335544622	relation_conflict_err	Alias %s conflicts with a table in the same statement
-204	335544635	dsql_no_relation_alias	There is no alias or table named %s at this scope level
-204	335544636	indexname	There is no index %s for table %s
-204	335544640	collation_requires_text	Invalid use of CHARACTER SET or COLLATE
-204	335544662	dsql_blob_type_unknown	BLOB SUB_TYPE %s is not defined
-204	335544759	bad_default_value	Can not define a not null column with NULL as default value
-204	335544760	invalid_clause	Invalid clause--- '%s'
-204	335544800	too_many_contexts	Too many Contexts of Relation/Procedure/Views. Maximum allowed is 255
-204	335544817	bad_limit_param	Invalid parameter to FIRST.Only integers >= 0 are allowed.
-204	335544818	bad_skip_param	Invalid parameter to SKIP.Only integers >= 0 are allowed.
-204	335544837	bad_substring_offset	Invalid offset parameter %d to SUBSTRING. Only positive integers are allowed.
-204	335544853	bad_substring_length	Invalid length parameter %d to SUBSTRING. Negative integers are not allowed.
-204	335544854	charset_not_installed	CHARACTER SET %s is not installed
-204	335544855	collation_not_installed	COLLATION %s for CHARACTER SET %s is not installed
-204	335544867	subtype_for_internal_use	Blob sub_types bigger than 1 (text) are for internal use only
-205	335544396	fldnotdef	Column %s is not defined in table %s
-205	335544552	grant_fld_notfound	Could not find column for GRANT
-205	335544883	fldnotdef2	Column @1 is not defined in procedure @2
-206	335544578	dsql_field_err	Column unknown
-206	335544587	dsql_blob_err	Column is not a BLOB
-206	335544596	dsql_subselect_err	Subselect illegal in this context
-206	336397208	dsql_line_col_error	At line %d, column %d
-206	336397209	dsql_unknown_pos	At unknown line and column
-206	336397210	dsql_no_dup_name	Column %s cannot be repeated in %s statement

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-208	335544617	order_by_err	Invalid ORDER BY clause
-219	335544395	relnotdef	Table %s is not defined
-219	335544872	domnotdef	Domain @1 is not defined
-230	335544487	walw_err	WAL Writer error
-231	335544488	logh_small	Log file header of %s too small
-232	335544489	logh_inv_version	Invalid version of log file %s
-233	335544490	logh_open_flag	Log file %s not latest in the chain but open flag still set
-234	335544491	logh_open_flag2	Log file %s not closed properly; database recovery may be required
-235	335544492	logh_diff_dbname	Database name in the log file %s is different
-236	335544493	logf_unexpected_eof	Unexpected end of log file %s at offset %ld
-237	335544494	logr_incomplete	Incomplete log record at offset %ld in log file %s
-238	335544495	logr_header_small	Log record header too small at offset %ld in log file %s
-239	335544496	logb_small	Log block too small at offset %ld in log file %s
-239	335544691	cache_too_small	Insufficient memory to allocate page buffer cache
-239	335544693	log_too_small	Log size too small
-239	335544694	partition_too_small	Log partition size too small
-243	335544500	no_wal	Database does not use Write-ahead Log
-257	335544566	start_cm_for_wal	WAL defined; Cache Manager must be started first
-260	335544690	cache_redef	Cache redefined
-260	335544692	log_redef	Log redefined
-261	335544695	partition_not_supp	Partitions not supported in series of log file specification
-261	335544696	log_length_spec	Total length of a partitioned log must be specified
-281	335544637	no_stream_plan	Table %s is not referenced in plan
-282	335544638	stream_twice	Table %s is referenced more than once in plan; use aliases to distinguish
-282	335544643	dsql_self_join	The table %s is referenced twice; use aliases to differentiate

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-282	335544659	duplicate_base_table	Table %s is referenced twice in view; use an alias to distinguish
-282	335544660	view_alias	View %s has more than one base table; use aliases to distinguish
-282	335544710	complex_view	Navigational stream %ld references a view with more than one base table
-283	335544639	stream_not_found	Table %s is referenced in the plan but not the from list
-284	335544642	index_unused	Index %s cannot be used in the specified plan
-291	335544531	primary_key_notnull	Column used in a PRIMARY constraint must be NOT NULL.
-292	335544534	ref_cnstrnt_update	Cannot update constraints (RDB$REF_CONSTRAINTS).
-293	335544535	check_cnstrnt_update	Cannot update constraints (RDB$CHECK_CONSTRAINTS).
-294	335544536	check_cnstrnt_del	Cannot delete CHECK constraint entry (RDB$CHECK_CONSTRAINTS)
-295	335544545	rel_cnstrnt_update	Cannot update constraints (RDB$RELATION_CONSTRAINTS).
-296	335544547	invld_cnstrnt_type	Internal gds software consistency check (invalid RDB$CONSTRAINT_TYPE)
-297	335544558	check_constraint	Operation violates CHECK constraint %s on view or table %s
-313	335544669	dsql_count_mismatch	Count of column list and variable list do not match
-313	336003100	upd_ins_doesnt_match_matching	UPDATE OR INSERT field list does not match MATCHING clause
-313	336003099	upd_ins_doesnt_match_pk	UPDATE OR INSERT field list does not match primary key of table @1
-314	335544565	transliteration_failed	Cannot transliterate character between character sets
-315	336068815	dyn_dtype_invalid	Cannot change datatype for column %s.Changing datatype is not supported for BLOB or ARRAY columns.
-383	336068814	dyn_dependency_exists	Column %s from table %s is referenced in %s
-401	335544647	invalid_operator	Invalid comparison operator for find operation
-402	335544368	segstr_no_op	Attempted invalid operation on a BLOB

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-402	335544414	blobnotsup	BLOB and array data types are not supported for %s operation
-402	335544427	datnotsup	Data operation not supported
-406	335544457	out_of_bounds	Subscript out of bounds
-407	335544435	nullsegkey	Null segment of UNIQUE KEY
-413	335544334	convert_error	Conversion error from string "%s"
-413	335544454	nofilter	Filter not found to convert type %ld to type %ld
-413	335544860	blob_convert_error	Unsupported conversion to target type BLOB (subtype %d)
-413	335544861	array_convert_error	Unsupported conversion to target type ARRAY
-501	335544327	bad_req_handle	Invalid request handle
-501	335544577	dsql_cursor_close_err	Attempt to reclose a closed cursor
-502	336003090	dsql_cursor_redefined	Statement already has a cursor %s assigned
-502	336003091	dsql_cursor_not_found	Cursor %s is not found in the current context
-502	336003092	dsql_cursor_exists	Cursor %s already exists in the current context
-502	336003093	dsql_cursor_rel_ambiguous	Relation %s is ambiguous in cursor %s
-502	336003094	dsql_cursor_rel_not_found	Relation %s is not found in cursor %s
-502	336003095	dsql_cursor_not_open	Cursor is not open
-502	335544574	dsql_decl_err	Invalid cursor declaration
-502	335544576	dsql_cursor_open_err	Attempt to reopen an open cursor
-504	336003089	dsql_cursor_invalid	Empty cursor name is not allowed
-504	335544572	dsql_cursor_err	Invalid cursor reference
-508	335544348	no_cur_rec	No current record for fetch operation
-510	335544575	dsql_cursor_update_err	Cursor %s is not updatable
-518	335544582	dsql_request_err	Request unknown
-519	335544688	dsql_open_cursor_request	The prepare statement identifies a prepare statement with an open cursor
-530	335544466	foreign_key	Violation of FOREIGN KEY constraint "%s" on table "%s"
-530	335544838	foreign_key_target_doesnt_exist	Foreign key reference target does not exist

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-530	335544839	foreign_key_references_present	Foreign key references are present for the record
-531	335544597	dsql_crdb_prepare_err	Cannot prepare a CREATE DATABASE/SCHEMA statement
-532	335544469	trans_invalid	Transaction marked invalid by I/O error
-551	335544352	no_priv	No permission for %s access to %s %s
-551	335544790	insufficient_svc_privileges	Service %s requires SYSDBA permissions.Reattach to the Service Manager using the SYSDBA account.
-552	335544550	not_rel_owner	Only the owner of a table may reassign ownership
-552	335544553	grant_nopriv	User does not have GRANT privileges for operation
-552	335544707	grant_nopriv_on_base	User does not have GRANT privileges on base table/view for operation
-553	335544529	existing_priv_mod	Cannot modify an existing user privilege
-595	335544645	stream_crack	The current position is on a crack
-596	335544644	stream_bof	Illegal operation when at beginning of stream
-597	335544632	dsql_file_length_err	Preceding file did not specify length, so %s must include starting page number
-598	335544633	dsql_shadow_number_err	Shadow number must be a positive integer
-599	335544607	node_err	Gen.c: node not supported
-599	335544625	node_name_err	A node name is not permitted in a secondary, shadow, cache or log file name
-600	335544680	crrp_data_err	Sort error: corruption in data structure
-601	335544646	db_or_file_exists	Database or file exists
-604	335544593	dsql_max_arr_dim_exceeded	Array declared with too many dimensions
-604	335544594	dsql_arr_range_error	Illegal array dimension range
-605	335544682	dsql_field_ref	Inappropriate self-reference of column
-607	336003074	dsql_dbkey_from_non_table	Cannot SELECT RDB$DB_KEY from a stored procedure.
-607	336003086	dsql_udf_return_pos_err	External function should have return position between 1 and %d
-607	336003096	dsql_type_not_supp_ext_tab	Data type %s is not supported for EXTERNAL TABLES. Relation '%s', field '%s'
-607	335544351	no_meta_update	Unsuccessful metadata update

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-607	335544549	systrig_update	Cannot modify or erase a system trigger
-607	335544657	dsql_no_blob_array	Array/BLOB/DATE data types not allowed in arithmetic
-607	335544746	reftable_requires_pk	"REFERENCES table" without "(column)" requires PRIMARY KEY on referenced table
-607	335544815	generator_name	GENERATOR %s
-607	335544816	udf_name	UDF %s
-607	335544858	must_have_phys_field	Can't have relation with only computed fields or constraints
-607	336397206	dsql_table_not_found	Table %s does not exist
-607	336397207	dsql_view_not_found	View %s does not exist
-607	336397212	dsql_no_array_computed	Array and BLOB data types not allowed in computed field
-607	336397214	dsql_only_can_subscript_array	Scalar operator used on field @1 which is not an array
-612	336068812	dyn_domain_name_exists	Cannot rename domain %s to %s.A domain with that name already exists.
-612	336068813	dyn_field_name_exists	Cannot rename column %s to %s.A column with that name already exists in table %s.
-615	335544475	relation_lock	Lock on table %s conflicts with existing lock
-615	335544476	record_lock	Requested record lock conflicts with existing lock
-615	335544507	range_in_use	Refresh range number %ld already in use
-616	335544530	primary_key_ref	Cannot delete PRIMARY KEY being used in FOREIGN KEY definition.
-616	335544539	integ_index_del	Cannot delete index used by an Integrity Constraint
-616	335544540	integ_index_mod	Cannot modify index used by an Integrity Constraint
-616	335544541	check_trig_del	Cannot delete trigger used by a CHECK Constraint
-616	335544543	cnstrnt_fld_del	Cannot delete column being used in an Integrity Constraint.
-616	335544630	dependency	There are %ld dependencies
-616	335544674	del_last_field	Last column in a table cannot be deleted
-616	335544728	integ_index_deactivate	Cannot deactivate index used by an integrity constraint

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-616	335544729	integ_deactivate_primary	Cannot deactivate index used by a PRIMARY/UNIQUE constraint
-617	335544542	check_trig_update	Cannot update trigger used by a CHECK Constraint
-617	335544544	cnstrnt_fld_rename	Cannot rename column being used in an Integrity Constraint.
-618	335544537	integ_index_seg_del	Cannot delete index segment used by an Integrity Constraint
-618	335544538	integ_index_seg_mod	Cannot update index segment used by an Integrity Constraint
-625	335544347	not_valid	Validation error for column %s, value "%s"
-625	335544879	not_valid_for_var	Validation error for variable @1, value "@2"
-625	335544880	not_valid_for	Validation error for @1, value "@2"
-637	335544664	dsql_duplicate_spec	Duplicate specification of %s- not supported
-637	336397213	dsql_implicit_domain_name	Implicit domain name @1 not allowed in user created domain
-660	335544533	foreign_key_notfound	Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY.
-660	335544628	idx_create_err	Cannot create index %s
-660	336003098	primary_key_required	Primary key required on table @1
-663	335544624	idx_seg_err	Segment count of 0 defined for index %s
-663	335544631	idx_key_err	Too many keys defined for index %s
-663	335544672	key_field_err	Too few key columns found for index %s (incorrect column name?)
-664	335544434	keytoobig	Key size exceeds implementation restriction for index "%s"
-677	335544445	ext_err	%s extension error
-685	335544465	bad_segstr_type	Invalid BLOB type for operation
-685	335544670	blob_idx_err	Attempt to index BLOB column in index %s
-685	335544671	array_idx_err	Attempt to index array column in index %s
-689	335544403	badpagtyp	Page %ld is of wrong type (expected %ld, found %ld)
-689	335544650	page_type_err	Wrong page type
-690	335544679	no_segments_err	Segments not allowed in expression index %s

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-691	335544681	rec_size_err	New record size of %ld bytes is too big
-692	335544477	max_idx	Maximum indexes per table (%d) exceeded
-693	335544663	req_max_clones_exceeded	Too many concurrent executions of the same request
-694	335544684	no_field_access	Cannot access column %s in view %s
-802	335544321	arith_except	Arithmetic exception, numeric overflow, or string truncation
-802	335544836	concat_overflow	Concatenation overflow. Resulting string cannot exceed 32K in length.
-803	335544349	no_dup	Attempt to store duplicate value (visible to active transactions) in unique index "%s"
-803	335544665	unique_key_violation	Violation of PRIMARY or UNIQUE KEY constraint "%s" on table "%s"
-804	335544380	wronumarg	Wrong number of arguments on call
-804	335544583	dsql_sqlda_err	SQLDA missing or incorrect version, or incorrect number/type of variables
-804	335544586	dsql_function_err	Function unknown
-804	335544713	dsql_sqlda_value_err	Incorrect values within SQLDA structure
-804	336397205	dsql_too_old_ods	ODS versions before ODS%d are not supported
-804	336003097	dsql_feature_not_supported_ods	Feature not supported on ODS version older than @1.@2
-804	335544584	dsql_var_count_err	Count of read-write columns does not equal count of values
-806	335544600	col_name_err	Only simple column names permitted for VIEW WITH CHECK OPTION
-807	335544601	where_err	No WHERE clause for VIEW WITH CHECK OPTION
-808	335544602	table_view_err	Only one table allowed for VIEW WITH CHECK OPTION
-809	335544603	distinct_err	DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION
-810	335544605	subquery_err	No subqueries permitted for VIEW WITH CHECK OPTION
-811	335544652	sing_select_err	Multiple rows in singleton select

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-816	335544651	ext_readonly_err	Cannot insert because the file is readonly or is on a read only medium.
-816	335544715	extfile_uns_op	Operation not supported for EXTERNAL FILE table %s
-817	336003079	isc_sql_dialect_conflict_num	DB dialect %d and client dialect %d conflict with respect to numeric precision %d.
-817	335544361	read_only_trans	Attempted update during read-only transaction
-817	335544371	segstr_no_write	Attempted write to read-only BLOB
-817	335544444	read_only	Operation not supported
-817	335544765	read_only_database	Attempted update on read-only database
-817	335544766	must_be_dialect_2_and_up	SQL dialect %s is not supported in this database
-817	335544793	ddl_not_allowed_by_db_sql_dial	Metadata update statement is not allowed by the current database SQL dialect %d
-817	336003101	upd_ins_with_complex_view	UPDATE OR INSERT without MATCHING could not be used with views based on more than one table
-817	336003102	dsql_incompatible_trigger_type	Incompatible trigger type
-817	336003103	dsql_db_trigger_type_cant_change	Database trigger type can't be changed
-820	335544356	obsolete_metadata	Metadata is obsolete
-820	335544379	wrong_ods	Unsupported on-disk structure for file %s; found %ld.%ld, support %ld.%ld
-820	335544437	wrodynver	Wrong DYN version
-820	335544467	high_minor	Minor version too high found %ld expected %ld
-820	335544881	need_difference	Difference file name should be set explicitly for database on raw device
-823	335544473	invalid_bookmark	Invalid bookmark handle
-824	335544474	bad_lock_level	Invalid lock level %d
-825	335544519	bad_lock_handle	Invalid lock handle
-826	335544585	dsql_stmt_handle	Invalid statement handle
-827	335544655	invalid_direction	Invalid direction for find operation
-827	335544718	invalid_key	Invalid key for find operation
-828	335544678	inval_key_posn	Invalid key position

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-829	336068816	dyn_char_fld_too_small	New size specified for column %s must be at least %d characters.
-829	336068817	dyn_invalid_dtype_conversion	Cannot change datatype for %s.Conversion from base type %s to %s is not supported.
-829	336068818	dyn_dtype_conv_invalid	Cannot change datatype for column %s from a character type to a non-character type.
-829	335544616	field_ref_err	Invalid column reference
-829	336068829	max_coll_per_charset	Maximum number of collations per character set exceeded
-829	336068830	invalid_coll_attr	Invalid collation attributes
-829	336068852	dyn_scale_too_big	New scale specified for column @1 must be at most @2.
-829	336068853	dyn_precision_too_small	New precision specified for column @1 must be at least @2.
-830	335544615	field_aggregate_err	Column used with aggregate
-831	335544548	primary_key_exists	Attempt to define a second PRIMARY KEY for the same table
-832	335544604	key_field_count_err	FOREIGN KEY column count does not match PRIMARY KEY
-833	335544606	expression_eval_err	Expression evaluation not supported
-833	335544810	date_range_exceeded	Value exceeds the range for valid dates
-834	335544508	range_not_found	Refresh range number %ld not found
-835	335544649	bad_checksum	Bad checksum
-836	335544517	except	Exception %d
-836	335544848	except2	Exception %s
-837	335544518	cache_restart	Restart shared cache manager
-838	335544560	shutwarn	Database %s shutdown in %d seconds
-841	335544677	version_err	Too many versions
-842	335544697	precision_err	Precision must be from 1 to 18
-842	335544698	scale_nogt	Scale must be between zero and precision
-842	335544699	expec_short	Short integer expected
-842	335544700	expec_long	Long integer expected

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-842	335544701	expec_ushort	Unsigned short integer expected
-842	335544712	expec_positive	Positive value expected
-901	336330753	gbak_unknown_switch	Found unknown switch
-901	336920577	gstat_unknown_switch	Found unknown switch
-901	335740929	gfix_db_name	Data base file name (%s) already given
-901	335544322	bad_dbkey	Invalid database key
-901	336920578	gstat_retry	Please retry, giving a database name
-901	336330754	gbak_page_size_missing	Page size parameter missing
-901	335740930	gfix_invalid_sw	Invalid switch %s
-901	336920579	gstat_wrong_ods	Wrong ODS version, expected %d, encountered %d
-901	336330755	gbak_page_size_toobig	Page size specified (%ld) greater than limit (16384 bytes)
-901	336920580	gstat_unexpected_eof	Unexpected end of database file.
-901	336330756	gbak_redir_ouput_missing	Redirect location for output is not specified
-901	335740932	gfix_incmp_sw	Incompatible switch combination
-901	336330757	gbak_switches_conflict	Conflicting switches for backup/restore
-901	335740933	gfix_replay_req	Replay log pathname required
-901	335544326	bad_dpb_form	Unrecognized database parameter block
-901	336330758	gbak_unknown_device	Device type %s not known
-901	335740934	gfix_pgbuf_req	Number of page buffers for cache required
-901	336330759	gbak_no_protection	Protection is not there yet
-901	335740935	gfix_val_req	Numeric value required
-901	335544328	bad_segstr_handle	Invalid BLOB handle
-901	336330760	gbak_page_size_not_allowed	Page size is allowed only on restore or create
-901	335740936	gfix_pval_req	Positive numeric value required
-901	335544329	bad_segstr_id	Invalid BLOB ID
-901	336330761	gbak_multi_source_dest	Multiple sources or destinations specified
-901	335740937	gfix_trn_req	Number of transactions per sweep required
-901	335544330	bad_tpb_content	Invalid parameter in transaction parameter block

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	336330762	gbak_filename_missing	Requires both input and output filenames
-901	335544331	bad_tpb_form	Invalid format for transaction parameter block
-901	336330763	gbak_dup_inout_names	Input and output have the same name.Disallowed.
-901	335544332	bad_trans_handle	Invalid transaction handle (expecting explicit transaction start)
-901	336330764	gbak_inv_page_size	Expected page size, encountered "%s"
-901	335740940	gfix_full_req	"full" or "reserve" required
-901	336330765	gbak_db_specified	REPLACE specified, but the first file %s is a database
-901	335740941	gfix_usrname_req	User name required
-901	336330766	gbak_db_exists	Database %s already exists.To replace it, use the-REP switch
-901	335740942	gfix_pass_req	Password required
-901	336330767	gbak_unk_device	Device type not specified
-901	336723983	gsec_cant_open_db	Unable to open database
-901	335740943	gfix_subs_name	Subsystem name
-901	336723984	gsec_switches_error	Error in switch specifications
-901	335544337	excess_trans	Attempt to start more than %ld transactions
-901	335740945	gfix_sec_req	Number of seconds required
-901	336723985	gsec_no_op_spec	No operation specified
-901	335740946	gfix_nval_req	Numeric value between 0 and 32767 inclusive required
-901	336723986	gsec_no_usr_name	No user name specified
-901	335544339	infinap	Information type inappropriate for object specified
-901	336723987	gsec_err_add	Add record error
-901	335740947	gfix_type_shut	Must specify type of shutdown
-901	335544340	infona	No information of this type available for object specified
-901	336330772	gbak_blob_info_failed	Gds_$blob_info failed
-901	335740948	gfix_retry	Please retry, specifying an option
-901	336723988	gsec_err_modify	Modify record error

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335544341	infunk	Unknown information item
-901	336330773	gbak_unk_blob_item	Do not understand BLOB INFO item %ld
-901	336723989	gsec_err_find_mod	Find/modify record error
-901	335544342	integ_fail	Action cancelled by trigger (%ld) to preserve data integrity
-901	336330774	gbak_get_seg_failed	Gds_$get_segment failed
-901	336723990	gsec_err_rec_not_found	Record not found for user: %s
-901	336330775	gbak_close_blob_failed	Gds_$close_blob failed
-901	335740951	gfix_retry_db	Please retry, giving a database name
-901	336723991	gsec_err_delete	Delete record error
-901	336330776	gbak_open_blob_failed	Gds_$open_blob failed
-901	336723992	gsec_err_find_del	Find/delete record error
-901	335544345	lock_conflict	Lock conflict on no wait transaction
-901	336330777	gbak_put_blr_gen_id_failed	Failed in put_blr_gen_id
-901	336330778	gbak_unk_type	Data type %ld not understood
-901	336330779	gbak_comp_req_failed	Gds_$compile_request failed
-901	336330780	gbak_start_req_failed	Gds_$start_request failed
-901	336723996	gsec_err_find_disp	Find/display record error
-901	336920605	gstat_open_err	Can't open database file %s
-901	336330781	gbak_rec_failed	gds_$receive failed
-901	336723997	gsec_inv_param	Invalid parameter, no switch defined
-901	335544350	no_finish	Program attempted to exit without finishing database
-901	336920606	gstat_read_err	Can't read a database page
-901	336330782	gbak_rel_req_failed	Gds_$release_request failed
-901	336723998	gsec_op_specified	Operation already specified
-901	336920607	gstat_sysmemex	System memory exhausted
-901	336330783	gbak_db_info_failed	gds_$database_info failed
-901	336723999	gsec_pw_specified	Password already specified
-901	336330784	gbak_no_db_desc	Expected database description record

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	336724000	gsec_uid_specified	Uid already specified
-901	335544353	no_recon	Transaction is not in limbo
-901	336330785	gbak_db_create_failed	Failed to create database %s
-901	336724001	gsec_gid_specified	Gid already specified
-901	336330786	gbak_decomp_len_error	RESTORE: decompression length error
-901	336724002	gsec_proj_specified	Project already specified
-901	335544355	no_segstr_close	BLOB was not closed
-901	336330787	gbak_tbl_missing	Cannot find table %s
-901	336724003	gsec_org_specified	Organization already specified
-901	336330788	gbak_blob_col_missing	Cannot find column for BLOB
-901	336724004	gsec_fname_specified	First name already specified
-901	335544357	open_trans	Cannot disconnect database with open transactions (%ld active)
-901	336330789	gbak_create_blob_failed	Gds_$create_blob failed
-901	336724005	gsec_mname_specified	Middle name already specified
-901	335544358	port_len	Message length error (encountered %ld, expected %ld)
-901	336330790	gbak_put_seg_failed	Gds_$put_segment failed
-901	336724006	gsec_lname_specified	Last name already specified
-901	336330791	gbak_rec_len_exp	Expected record length
-901	336330792	gbak_inv_rec_len	Wrong length record, expected %ld encountered %ld
-901	336724008	gsec_inv_switch	Invalid switch specified
-901	336330793	gbak_exp_data_type	Expected data attribute
-901	336724009	gsec_amb_switch	Ambiguous switch specified
-901	336330794	gbak_gen_id_failed	Failed in store_blr_gen_id
-901	336724010	gsec_no_op_specified	No operation specified for parameters
-901	335544363	req_no_trans	No transaction for request
-901	336330795	gbak_unk_rec_type	Do not recognize record type %ld
-901	336724011	gsec_params_not_allowed	No parameters allowed for this operation

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335544364	req_sync	Request synchronization error
-901	336330796	gbak_inv_bkup_ver	Expected backup version 1, 2, 3, 4, 5, 6, or 7.Found %ld
-901	336724012	gsec_incompat_switch	Incompatible switches specified
-901	335544365	req_wrong_db	Request referenced an unavailable database
-901	336330797	gbak_missing_bkup_desc	Expected backup description record
-901	336330798	gbak_string_trunc	String truncated
-901	336330799	gbak_cant_rest_record	warning-- record could not be restored
-901	336330800	gbak_send_failed	Gds_$send failed
-901	335544369	segstr_no_read	Attempted read of a new, open BLOB
-901	336330801	gbak_no_tbl_name	No table name for data
-901	336330802	gbak_unexp_eof	Unexpected end of file on backup file
-901	335544370	segstr_no_trans	Attempted action on blob outside transaction
-901	336330803	gbak_db_format_too_old	Database format %ld is too old to restore to
-901	335544372	segstr_wrong_db	Attempted reference to BLOB in unavailable database
-901	336330804	gbak_inv_array_dim	Array dimension for column %s is invalid
-901	336330807	gbak_xdr_len_expected	Expected XDR record length
-901	335544376	unres_rel	Table %s was omitted from the transaction reserving list
-901	335544377	uns_ext	Request includes a DSRI extension not supported in this implementation
-901	335544378	wish_list	Feature is not supported
-901	335544382	random	%s
-901	335544383	fatal_conflict	Unrecoverable conflict with limbo transaction %ld
-901	335740991	gfix_exceed_max	Internal block exceeds maximum size
-901	335740992	gfix_corrupt_pool	Corrupt pool
-901	336330817	gbak_open_bkup_error	Cannot open backup file %s
-901	335740993	gfix_mem_exhausted	Virtual memory exhausted
-901	336330818	gbak_open_error	Cannot open status and error output file %s

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335740994	gfix_bad_pool	Bad pool id
-901	335740995	gfix_trn_not_valid	Transaction state %d not in valid range.
-901	335544392	bdbincon	Internal error
-901	336724044	gsec_inv_username	Invalid user name (maximum 31 bytes allowed)
-901	336724045	gsec_inv_pw_length	Warning- maximum 8 significant bytes of password used
-901	336724046	gsec_db_specified	Database already specified
-901	336724047	gsec_db_admin_specified	Database administrator name already specified
-901	336724048	gsec_db_admin_pw_specified	Database administrator password already specified
-901	336724049	gsec_sql_role_specified	SQL role name already specified
-901	335741012	gfix_unexp_eoi	Unexpected end of input
-901	335544407	dbbnotzer	Database handle not zero
-901	335544408	tranotzer	Transaction handle not zero
-901	335741018	gfix_recon_fail	Failed to reconnect to a transaction in database %s
-901	335544418	trainlim	Transaction in limbo
-901	335544419	notinlim	Transaction not in limbo
-901	335544420	traoutsta	Transaction outstanding
-901	335544428	badmsgnum	Undefined message number
-901	335741036	gfix_trn_unknown	Transaction description item unknown
-901	335741038	gfix_mode_req	"read_only" or "read_write" required
-901	335544431	blocking_signal	Blocking signal has been received
-901	335741042	gfix_pzval_req	Positive or zero numeric value required
-901	335544442	noargacc_read	Database system cannot read argument %ld
-901	335544443	noargacc_write	Database system cannot write argument %ld
-901	335544450	misc_interpreted	%s
-901	335544468	tra_state	Transaction %ld is %s
-901	335544485	bad_stmt_handle	Invalid statement handle
-901	336330934	gbak_missing_block_fac	Blocking factor parameter missing
-901	336330935	gbak_inv_block_fac	Expected blocking factor, encountered "%s"

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	336330936	gbak_block_fac_specified	A blocking factor may not be used in conjunction with device CT
-901	336330940	gbak_missing_username	User name parameter missing
-901	336068796	dyn_role_does_not_exist	SQL role %s does not exist
-901	336330941	gbak_missing_password	Password parameter missing
-901	336068797	dyn_no_grant_admin_opt	User %s has no grant admin option on SQL role %s
-901	335544510	lock_timeout	Lock time-out on wait transaction
-901	336068798	dyn_user_not_role_member	User %s is not a member of SQL role %s
-901	336068799	dyn_delete_role_failed	%s is not the owner of SQL role %s
-901	336068800	dyn_grant_role_to_user	%s is a SQL role and not a user
-901	336068801	dyn_inv_sql_role_name	User name %s could not be used for SQL role
-901	336068802	dyn_dup_sql_role	SQL role %s already exists
-901	336068803	dyn_kywd_spec_for_role	Keyword %s can not be used as a SQL role name
-901	336068804	dyn_roles_not_supported	SQL roles are not supported in on older versions of the database.A backup and restore of the database is required.
-901	336330952	gbak_missing_skipped_bytes	missing parameter for the number of bytes to be skipped
-901	336330953	gbak_inv_skipped_bytes	Expected number of bytes to be skipped, encountered "%s"
-901	336068820	dyn_zero_len_id	Zero length identifiers are not allowed
-901	336330965	gbak_err_restore_charset	Bad attribute for RDB$CHARACTER_SETS
-901	336330967	gbak_err_restore_collation	Bad attribute for RDB$COLLATIONS
-901	336330972	gbak_read_error	Unexpected I/O error while reading from backup file
-901	336330973	gbak_write_error	Unexpected I/O error while writing to backup file
-901	336330985	gbak_db_in_use	Could not drop database %s (database might be in use)
-901	336330990	gbak_sysmemex	System memory exhausted
-901	335544559	bad_svc_handle	Invalid service handle
-901	335544561	wrospbver	Wrong version of service parameter block
-901	335544562	bad_spb_form	Unrecognized service parameter block

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335544563	svcnotdef	Service %s is not defined
-901	336331002	gbak_restore_role_failed	Bad attributes for restoring SQL role
-901	336331005	gbak_role_op_missing	SQL role parameter missing
-901	336331010	gbak_page_buffers_missing	Page buffers parameter missing
-901	336331011	gbak_page_buffers_wrong_param	Expected page buffers, encountered "%s"
-901	336331012	gbak_page_buffers_restore	Page buffers is allowed only on restore or create
-901	336331014	gbak_inv_size	Size specification either missing or incorrect for file %s
-901	336331015	gbak_file_outof_sequence	File %s out of sequence
-901	336331016	gbak_join_file_missing	Can't join-- one of the files missing
-901	336331017	gbak_stdin_not_supptd	standard input is not supported when using join operation
-901	336331018	gbak_stdout_not_supptd	Standard output is not supported when using split operation
-901	336331019	gbak_bkup_corrupt	Backup file %s might be corrupt
-901	336331020	gbak_unk_db_file_spec	Database file specification missing
-901	336331021	gbak_hdr_write_failed	Can't write a header record to file %s
-901	336331022	gbak_disk_space_ex	Free disk space exhausted
-901	336331023	gbak_size_lt_min	File size given (%d) is less than minimum allowed (%d)
-901	336331025	gbak_svc_name_missing	Service name parameter missing
-901	336331026	gbak_not_ownr	Cannot restore over current database, must be SYSDBA or owner of the existing database.
-901	336331031	gbak_mode_req	"read_only" or "read_write" required
-901	336331033	gbak_just_data	Just data ignore all constraints etc.
-901	336331034	gbak_data_only	Restoring data only ignoring foreign key, unique, not null & other constraints
-901	335544609	index_name	INDEX %s
-901	335544610	exception_name	EXCEPTION %s
-901	335544611	field_name	COLUMN %s
-901	335544613	union_err	Union not supported

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335544614	dsql_construct_err	Unsupported DSQL construct
-901	335544623	dsql_domain_err	Illegal use of keyword VALUE
-901	335544626	table_name	TABLE %s
-901	335544627	proc_name	PROCEDURE %s
-901	335544641	dsql_domain_not_found	Specified domain or source column %s does not exist
-901	335544656	dsql_var_conflict	Variable %s conflicts with parameter in same procedure
-901	335544666	srvr_version_too_old	Server version too old to support all CREATE DATABASE options
-901	335544673	no_delete	Cannot delete
-901	335544675	sort_err	Sort error
-901	335544703	svcnoexe	Service %s does not have an associated executable
-901	335544704	net_lookup_err	Failed to locate host machine.
-901	335544705	service_unknown	Undefined service %s/%s.
-901	335544706	host_unknown	The specified name was not found in the hosts file or Domain Name Services.
-901	335544711	unprepared_stmt	Attempt to execute an unprepared dynamic SQL statement.
-901	335544716	svc_in_use	Service is currently busy: %s
-901	335544731	tra_must_sweep	
-901	335544740	udf_exception	A fatal exception occurred during the execution of a user defined function.
-901	335544741	lost_db_connection	Connection lost to database
-901	335544742	no_write_user_priv	User cannot write to RDB$USER_PRIVILEGES
-901	335544767	blob_filter_exception	A fatal exception occurred during the execution of a blob filter.
-901	335544768	exception_access_violation	Access violation.The code attempted to access a virtual address without privilege to do so.
-901	335544769	exception_datatype_missalignment	Datatype misalignment.The attempted to read or write a value that was not stored on a memory boundary.
-901	335544770	exception_array_bounds_exceeded	Array bounds exceeded.The code attempted to access an array element that is out of bounds.

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335544771	exception_float_denormal_operand	Float denormal operand.One of the floating-point operands is too small to represent a standard float value.
-901	335544772	exception_float_divide_by_zero	Floating-point divide by zero.The code attempted to divide a floating-point value by zero.
-901	335544773	exception_float_inexact_result	Floating-point inexact result.The result of a floating-point operation cannot be represented as a deciaml fraction.
-901	335544774	exception_float_invalid_operand	Floating-point invalid operand.An indeterminant error occurred during a floating-point operation.
-901	335544775	exception_float_overflow	Floating-point overflow.The exponent of a floating-point operation is greater than the magnitude allowed.
-901	335544776	exception_float_stack_check	Floating-point stack check.The stack overflowed or underflowed as the result of a floating-point operation.
-901	335544777	exception_float_underflow	Floating-point underflow.The exponent of a floating-point operation is less than the magnitude allowed.
-901	335544778	exception_integer_divide_by_zero	Integer divide by zero.The code attempted to divide an integer value by an integer divisor of zero.
-901	335544779	exception_integer_overflow	Integer overflow.The result of an integer operation caused the most significant bit of the result to carry.
-901	335544780	exception_unknown	An exception occurred that does not have a description.Exception number %X.
-901	335544781	exception_stack_overflow	Stack overflow.The resource requirements of the runtime stack have exceeded the memory available to it.
-901	335544782	exception_sigsegv	Segmentation Fault. The code attempted to access memory without priviledges.
-901	335544783	exception_sigill	Illegal Instruction. The Code attempted to perfrom an illegal operation.
-901	335544784	exception_sigbus	Bus Error. The Code caused a system bus error.
-901	335544785	exception_sigfpe	Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception.
-901	335544786	ext_file_delete	Cannot delete rows from external files.
-901	335544787	ext_file_modify	Cannot update rows in external files.

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335544788	adm_task_denied	Unable to perform operation.You must be either SYSDBA or owner of the database
-901	335544794	cancelled	Operation was cancelled
-901	335544797	svcnouser	User name and password are required while attaching to the services manager
-901	335544801	datype_notsup	Data type not supported for arithmetic
-901	335544803	dialect_not_changed	Database dialect not changed.
-901	335544804	database_create_failed	Unable to create database %s
-901	335544805	inv_dialect_specified	Database dialect %d is not a valid dialect.
-901	335544806	valid_db_dialects	Valid database dialects are %s.
-901	335544811	inv_client_dialect_specified	Passed client dialect %d is not a valid dialect.
-901	335544812	valid_client_dialects	Valid client dialects are %s.
-901	335544814	service_not_supported	Services functionality will be supported in a later versionof the product
-901	335544820	invalid_savepoint	Unable to find savepoint with name %s in transaction context
-901	335544835	bad_shutdown_mode	Target shutdown mode is invalid for database "%s"
-901	335544840	no_update	Cannot update
-901	335544842	stack_trace	%s
-901	335544843	ctx_var_not_found	Context variable %s is not found in namespace %s
-901	335544844	ctx_namespace_invalid	Invalid namespace name %s passed to %s
-901	335544845	ctx_too_big	Too many context variables
-901	335544846	ctx_bad_argument	Invalid argument passed to %s
-901	335544847	identifier_too_long	BLR syntax error. Identifier %s... is too long
-901	335544859	invalid_time_precision	Time precision exceeds allowed range (0-%d)
-901	336397211	dsql_too_many_values	Too many values (more than %d) in member list to match against
-901	335544866	met_wrong_gtt_scope	@1 cannot depend on @2335544870|collation_name|COLLATION @1
-901	335544868	illegal_prc_type	Procedure @1 is not selectable (it does not contain a SUSPEND statement)

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	335544869	invalid_sort_datatype	Datatype @1 is not supported for sorting operation
-901	335544871	domain_name	DOMAIN @1
-901	335544874	max_db_per_trans_allowed	A multi database transaction cannot span more than @1 databases
-901	335544876	bad_proc_BLR	Error while parsing procedure @1's BLR
-901	335544877	key_too_big	Index key too big
-901	336397236	dsql_unsupp_feature_dialect	Feature is not supported in dialect @1
-901	336330786	gbak_decomp_len_error	RESTORE: decompression length error
-901	336330792	gbak_inv_rec_len	Wrong length record, expected @1 encountered @2
-901	336330817	gbak_open_bkup_error	Cannot open backup file @1
-901	336330818	gbak_open_error	Cannot open status and error output file @1
-901	336068796	dyn_role_does_not_exist	SQL role @1 does not exist
-901	336330965	gbak_err_restore_charset	Character set
-901	336330967	gbak_err_restore_collation	Collation
-901	336068840	dyn_wrong_gtt_scope	@1 cannot reference @2
-901	336068856	dyn_ods_not_supp_feature	Feature '@1' is not supported in ODS @2.@3
-901	336331002	gbak_restore_role_failed	SQL role
-901	336330759	gbak_no_protection	Protection is not there yet
-901	336330760	gbak_page_size_not_allowed	Page size is allowed only on restore or create
-901	336330761	gbak_multi_source_dest	Multiple sources or destinations specified
-901	335740940	gfix_full_req	"full" or "reserve" required
-901	336330765	gbak_db_specified	REPLACE specified, but the first file @1 is a database
-901	336330766	gbak_db_exists	Database @1 already exists.To replace it, use the-REP switch
-901	335740943	gfix_subs_name	Subsystem name
-901	336330767	gbak_unk_device	Device type not specified
-901	335544337	excess_trans	Attempt to start more than @1 transactions
-901	335740947	gfix_type_shut	Must specify type of shutdown

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-901	336723988	gsec_err_modify	Modify record error
-901	336330773	gbak_unk_blob_item	Do not understand BLOB INFO item @1
-901	336723991	gsec_err_delete	Delete record error
-901	336920605	gstat_open_err	Can't open database file @1
-901	336330784	gbak_no_db_desc	Expected database description record
-901	336330785	gbak_db_create_failed	Failed to create database @1
-901	335740929	gfix_db_name	Data base file name (@1) already given
-901	336986113	fbsvcmgr_bad_am	Wrong value for access mode
-901	335740930	gfix_invalid_sw	Invalid switch @1
-901	336986114	fbsvcmgr_bad_wm	Wrong value for write mode
-901	336920578	gstat_retry	Please retry, giving a database name
-901	336986115	fbsvcmgr_bad_rs	Wrong value for reserve space
-901	335740932	gfix_incmp_sw	Incompatible switch combination
-901	336986116	fbsvcmgr_info_err	Unknown tag (@1) in info_svr_db_info block after isc_svc_query()
-901	335740933	gfix_replay_req	Replay log pathname required
-901	336986117	fbsvcmgr_query_err	Unknown tag (@1) in isc_svc_query() results
-901	336986118	fbsvcmgr_switch_unknown	Unknown switch "@1"
-901	336330758	gbak_unknown_device	Device type @1 not known
-901	335544327	bad_req_handle	Invalid request handle
-901	336330796	gbak_inv_bkup_ver	Expected backup version 1..8.Found @1
-901	336330802	gbak_unexp_eof	Unexpected end of file on backup file
-902	335544333	bug_check	Internal gds software consistency check (%s)
-902	335544335	db_corrupt	Database file appears corrupt (%s)
-902	335544344	io_error	I/O error for file %.0s"%s"
-902	335544346	metadata_corrupt	Corrupt system table
-902	335544373	sys_request	Operating system directive %s failed
-902	335544384	badblk	Internal error
-902	335544385	invpoolcl	Internal error

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-902	335544387	relbadblk	Internal error
-902	335544388	blktoobig	Block size exceeds implementation restriction
-902	335544394	badodsver	Incompatible version of on-disk structure
-902	335544397	dirtypage	Internal error
-902	335544398	waifortra	Internal error
-902	335544399	doubleloc	Internal error
-902	335544400	nodnotfnd	Internal error
-902	335544401	dupnodfnd	Internal error
-902	335544402	locnotmar	Internal error
-902	335544404	corrupt	Database corrupted
-902	335544405	badpage	Checksum error on database page %ld
-902	335544406	badindex	Index is broken
-902	335544409	trareqmis	Transaction--request mismatch (synchronization error)
-902	335544410	badhndcnt	Bad handle count
-902	335544411	wrotpbver	Wrong version of transaction parameter block
-902	335544412	wroblrver	Unsupported BLR version (expected %ld, encountered %ld)
-902	335544413	wrodpbver	Wrong version of database parameter block
-902	335544415	badrelation	Database corrupted
-902	335544416	nodetach	Internal error
-902	335544417	notremote	Internal error
-902	335544422	dbfile	Internal error
-902	335544423	orphan	Internal error
-902	335544432	lockmanerr	Lock manager error
-902	335544436	sqlerr	SQL error code = %ld
-902	335544448	bad_sec_info	
-902	335544449	invalid_sec_info	
-902	335544470	buf_invalid	Cache buffer for page %ld invalid
-902	335544471	indexnotdefined	There is no index in table %s with id %d

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-902	335544472	login	Your user name and password are not defined. Ask your database administrator to set up a Firebird login.
-902	335544506	shutinprog	Database %s shutdown in progress
-902	335544528	shutdown	Database %s shutdown
-902	335544557	shutfail	Database shutdown unsuccessful
-902	335544569	dsql_error	Dynamic SQL Error
-902	335544653	psw_attach	Cannot attach to password database
-902	335544654	psw_start_trans	Cannot start transaction for password database
-902	335544717	err_stack_limit	Stack size insufficent to execute current request
-902	335544721	network_error	Unable to complete network request to host "%s".
-902	335544722	net_connect_err	Failed to establish a connection.
-902	335544723	net_connect_listen_err	Error while listening for an incoming connection.
-902	335544724	net_event_connect_err	Failed to establish a secondary connection for event processing.
-902	335544725	net_event_listen_err	Error while listening for an incoming event connection request.
-902	335544726	net_read_err	Error reading data from the connection.
-902	335544727	net_write_err	Error writing data to the connection.
-902	335544732	unsupported_network_drive	Access to databases on file servers is not supported.
-902	335544733	io_create_err	Error while trying to create file
-902	335544734	io_open_err	Error while trying to open file
-902	335544735	io_close_err	Error while trying to close file
-902	335544736	io_read_err	Error while trying to read from file
-902	335544737	io_write_err	Error while trying to write to file
-902	335544738	io_delete_err	Error while trying to delete file
-902	335544739	io_access_err	Error while trying to access file
-902	335544745	login_same_as_role_name	Your login %s is same as one of the SQL role name. Ask your database administrator to set up a valid Firebird login.
-902	335544791	file_in_use	The file %s is currently in use by another process.Try again later.

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-902	335544795	unexp_spb_form	Unexpected item in service parameter block, expected %s
-902	335544809	extern_func_dir_error	Function %s is in %s, which is not in a permitted directory for external functions.
-902	335544819	io_32bit_exceeded_err	File exceeded maximum size of 2GB.Add another database file or use a 64 bit I/O version of Firebird.
-902	335544831	conf_access_denied	Access to %s "%s" is denied by server administrator
-902	335544834	cursor_not_open	Cursor is not open
-902	335544841	cursor_already_open	Cursor is already open
-902	335544856	att_shutdown	Connection shutdown
-902	335544882	long_login	Login name too long (@1 characters, maximum allowed @2)
-904	335544324	bad_db_handle	Invalid database handle (no active connection)
-904	335544375	unavailable	Unavailable database
-904	335544381	imp_exc	Implementation limit exceeded
-904	335544386	nopoolids	Too many requests
-904	335544389	bufexh	Buffer exhausted
-904	335544391	bufinuse	Buffer in use
-904	335544393	reqinuse	Request in use
-904	335544424	no_lock_mgr	No lock manager available
-904	335544430	virmemexh	Unable to allocate memory from operating system
-904	335544451	update_conflict	Update conflicts with concurrent update
-904	335544453	obj_in_use	Object %s is in use
-904	335544455	shadow_accessed	Cannot attach active shadow file
-904	335544460	shadow_missing	A file in manual shadow %ld is unavailable
-904	335544661	index_root_page_full	Cannot add index, index root page is full.
-904	335544676	sort_mem_err	Sort error: not enough memory
-904	335544683	req_depth_exceeded	Request depth exceeded. (Recursive definition?)
-904	335544758	sort_rec_size_err	Sort record size of %ld bytes is too big
-904	335544761	too_many_handles	Too many open handles to database

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-904	335544792	service_att_err	Cannot attach to services manager
-904	335544799	svc_name_missing	The service name was not specified.
-904	335544813	optimizer_between_err	Unsupported field type specified in BETWEEN predicate.
-904	335544827	exec_sql_invalid_arg	Invalid argument in EXECUTE STATEMENT-cannot convert to string
-904	335544828	exec_sql_invalid_req	Wrong request type in EXECUTE STATEMENT '%s'
-904	335544829	exec_sql_invalid_var	Variable type (position %d) in EXECUTE STATEMENT '%s' INTO does not match returned column type
-904	335544830	exec_sql_max_call_exceeded	Too many recursion levels of EXECUTE STATEMENT
-904	335544832	wrong_backup_state	Cannot change difference file name while database is in backup mode
-904	335544852	partner_idx_incompat_type	Partner index segment no %d has incompatible data type
-904	335544857	blobtoobig	Maximum BLOB size exceeded
-904	335544862	record_lock_not_supp	Stream does not support record locking
-904	335544863	partner_idx_not_found	Cannot create foreign key constraint %s. Partner index does not exist or is inactive.
-904	335544864	tra_num_exc	Transactions count exceeded. Perform backup and restore to make database operable again
-904	335544865	field_disappeared	Column has been unexpectedly deleted
-904	335544878	concurrent_transaction	Concurrent transaction number is @1
-906	335544744	max_att_exceeded	Maximum user count exceeded.Contact your database administrator.
-909	335544667	drdb_completed_with_errs	Drop database completed with errors
-911	335544459	rec_in_limbo	Record from transaction %ld is stuck in limbo
-913	335544336	deadlock	Deadlock
-922	335544323	bad_db_format	File %s is not a valid database
-923	335544421	connect_reject	Connection rejected by remote interface
-923	335544461	cant_validate	Secondary server attachments cannot validate databases

Table X-1. Firebird 2.1 Error Codes

SQLCODE	GDSCODE	SYMBOL	ENGLISH TEXT OF ERROR MESSAGE
-923	335544464	cant_start_logging	Secondary server attachments cannot start logging
-924	335544325	bad_dpb_content	Bad parameters on attach or create database
-924	335544441	bad_detach	Database detach completed with errors
-924	335544648	conn_lost	Connection lost to pipe server
-926	335544447	no_rollback	No rollback performed
-999	335544689	ib_error	Firebird error

Appendix XI
Reserved Words

TABLE XI-1 CONTAINS KEYWORDS THAT are reserved in some way in Firebird. Some have special markings:

KEYWORD (con.) marks words that are reserved in their specific contexts. For example, the word UPDATING is a keyword in PSQL and will be disallowed as a variable or argument name.

[KEYWORD] marks words that are not currently reserved words but should be treated as such for planned future implementation or for compatibility with InterBase.

/* KEYWORD */ marks words that are reserved words in Firebird 1.0.x but were released in Firebird 1.5.

Changes in Firebird 2

Those keywords that are new in Firebird 2, or have changed status since Firebird 1.5, will be found in BOLD FACE, each on a line of its own, with comments. Of those, the ones marked with an asterisk (*) are not present in the SQL standard.

Table XI-1. Firebird Reserved Words

ABS is a keyword from v.2.1 onward, but is not reserved

ACCENT * is a keyword from v.2.1 onward, but is not reserved

ACOS * is a keyword from v.2.1 onward, but is not reserved

ACTION remains a keyword but is non-reserved from v.2.0 on

ACTIVE	ADD	ADMIN
AFTER	ALL	ALTER

ALWAYS * is a keyword from v.2.1 onward, but is not reserved

AND	ANY	ARE
AS	ASC	ASCENDING

ASCII_CHAR * is a keyword from v.2.1 onward, but is not reserved

ASCII_VAL * is a keyword from v.2.1 onward, but is not reserved

ASIN * is a keyword from v.2.1 onward, but is not reserved

AT

ATAN * is a keyword from v.2.1 onward, but is not reserved

Table XI-1. Firebird Reserved Words

ATAN2 * is a keyword from v.2.1 onward, but is not reserved

AUTO	AUTODDL	AVG

BACKUP * is a new keyword, but it is not reserved.

BASED	BASE_NAME

BASENAME * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

BEFORE	BEGIN	BETWEEN
BIGINT		

BIN_AND * is a keyword from v.2.1 onward, but is not reserved

BIN_NOT * will be a keyword from v.2.1.1 onward, but is not reserved

BIN_OR * is a keyword from v.2.1 onward, but is not reserved

BIN_SHL * is a keyword from v.2.1 onward, but is not reserved

BIN_SHR * is a keyword from v.2.1 onward, but is not reserved

BIN_XOR * is a keyword from v.2.1 onward, but is not reserved

BIT_LENGTH (new reserved keyword)

BLOB	BLOBEDIT

BLOCK * (a new keyword, but not reserved)

[BOOLEAN]

BOTH is a new keyword, but it is not reserved.

/* BREAK */	BUFFER	BY

CACHE * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

CASCADE remains a keyword but is non-reserved from v.2.0 on.

CASE	CAST

CEIL and **CEILING** are keywords from v.2.1 onward but are not reserved

CHAR	CHARACTER

CHAR_LENGTH (new reserved keyword)

CHARACTER_LENGTH (new reserved keyword)

CHECK

Table XI-1. Firebird Reserved Words

CHECK_POINT_LEN * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

CHECK_POINT_LENGTH

CLOSE (claimed as a new reserved keyword, although appears to have had that status already at v.1.5)

COALESCE (con.)	COLLATE	

COLLATION was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

COMMENT * is a new keyword, but it is not reserved.

COMMIT	COMMITTED	COMPILETIME
COMPUTED	CONDITIONAL	CONNECT
CONSTRAINT	CONTAINING	CONTINUE

COS * is a keyword from v.2.1 onward but is not reserved

COSH * is a keyword from v.2.1 onward but is not reserved

COT * is a keyword from v.2.1 onward but is not reserved

COUNT	CREATE	

CROSS (new reserved keyword)

CSTRING	CURRENT	CURRENT_CONNECTION
CURRENT_DATE	CURRENT_ROLE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_TRANSACTION	CURRENT_USER
DATABASE	DATE	

DATEADD * is a keyword from v.2.1 onward but is not reserved

DATEDIFF * is a keyword from v.2.1 onward but is not reserved

DAY	DB_KEY	DEBUG
DEC	DECIMAL	DECLARE

DECODE * is a keyword from v.2.1 onward but is not reserved

DEFAULT

[DEFERRED]	DELETE	DELETING (con.)
DESC	DESCENDING	DESCRIBE

Table XI-1. Firebird Reserved Words

/* DESCRIPTOR */

DIFFERENCE * (a new keyword but not reserved)

DISCONNECT	DISPLAY	DISTINCT
DO	DOMAIN	DOUBLE
DROP	ECHO	EDIT
ELSE	END	ENTRY_POINT
ESCAPE	EVENT	EXCEPTION
EXECUTE	EXISTS	EXIT

EXP is a keyword from v.2.1 onward but is not reserved

EXTERN	EXTERNAL	EXTRACT
[FALSE]		

FETCH (claimed as a new reserved keyword, although appears to have had that status already at v.1.5)

FILE	FILTER	/* FIRST */
FLOAT		

FLOOR is a keyword from v.2.1 onward but is not reserved

FOR	FOREIGN	FOUND

FREE_IT * remains a keyword but is non-reserved from v.2.0 on.

FROM	FULL	FUNCTION
GDSCODE	GEN_ID	

GEN_UUID is a keyword from v.2.1 onward but is not reserved

GENERATED is a keyword from v.2.1 onward but is not reserved

GENERATOR

GLOBAL (reserved from v.2.1 onward)

GOTO	GRANT	GROUP

GROUP_COMMIT_WAIT * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

GROUP_COMMIT_WAIT_TIME

Table XI-1. Firebird Reserved Words

HASH * (Keyword from v.2.1 onward but not reserved)

HAVING	HEADING	HELP
HOUR	IF	

IIF * removed in v.1.5, reinstated in v.2.0 as a keyword without reserved status

IMMEDIATE	IN	INACTIVE
INDEX	INDICATOR	INIT
INNER	INPUT	INPUT_TYPE
INSERT	INSERTING (con.)	

INSENSITIVE (reserved from v.2.1 onward)

INT	INTEGER	INTO
IS	ISOLATION	ISQL
JOIN	KEY	LAST (con.)
LC_MESSAGES	LC_TYPE	

LEADING (new reserved keyword)

LEAVE (con.)	LEFT	LENGTH
LEV	LEVEL	LIKE

LIST * is a keyword from v.2.1 onward but is not reserved

LN * is a keyword from v.2.1 onward but is not reserved

LOCK (con.)

LOG * is a keyword from v.2.1 onward but is not reserved

LOG10 * is a keyword from v.2.1 onward but is not reserved

LOGFILE * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

LOG_BUFFER_SIZE

LOG_BUF_SIZE * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

LONG

LOWER (new reserved keyword)

Table XI-1. Firebird Reserved Words

LPAD * is a keyword from v.2.1 onward but is not reserved

MANUAL

MATCHED is a keyword from v.2.1 onward but is not reserved

MATCHING * is a keyword from v.2.1 onward but is not reserved

MAX	MAX_SEGMENT	MAXIMUM
MAXIMUM_SEGMENT		

MAXVALUE * is a keyword from v.2.1 onward but is not reserved

MERGE	MESSAGE	MIN
MINIMUM	MINUTE	

MINVALUE * is a keyword from v.2.1 onward but is not reserved

MOD is a keyword from v.2.1 onward but is not reserved

MODULE_NAME	MONTH	NAMES
NATIONAL	NATURAL	NCHAR

NEXT (a new keyword but not reserved)

NO	NOAUTO	NOT
NULL	NULLIF (con.)	NULLS (con.)

NUM_LOG_BUFS * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

NUM_LOG_BUFFERS	NUMERIC	

OCTET_LENGTH (new reserved keyword)

OF	ON	ONLY

OPEN (claimed as a new reserved keyword, although appears to have had that status already at v.1.5)

OPTION	OR	ORDER
OUTER	OUTPUT	OUTPUT_TYPE

OVERLAY is a keyword from v.2.1 onward but is not reserved

OVERFLOW

PAD is a keyword from v.2.1 onward but is not reserved

Table XI-1. Firebird Reserved Words

PAGE	PAGELENGTH	PAGES
PAGE_SIZE	PARAMETER	PASSWORD
[PERCENT]		

PI * is a keyword from v.2.1 onward but is not reserved

PLACING is a keyword from v.2.1 onward but is not reserved

PLAN	POSITION	POST_EVENT

POWER is a keyword from v.2.1 onward but is not reserved

PRECISION	PREPARE	

PRESERVE is a keyword from v.2.1 onward but is not reserved

PRIMARY	PRIVILEGES	PROCEDURE
PUBLIC	QUIT	

RAND * is a keyword from v.2.1 onward but is not reserved

RAW_PARTITIONS * was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

RDB$DB_KEY	READ	REAL
RECORD_VERSION	RECREATE	REFERENCES

RECURSIVE (reserved from v.2.1 onward)

RELEASE

REPLACE * is a keyword from v.2.1 onward but is not reserved

RESERVE	RESERVING	

RESTART (a new keyword but not reserved)

RESTRICT was formerly a reserved keyword; from v.2.0, it is neither reserved nor a keyword.

RETAIN	RETURN	

RETURNING * (a new keyword but not reserved)

RETURNING_VALUES	RETURNS	REVOKE

REVERSE * is a keyword from v.2.1 onward but is not reserved

RIGHT

Table XI-1. Firebird Reserved Words

ROLE (kw non-res, changed)

ROLLBACK

ROUND * is a keyword from v.2.1 onward but is not reserved

ROW_COUNT

ROWS (new reserved keyword)

RPAD * is a keyword from v.2.1 onward but is not reserved

RUNTIME	SAVEPOINT	

SCALAR_ARRAY * (a new keyword but not reserved)

SCHEMA	SECOND	SELECT

SENSITIVE (reserved from v.2.1 onward)

SEQUENCE (a new keyword but not reserved)

SET	SHADOW	SHARED
SHELL	SHOW	

SIGN * is a keyword from v.2.1 onward but is not reserved

SIN * is a keyword from v.2.1 onward but is not reserved

SINGULAR

SINH * is a keyword from v.2.1 onward but is not reserved

SIZE	/* SKIP */	SMALLINT
SNAPSHOT	SOME	SORT

SPACE is a keyword from v.2.1 onward but is not reserved

SQL	SQLCODE	SQLERROR

SQRT is a keyword from v.2.1 onward but is not reserved

SQLWARNING	STABILITY	

START is a reserved word from v.2.1 onward

STARTING

STARTS	STATEMENT (con.)	STATIC
STATISTICS	SUB_TYPE	/* SUBSTRING */

Table XI-1. Firebird Reserved Words

SUM	SUSPEND	TABLE
TAN * is a keyword from v.2.1 onward but is not reserved		
TANH * is a keyword from v.2.1 onward but is not reserved		
TEMPORARY is a keyword from v.2.1 onward but is not reserved		
TERM	TERMINATOR	THEN
[TIES]	TIME	TIMESTAMP
TO		
TRAILING (new reserved keyword)		
TRANSACTION	TRANSLATE	TRANSLATION
TRIGGER		
TRIM (new reserved keyword)		
[TRUE]		
TRUNC * is a keyword from v.2.1 onward but is not reserved		
TYPE remains a keyword but is non-reserved from v.2.0 on.		
UNCOMMITTED	UNION	UNIQUE
[UNKNOWN]	UPDATE	UPDATING (con.)
UPPER	USER	
USING (new status: reserved keyword)		
VALUE	VALUES	VARCHAR
VARIABLE	VARYING	VERSION
VIEW	WAIT	
WEEK * is a keyword from v.2.1 onward but is not reserved		
WEEKDAY * remains a keyword but is non-reserved from v.2.0 on.		
WHEN	WHENEVER	WHERE
WHILE	WITH	WORK
WRITE	YEAR	
YEARDAY * remains a keyword but is non-reserved from v.2.0 on.		

Appendix XII
Readings and Resources

No supplementary information.

Appendix XIII Miscellaneous Improvements

In this Appendix is a miscellany of improvements that didn't find a home elsewhere in this supplement. A copy of the update script for the security database appears at the end.

Miscellaneous

Improved Diagnostic and Logging Facilities

- BUGCHECK log messages now include file name and line number
- Routines that print out various internal structures (DSQL node tree, BLR, DYN, etc) have been updated
- Thread-safe and signal-safe debug logging facilities have been implemented
- Syslog messages will be copied to the user's tty if a process is attached to it
- Lock manager memory dumps have been made more informative and OWN_hung is detected correctly

Invariant Tracking in PSQL

Invariant tracking in PSQL and request cloning logic were reworked to fix a number of issues with recursive procedures. The changes will affect particularly the accuracy and performance of complex expression evaluations in this environment.

Invariant tracking is the process performed by the BLR compiler and the optimizer to decide whether an “invariant” (an expression, which might be a nested subquery) is independent from the parent context. It is used to perform one-time evaluations of such expressions and then cache the result.

If some invariant is not determined, we lose in performance. If some variant is wrongly treated as invariant, we see wrong results.

Example

```
select * from rdb$relations
where rdb$relation_id <
( select rdb$relation_id from rdb$database )
```

This query now performs only one fetch from rdb$database instead of evaluating the subquery for every row of rdb$relations.

Changes to Synchronisation Logic

1. Lock contention in the lock manager and in the SuperServer thread pool manager has been reduced significantly
2. A rare race condition was detected and fixed, that could cause Superserver to hang during request

processing until the arrival of the next request

3. Decoupling of lock manager synchronization objects for different engine instances was implemented

Improved Connection Handling by Superserver on POSIX

Posix SS builds now handle SIGTERM and SIGINT to shut down all connections gracefully.

“ODS Type” Recognition

Firebird 2.0 introduced a concept of *ODS type* to distinguish between Firebird and InterBase databases. It is a physical page layout detail that prevents Firebird and InterBase from sharing ODS11 databases. Technically, starting with ODS 11, the highest bit of the major ODS version field is set to 1. It is an internal setting that cannot be accessed in any way from clients.

Security Upgrade Script

```
/* Script security_database.sql
*
* The contents of this file are subject to the Initial
* Developer's Public License Version 1.0 (the  “License");
* you may not use this file except in compliance with the
* License. You may obtain a copy of the License at
*  http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_idpl.
*
* Software distributed under the License is distributed AS IS,
* WITHOUT WARRANTY OF ANY KIND, either express or implied.
* See the License for the specific language governing rights
* and limitations under the License.
*
* The Original Code was created by Alex Peshkov on 16-Nov-2004
* for the Firebird Open Source RDBMS project.
*
* Copyright (c) 2004 Alex Peshkov
* and all contributors signed below.
*
* All Rights Reserved.
*  Contributor(s): ______________________________________.
*
*/

-- 1. temporary table to alter domains correctly.
CREATE TABLE UTMP (
USER_NAME VARCHAR(128) CHARACTER SET ASCII,
```

```
SYS_USER_NAME VARCHAR(128) CHARACTER SET ASCII,
GROUP_NAME VARCHAR(128) CHARACTER SET ASCII,
UID INTEGER,
GID INTEGER,
PASSWD VARCHAR(64) CHARACTER SET BINARY,
PRIVILEGE INTEGER,
COMMENT BLOB SUB_TYPE TEXT SEGMENT SIZE 80
CHARACTER SET UNICODE_FSS,
FIRST_NAME VARCHAR(32) CHARACTER SET UNICODE_FSS
DEFAULT _UNICODE_FSS '',
MIDDLE_NAME VARCHAR(32) CHARACTER SET UNICODE_FSS
DEFAULT _UNICODE_FSS '',
LAST_NAME VARCHAR(32) CHARACTER SET UNICODE_FSS
DEFAULT _UNICODE_FSS ''
);
COMMIT;

-- 2. save users data
INSERT INTO UTMP(USER_NAME, SYS_USER_NAME, GROUP_NAME,
UID, GID, PRIVILEGE, COMMENT, FIRST_NAME, MIDDLE_NAME,
LAST_NAME, PASSWD)
SELECT USER_NAME, SYS_USER_NAME, GROUP_NAME,
UID, GID, PRIVILEGE, COMMENT, FIRST_NAME, MIDDLE_NAME,
LAST_NAME, PASSWD
FROM USERS;
COMMIT;

-- 3. drop old tables and domains
DROP TABLE USERS;
DROP TABLE HOST_INFO;
COMMIT;

DROP DOMAIN COMMENT;
DROP DOMAIN NAME_PART;
DROP DOMAIN GID;
DROP DOMAIN HOST_KEY;
DROP DOMAIN HOST_NAME;
DROP DOMAIN PASSWD;
DROP DOMAIN UID;
DROP DOMAIN USER_NAME;
DROP DOMAIN PRIVILEGE;
COMMIT;
```

```
-- 4. create new objects in database
CREATE DOMAIN RDB$COMMENT AS BLOB SUB_TYPE TEXT SEGMENT SIZE 80
CHARACTER SET UNICODE_FSS;
CREATE DOMAIN RDB$NAME_PART AS VARCHAR(32)
CHARACTER SET UNICODE_FSS DEFAULT _UNICODE_FSS '';
CREATE DOMAIN RDB$GID AS INTEGER;
CREATE DOMAIN RDB$PASSWD AS VARCHAR(64) CHARACTER SET BINARY;
CREATE DOMAIN RDB$UID AS INTEGER;
CREATE DOMAIN RDB$USER_NAME AS VARCHAR(128)
CHARACTER SET UNICODE_FSS;
CREATE DOMAIN RDB$USER_PRIVILEGE AS INTEGER;
COMMIT;

CREATE TABLE RDB$USERS (
RDB$USER_NAME RDB$USER_NAME NOT NULL PRIMARY KEY,
/* local system user name
for setuid for file permissions */
RDB$SYS_USER_NAME RDB$USER_NAME,
RDB$GROUP_NAME RDB$USER_NAME,
RDB$UID RDB$UID,
RDB$GID RDB$GID,
RDB$PASSWD RDB$PASSWD, /* SEE NOTE BELOW */

/* Privilege level of user -
mark a user as having DBA privilege */
RDB$PRIVILEGE RDB$USER_PRIVILEGE,

RDB$COMMENT RDB$COMMENT,
RDB$FIRST_NAME RDB$NAME_PART,
RDB$MIDDLE_NAME RDB$NAME_PART,
RDB$LAST_NAME RDB$NAME_PART);
COMMIT;

CREATE VIEW USERS (USER_NAME, SYS_USER_NAME, GROUP_NAME,
UID, GID, PASSWD, PRIVILEGE, COMMENT, FIRST_NAME,
MIDDLE_NAME, LAST_NAME, FULL_NAME) AS

SELECT RDB$USER_NAME, RDB$SYS_USER_NAME, RDB$GROUP_NAME,
RDB$UID, RDB$GID, RDB$PASSWD, RDB$PRIVILEGE, RDB$COMMENT,
RDB$FIRST_NAME, RDB$MIDDLE_NAME, RDB$LAST_NAME,
RDB$first_name || _UNICODE_FSS ' ' || RDB$middle_name
```

```
|| _UNICODE_FSS ' ' || RDB$last_name
FROM RDB$USERS
WHERE CURRENT_USER = 'SYSDBA'
OR CURRENT_USER = RDB$USERS.RDB$USER_NAME;
COMMIT;

GRANT ALL ON RDB$USERS to VIEW USERS;
GRANT SELECT ON USERS to PUBLIC;
GRANT UPDATE(PASSWD, GROUP_NAME, UID, GID, FIRST_NAME,
MIDDLE_NAME, LAST_NAME)
ON USERS TO PUBLIC;
COMMIT;

-- 5. move data from temporary table and drop it
INSERT INTO RDB$USERS(RDB$USER_NAME, RDB$SYS_USER_NAME,
RDB$GROUP_NAME, RDB$UID, RDB$GID, RDB$PRIVILEGE, RDB$COMMENT,
RDB$FIRST_NAME, RDB$MIDDLE_NAME, RDB$LAST_NAME, RDB$PASSWD)
SELECT USER_NAME, SYS_USER_NAME, GROUP_NAME, UID, GID,
PRIVILEGE, COMMENT, FIRST_NAME, MIDDLE_NAME, LAST_NAME,
PASSWD
FROM UTMP;
COMMIT;

DROP TABLE UTMP;
COMMIT;
```

The field **RDB$PASSWD** should be constrained as NOT NULL. For information about this, see the warning 20 in the topic, *Upgrading Your Security Database*, in Chapter 1, *Installation*.

Index

A

B

C

D

E

F

G

I

K

L

M

N

O

P

Q

R

S

IBPhoenix Publications

Verneuil le Chateau
FRANCE