Firebird 2.5 Release Notes

Helen Borrie (Collator/Editor)

28 March 2009 - Document v.0250_14 - for Firebird 2.5 Beta 1

Firebird 2.5 Release Notes

28 March 2009 - Document v.0250_14 - for Firebird 2.5 Beta 1

Helen Borrie (Collator/Editor)

Table of Contents

List of Tables

Chapter 1

General Notes

Bug Reporting

- If you think you have discovered a new bug in this release, please make a point of reading the instructions for bug reporting in the article How to Report Bugs Effectively, at the Firebird Project website.
- If you think a bug fix hasn't worked, or has caused a regression, please locate the original bug report in the Tracker, reopen it if necessary, and follow the instructions below.

Follow these guidelines as you attempt to analyse your bug:

1. Write detailed bug reports, supplying the exact build number of your Firebird kit. Also provide details of the OS platform. Include reproducible test data in your report and post it to our Tracker.
2. You are warmly encouraged to make yourself known as a field-tester of this alpha by subscribing to the field-testers' list and posting the best possible bug description you can.
3. If you want to start a discussion thread about a bug or an implementation, please do so by subscribing to the firebird-devel list. In that forum you might also see feedback about any tracker ticket you post regarding this alpha.

Documentation

You will find all of the README documents referred to in these notes in the doc sub-directory of your Firebird 2.5 Alpha 1 installation.

An automated "Release Notes" page in the Tracker provides lists and links for all of the Tracker tickets associated with this alpha. Use this link.

--*The Firebird Project*

Chapter 2

New in Firebird 2.5

The primary goal for Firebird 2.5 was to establish the basics for a new threading architecture that is almost entirely common to the Superserver, Classic and Embedded models, taking in lower level synchronization and thread safety generally.

Although SQL enhancements are not a primary objective of this release, for the first time, user management becomes accessible through SQL CREATE/ALTER/DROP USER statements and syntaxes for ALTER VIEW and CREATE OR ALTER VIEW are implemented. PSQL improvements include the introduction of autonomous transactions and ability to query another database via EXECUTE STATEMENT.

Other New Features

Other new features and improvements in this release include:

Administrative Enhancements

- System audit tracing and user trace sessions via the Services API, making it possible to monitor and analyse everything going on in a database in real time
- New system role RDB$ADMIN in the ODS 11.2 database allows SYSDBA to transfer its privileges to another user on a per-database basis
- More information in the monitoring tables
- Asynchronous cancellation of connections
- Capability for ordinary users to monitor any of their own attachments as well as CURRENT_CONNECTION

Other SQL Language Additions

- Regular expression support using the SIMILAR TO predicate
- ALTER COLUMN for computed columns
- Autonomous transactions within a PSQL module (stored procedure, trigger or dynamically executable PSQL block)
- Enhanced access to stored procedures in view definitions
- Optional GRANTED BY or GRANTED AS for GRANT and REVOKE statements, enabling the grantor to be a user other than the CURRENT_USER (the default).
- REVOKE ALL syntax to dispose of all privileges for a user or role at once

- Support for WHERE SOME_COL = ? OR ? IS NULL predications

Data-handling Enhancements

- New built-in functions for converting UUID CHAR(16) OCTETS strings to RFC4122-compliant format and vice versa
- Ability to pass 32-bit and 64-bit integers as hexadecimal in numeric literal and X-prefixed binary string literal formats

API Additions

- Statements now return an SQL-2003 standard 5-alphanumeric SQLSTATE completion code
- New constant DSQL_unprepare available for use with isc_dsql_free_statement for efficient unpreparing of statements

International Language Support

- Default COLLATE clause for CREATE DATABASE
- Ability to change the default COLLATE for a used character set
- GBAK restore switches FIX_FSS_DATA and FIX_FSS_METADATA to restore legacy databases with UNICODE_FSS data and/or metadata correctly without resorting to scripts and manual techniques
- Accent-insensitive collation for Unicode

Chapter 3

Changes in the Firebird Engine

The primary objective of this release was to refactor Firebird's threading architecture to take advantage of the symmetric multiprocessing (SMP) capabilities of multiprocessor hardware. This has a noticeable effect on the scaleability of Superserver when multiple databases are being accessed simultaneously but its major effect is the emergence of the architectural “Superclassic” model that will underpin the fine-grained multi-threading under development for Firebird 3.

New Threading Architecture

Dmitry Yemanov
Vladyslav Khorsun
Alex Peshkov
also -
Nickolay Samofatov
Roman Simakov

For Superserver, the new architecture will be most obvious in two ways:

1. Superserver threads distributed evenly to available processors according the the database clients attach to.

> **Note**
>
> The default *CpuAffinity* setting still binds SuperServer to a single processor only. In order to scale better when working with multiple databases, this setting should be changed in `firebird.conf`.
>
> The default value may change before the final release.

2. A slight improvement in scaling for single database usage on SMP hardware

It is with Classic that the effects are most evident:

1. Classic Server can now be multi-threaded. The one worker thread per process model remains but now it is possible to use additional threads for parallel tasks such as asynchronous shutdown, sweep, inter-process communications with the lock manager and more.

2. On POSIX, services in Classic also run in threads now, rather than in forked processes as previously.

> **Note**
>
> For Windows Classic, services became threadable in v.2.1.

3. The embedded libraries—*libfbembed.so* on POSIX and *fbembed.dll* on Windows—are now multi-thread-capable and thread-safe, so they can be used in multi-threaded applications.
4. Testing suggests that the performance of Classic in this version will be be significantly faster than previous Classic versions.

"Superclassic"

This multi-threaded mode for Classic has been dubbed "Superclassic" for its capability to handle multiple worker threads—dedicated or pooled—inside a single server process. It shares all the usual Classic features, with a few differences:

- Safe, full shutdown of the server engine is possible on any platform
- Under some TPC conditions, it can outperform Classic—by about 15-20%
- It uses fewer kernel resources (although not less memory)
- When a Superclassic process crashes, it takes all its connections with it
- Recognised limitations in the Services API for the Classic server, such as the inability to retrieve the list of attachments/active users, do not apply to SuperClassic.
- On POSIX, Superclassic does not require *[x]inetd*.

> **Embedded Server**
>
> The embedded server in the Windows library, *fbembed.dll*, now uses Superclassic, not Superserver as previously, thus unifying its model with that of local connection to Superclassic on POSIX. The database file-lock that previously restricted connections to a single application space is replaced by a global lock table that allows simultaneous access to the same database from different embedded server modules. This facilitates concurrent debugging of applications and use of native utility tools like *gbak*, *gstat* and so on.

Usage Notes

Windows

On Windows, the same `fb_inet_server.exe` binary delivers either the Classic or the Superclassic working modes, according to switch settings. Classic is the default mode.

To use the Superclassic mode as a service, add the `-m[ulti-threaded]` switch to the *instsvc.exe*command line, as follows:

```
instsvc install -multithreaded
```

When intending to run Superclassic *as an application*, use

```
fb_inet_server -a -m
```

New Binary for POSIX

On POSIX, the new binary `fb_smp_server` is supplied for the Superclassic model. It contains the network listener, meaning it works similarly to `fbserver` with regard to attachment requests and does not require *[x]inetd*.

The multi-threaded engine used by `fb_smp_server` is the `libfbembed.so` library, in accordance with OSRI requirements. The Classic packages also include `fbguard` (the Guardian) which, for Classic, starts `fb_smp_server`, rather than `fbserver` as it does when the Superserver model is installed with Guardian.

Thread-safe Client Library

Dmitry Yemanov
Vladyslav Khorsun
Alex Peshkov

Tracker reference CORE-707.

The client libraries, including the embedded one, can now be used in multi-threaded applications without any application-level synchronization.

Improvements

Improvements implemented include:

Immediate Detection of Disconnected Clients on Classic

Vladyslav Khorsun

The Classic server now detects immediately when a Classic process has been broken by a client disconnection. Its response is to terminate any pending activity, roll back the active transaction and close the network connection.

Tracker reference CORE-818.

Optimizations

Important optimizations include:

Data Retrieval

Dmitry Yemanov

An optimization improves data retrieval performance for tables from which no fields are accessed. This applies, for example to the **SELECT COUNT(*)** style of query.

Tracker reference CORE-1598.

BLOB Memory Usage

Adriano dos Santos Fernandes

An optimization avoids memory consumption of <page size> bytes for each temporary BLOB created during assignment.

Tracker reference CORE-1658.

DLL Loading for Windows Embedded Engine

Adriano dos Santos Fernandes

The root determination mechanism for the Windows embedded engine has been changed to avoid common problems that occur when an installation of the application structure encounters “DLL Hell”. Previously, the implicit root directory was the directory containing the user application's main executable file. Now it is the directory where the renamed *fbembed.dll* library is located.

Tracker reference CORE-1814.

UDFs Safeguard

Adriano dos Santos Fernandes

Tracker reference CORE-1937.

When a string UDF is written to return a pointer not allocated by the same runtime as the Firebird server is accessing, the presence of the FREE_IT keyword in its declaration corrupts memory and crashes the server. As a safeguard against such dysfunctional UDFs, the engine now

1. detects such UDFs and throws an exception
2. depends on the presence of the updated *ib_util* library in the path for all server models, including embedded

Diagnostics

Transaction Diagnostics

Claudio Valderrama

Better diagnostics and error reporting when TPB contents are malformed. The new TPB validation logic now rejects:

- explicitly conflicting options within the same category, e.g., **{WAIT}** and **{NOWAIT}** specified together, or **{READ COMMITTED}** and **{SNAPSHOT}**, or **{READ ONLY}** and **{WRITE}**
- options making no sense, e.g. **[NO] RECORD VERSION** specified for a SNAPSHOT isolation mode
- incorrect order of table reservation options, e.g. **{PROTECTED READ <TABLE>}** instead of **{READ <TABLE> PROTECTED}**

Tracker reference CORE-1600.

Access Privilege Error Messages

Alex Peshkov

Both table and column names are now reported when access privilege exceptions occur for a column.

Tracker reference CORE-1234.

Metadata Improvements

Preserve Character Set Default Collation

Adriano dos Santos Fernandes

An improvement allows the current value of RDB$DEFAULT_COLLATE_NAME in the system table RDB $CHARACTER_SETS to survive the backup/restore cycle. The mechanism for such customisation is the new ALTER CHARACTER SET command.

Tracker reference CORE-789.

Chapter 4

Changes to the Firebird API and ODS

ODS (On-Disk Structure) Changes

On-disk structure (ODS) changes include the following:

New ODS Number

Firebird 2.5 creates databases with an ODS (On-Disk Structure) version of 11.2

API (Application Programming Interface) Extensions

Additions to the Firebird API include.-

Connection Strings & Character Sets

A. dos Santos Fernandes

Previous versions had no way to interoperate with the character set(s) used by the operating system and its filesystem. Firebird 2.5 has been made “environmentally aware” with regard to the file names of databases and other files accessed through API connection requests, improving significantly its ability to accept and work with file names containing characters that are not in the ASCII subset.

> **Only DPB Connections Support this Feature**
>
> In the current implementation, only connections made through the DPB (database parameter block) support this feature. It is not supported for Services API (*isc_spb**) functions.

isc_dpb_utf8_filename

The new connection option **isc_dpb_utf8_filename** has been introduced, to enable Firebird to be specifically informed that the file name being passed is in the UTF8 (UTF-8) character set. If the option is not used, the character set defaults to the codepage of the operating system.

Client-Server Compatibility

New client, older server

If the client is V.2.5 or newer and it is connecting to a pre-V.2.5 remote server, using the **isc_dpb_utf8_filename** option causes the *client* to convert the file name from UTF-8 to the *client codepage* before passing it to the server. It removes the **isc_dpb_utf8_filename** option from the DPB.

Compatibility is assured when the same codepage is being used on both the the client and server stations.

New client, new server, without isc_dpb_utf8_filename
If the client is V.2.5 or newer and it is connecting to a V.2.5 or newer remote server without using the **isc_dpb_utf8_filename**, the client converts the file name from the OS codepage to UTF-8 and inserts the **isc_dpb_utf8_filename** option into the DPB.

The file name received on the server is not subject to any special treatment. However, unlike older clients, the V.2.5 client may convert the file name automatically and insert the **isc_dpb_utf8_filename** option into the DPB. Compatibility is guaranteed, regardless, when the host and client are using the same code page.

New client, new server, with isc_dpb_utf8_filename
Whenever the **isc_dpb_utf8_filename** option is used, the client passes the unmodified filename to the server. The client thus always passes a UTF-8 file name to the server along with the **isc_dpb_utf8_filename** option.

Code Page Conversions

On Windows the code page used for conversions is Windows ANSI. On all other platforms, UTF-8 is used.

The operating system codepage and UTF-8 may not be the best choice for file names. For example, if you had a script or other text file for processing in *isql* or some other script-running tool that used another connection character set, it would not be possible to edit the file correctly using multiple character sets (code pages).

There is a solution: the *Unicode code point*. If used correctly, it enables correct interpretation of a character even if the client is older than V.2.5.

Using Unicode Code Points

Any Unicode character may now now be encoded on the connection string file name as though it were an ASCII character. It is accomplished by using the symbol # as a prefix for a Unicode code point number (in hexadecimal format, similar to U+XXXX notation).

Write it as **#XXXX** with X being 0-9, a-f, A-F.

If one of the characters happens to be the literal #, you could either “double” the hash character (##) or use the code point number for it, **#0023**.

> **Note**
>
> The hash character is interpreted at the server with these new semantics, even if the client is older than v2.5.

Support for SQLSTATE Completion Codes

W. Oliver
D. Yemanov

Tracker reference CORE-1761.

A new client-side API function, **fb_sqlstate()** is available to convert the status vector item for an error into the corresponding SQL-2003 standard 5-alphanumeric SQLSTATE.

- The SQLSTATE code represents the concatenation of a 2-character SQL CLASS and a 3-character SQL SUBCLASS.
- Statements now return an SQLSTATE completion code.
- The *isql* utility now prints the SQLSTATE diagnostic for errors instead of the SQLCODE one
- The SQLCODE diagnostic is deprecated—meaning it will disappear in a future release

> **Deprecated SQLCODE**
>
> Although the SQLCODE is deprecated and use of the SQLSTATE is preferred, it remains in Firebird for the time being. The *isc_sqlcode()* API function is still supported, as is the **WHEN SQLCODE** exception handling.

Appendix A: SQLSTATE provides a list of all SQLSTATE codes in use in this release, along with the corresponding message texts.

"Efficient Unprepare"

W. Oliver
D. Yemanov

Tracker reference CORE-1741.

The new option *DSQL_unprepare* (numeric value 4) for the API routine *isc_dsql_free_statement()* allows the DSQL statement handle to survive the "unpreparing" of the statement.

Previously, the *isc_dsql_free_statement()* function supported only *DSQL_close* (for closing a named cursor) and *DSQL_drop* (which frees the statement handle).

The API addition is:

```
#define DSQL_close 1
#define DSQL_drop 2
#define DSQL_unprepare 4
```

Cancel Operation Function

Alex Peshkov

New *fb_cancel_operation()* API call, allowing cancellation of the current activity being performed by some kind of blocking API call in the given connection.

Syntax

```
ISC_STATUS fb_cancel_operation(ISC_STATUS* status_vector,
                               isc_db_handle* db_handle,
```

```
                                        ISC_USHORT option);
```

Parameters

status vector (ISC_STATUS status_vector)*
A regular status vector pointer structure.

db_handle (pointer to a isc_db_handle)
A regular, valid database handle. It identifies the attachment.

option (unsigned short: symbol)
Determines the action to be performed. The option symbols are:

- *fb_cancel_raise*: cancels any activity related to the **db_handle** specified in the second parameter. The effect will be that, as soon as possible, the engine will try to stop the running request and return an exception to the caller via the status vector (parameter 1).

  "..as soon as possible" will be, under normal conditions, at the next rescheduling point.

- *fb_cancel_disable*: disables execution of *fb_cancel_raise* requests for the specified attachment. It can be useful when your program is executing critical operations, such as cleanup, for example.

- *fb_cancel_enable*: re-enables delivery of a cancel execution that was previously disabled. The 'cancel' state is effective by default, being initialized when the attachment is created.

Usage

The cycle of *fb_cancel_disable* and *fb_cancel_enable* requests may be repeated as often as necessary. If the engine is already in the requested state there is no exception: it is simply a no-op.

Usually *fb_cancel_raise* is called when you need to stop a long-running request. It is called from a separate thread, not from the signal handler, because it is *not* async signal safe.

> **Pay attention to asynchronous nature of this API call!**

Another aspect of asynchronous execution is that, at the end of API call, the attachment's activity might be cancelled or it might not. The latter is always a possibility. The asynchronicity also means that returned status vector will almost always return *FB_SUCCESS*. Exceptions, though, are possible: a network packet error, for example.

An Example

```
Thread A:
fb_cancel_operation(isc_status, &DB, fb_cancel_enable);
isc_dsql_execute_immediate(isc_status, &DB, &TR, 0, "long running statement", 3, NULL);
// waits for API call to finish...

  Thread B:
  fb_cancel_operation(local_status, &DB, fb_cancel_raise);

Thread A:
if (isc_status[1])
```

```
isc_print_status(isc_status); // will print "operation was cancelled"
```

Shutdown Functions

Alex Peshkov

This release exposes a variety of API functions for instigating server shutdowns of various types from client applications.

Two Interrelated fb_shutdown* Functions

This release exposes two *fb_shutdown** functions that may be useful for embedded server applications: *fb_shutdown()* and *fb_shutdown_callback*.

Prototypes

```
typedef int (*FB_SHUTDOWN_CALLBACK)(const int reason, const int mask, void* arg);

int fb_shutdown(unsigned int timeout,
                const int reason);

ISC_STATUS fb_shutdown_callback(ISC_STATUS* status_vector,
                                FB_SHUTDOWN_CALLBACK callback_function,
                                const int mask,
                                void* arg);
```

fb_shutdown()

fb_shutdown() performs a smart shutdown of various Firebird subsystems (yValve, engine, redirector). It was primarily designed for use by the internal engine, since it is only applicable to the current process. It is exposed by the API for its possible usefulness to user applications in the embedded server environment.

Currently operational only for the embedded engine, this function terminates all the current activity, rolls back active transactions, disconnects active attachments and shuts down the embedded engine instance gracefully.

> **Important for Application Developers**
>
> **fb_shutdown()** does not perform a shutdown of a remote server to which your application might be concurrently attached. In fact, all of the Firebird client libraries—including the one in embedded—call it automatically at exit(), as long as the client is attached to at least one database or service.
>
> Hence, it should never be called by a client in the context of a remote attachment.

Parameters

fb_shutdown() takes two parameters:

1. timeout in milliseconds
2. reason for shutdown

The reason codes (**const int reason**), which are negative, are listed in `ibase.h`: refer to constants starting with **fb_shutrsn**.

> **Note**
>
> When calling **fb_shutdown()** from your program, you must pass the value as *positive*, for it will be passed as an argument to **fb_shutdown_callback()** by way of your **callback_function**, the routine where you would code the appropriate actions.

Return Values

- A return value of zero means shutdown was successful
- A non-zero value means some errors occurred during the shutdown. Details will be written to `firebird.log`.

fb_shutdown_callback()

fb_shutdown_callback() sets up the callback function that is to be called during shutdown. It is a call that almost always returns successfully, although there are cases, such as an out-of-memory condition, which could cause it to return an error.

Parameters

fb_shutdown_callback() takes four parameters:

status vector (ISC_STATUS status_vector)*
A regular status vector pointer structure.

pointer to callback function (FB_SHUTDOWN_CALLBACK callback_function)
This points to the callback function you have written to perform the actions (if any) to be taken when the callback occurs.

Your callback function can take three parameters. The first and second parameters help to determine what action is to be taken in your callback:

1. reason for shutdown

   Two shutdown reasons are of especial interest:

   - fb_shutrsn_exit_called: Firebird is closing due to exit() or unloaded client/embedded library
   - fb_shutrsn_signal, applies only to POSIX: a SIGINT or SIGTERM signal was caught

2. actual value of the mask with which it was called

   The purpose of this parameter to help determine whether the callback was invoked before or after engine shutdown.

3. argument passed to fb_shutdown_callback() by the user application

   Can be used for any purpose you like and may be NULL.

Return Value from the Callback Function
If the callback function returns zero, it means it performed its job successfully. A non-zero return value is interpreted according to the call mask (see next parameter topic, below):

- For *fb_shut_postproviders* calls, it means some errors occurred and it will result in a non-zero value being returned from **fb_shutdown()**. It is the responsibility of the callback function to notify the world of the exact reasons for the error condition being returned.
- For *fb_shut_preproviders* calls, it means that shutdown will not be performed.

> **Tip**
>
> It is *NOT* a good idea to return non-zero if the shutdown is due to exit() having been called ! ;-)

call mask (const int mask)
Can have the following symbolic values:

- fb_shut_preproviders: callback function will be called before shutting down engine
- fb_shut_postproviders: callback function will be called after shutting down engine
- An ORed combination of them, to have the same function called in either case

argument (void arg)*
This is the argument to be passed to **callback_function**.

Using the fb_shutdown Functions

Following is a sample of using the shutdown and shutdown callback feature to prevent your program from being terminated if someone presses Ctrl-C while it is has database attachments.

```
#include <ibase.h>

// callback function for shutdown
static int ignoreCtrlC(const int reason, const int, void*)
{
 return reason == fb_shutrsn_signal ? 1 : 0;
}

int main(int argc, char *argv[])
{
 ISC_STATUS_ARRAY status;
 if (fb_shutdown_callback(status, ignoreCtrlC, fb_shut_preproviders, 0))
 {
  isc_print_status(status);
  return 1;
 }
 // your code continues ...
}
```

New isc_spb_prp_* Constants for Shutdown

The new database shutdown modes can now be set using calls to the Services API. A number of new **isc_spb_prp_*** constants are available as arguments.

isc_spb_prp_shutdown_mode and isc_spb_prp_online_mode

These arguments are used for shutting down a database and bringing it back on-line, respectively. Each carries a single-byte parameter to set the new shutdown mode, exactly in accord with the *gfix -shut* settings:

- isc_spb_prp_sm_normal
- isc_spb_prp_sm_multi
- isc_spb_prp_sm_single
- isc_spb_prp_sm_full

The shutdown request also requires the *type of shutdown* to be specified, viz., one of

- isc_spb_prp_force_shutdown
- isc_spb_prp_attachments_shutdown
- isc_spb_prp_transactions_shutdown

Each takes a 4-byte integer parameter, specifying the timeout for the shutdown operation requested.

> **Note**
>
> The older-style parameters are also supported and should be used to enter the default shutdown (currently 'multi') and online ('normal') modes.

Usage Examples

Following are a few examples of using the new parameters with the *fbsvmgr* utility. For simplicity, it is assumed that login has already been established. Each example, though broken to fit the page-width, is a single line command.

Shutdown database to single-user maintenance mode:

```
fbsvcmgr service_mgr action_properties dbname employee
  prp_shutdown_mode prp_sm_single prp_force_shutdown 0
```

Next, enable multi-user maintenance:

```
fbsvcmgr service_mgr action_properties dbname employee
  prp_online_mode prp_sm_multi
```

Now go into full shutdown mode, disabling new attachments for the next 60 seconds:

```
fbsvcmgr service_mgr action_properties dbname employee
  prp_shutdown_mode prp_sm_full prp_attachments_shutdown 60
```

Return to normal state:

```
fbsvcmgr service_mgr action_properties dbname employee
  prp_online_mode prp_sm_normal
```

Tighter Control Over Header-level Changes

Alex Peshkov

Several DPB parameters have been made inaccessible to ordinary users, closing some dangerous loopholes. In some cases, they are settings that would alter the database header settings and potentially cause corruptions if not performed under administrator control; in others, they initiate operations that are otherwise restricted to the SYSDBA. They are.-

- isc_dpb_shutdown and isc_dpb_online
- isc_dpb_gbak_attach, isc_dpb_gfix_attach and isc_dpb_gstat_attach
- isc_dpb_verify
- isc_dpb_no_db_triggers
- isc_dpb_set_db_sql_dialect
- isc_dpb_sweep_interval
- isc_dpb_force_write
- isc_dpb_no_reserve
- isc_dpb_set_db_readonly
- isc_dpb_set_page_buffers (on Superserver)

The parameter `isc_dpb_set_page_buffers` can still be used by ordinary users on Classic and 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.

> **Important Note for Developers and Users of Data Access Drivers and Tools**
>
> This change will affect any of the listed DPB parameters that have been explicitly set, either by including them in the DPB implementation by default property values or by enabling them in tools and applications that access databases as ordinary users. For example, a Delphi application that included 'RESERVE PAGE SPACE=TRUE' and 'FORCED WRITES=TRUE' in its database Params property, which caused no problems when the application connected to Firebird 1.x, 2.0.1. 2.0.3, 2.04 or 2.1.0/2.1.1, now rejects 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.”

New Trace Services for Applications

Vlad Khorsun

Five new services relating to the management of the new user trace sessions have been added to the Services Manager, each with its corresponding Services API action function.

isc_action_svc_trace_start

Starts a user trace session

Parameter(s)

```
isc_spb_trc_name : trace session name, string, optional
isc_spb_trc_cfg  : trace session configuration, string, mandatory
```

The mandatory parameter is a string encompassing the text for the desired configuration. A template file named `fbtrace.conf` is provided in Firebird's root directory as a guide to the contents of this string.

> **Note**
>
> 1. Unlike system audit sessions, a user session does not read the configuration from a file. It will be the responsibility of the application developer to devise a mechanism for storing configurations locally at the client and retrieving them for run-time use.
> 2. Superfluous white space in the string is fine: it will simply be ignored.

Output

Results of the trace session in text format.

isc_action_svc_trace_stop

Stops a designated trace session

Parameter(s)

```
isc_spb_trc_id : trace session ID, integer, mandatory
```

Output

A text message providing the result (status) of the request:

- Trace session ID NNN stopped
- No permissions to stop other user trace session
- Trace session ID NNN not found

isc_action_svc_trace_suspend

Suspends a designated trace session

Parameter(s)

```
isc_spb_trc_id : trace session ID, integer, mandatory
```

Output

A text message providing the result (status) of the request:

- Trace session ID NNN paused
- No permissions to change other user trace session
- Trace session ID NNN not found

isc_action_svc_trace_resume

Resumes a designated trace session that has been suspended

Parameter(s)

```
isc_spb_trc_id : trace session ID, integer, mandatory
```

Output

A text message providing the result (status) of the request:

- Trace session ID NNN resumed
- No permissions to change other user trace session
- Trace session ID NNN not found

isc_action_svc_trace_list

Lists existing trace sessions

No parameters

Output

A text message listing the trace sessions and their states:

- Session ID: <number>
- name: <string>. Prints the trace session name if it is not empty
- user: <string>. Prints the user name of the user that created the trace session
- date: YYYY-MM-DD HH:NN:SS, start date and time of the user session
- flags: <string>, a comma-separated set comprising some or all of the following:

  active | *suspend*
  : Run state of the session.

  admin
  : Shows *admin* if an administrator user created the session. Absent if an ordinary user created the session.

  system
  : Shows *system* if the session was created by the Firebird engine (system audit session). Absent if an ordinary user created the session.

  audit | *trace*
  : Indicates the kind of session: *audit* for an engine-created audit session or *trace* for a user trace session.

log full
: Conditional, appears if it is a user trace session and the session log file is full.

> **Note**
>
> The output of each service can usually be obtained using a regular *isc_service_query* call with either of the *isc_info_svc_line* or *isc_info_svc_to_eof* information items.

Other Services API Additions

Alex Peshkov

Other additions to the Services API include:

Mapping for RDB$ADMIN Role in Services API

Two tag items have been added to the services parameter block (SPB) to to enable or disable the RDB$ADMIN role for a privileged operating system user when requesting access to the security database.

> **Note**
>
> This capability is implemented in the *gsec* utility by way of the new **-mapping** switch. Refer to the relevant notes in the relevant section of the *Command-line Utilities* chapter.

Tag Item isc_action_svc_set_mapping

Enables the RDB$ADMIN role for the appointed OS user for a service request to access *security2.fdb*.

Tag Item isc_action_svc_drop_mapping

Disables the RDB$ADMIN role for the appointed OS user for a service request to access *security2.fdb*.

Tag item isc_spb_bkp_no_triggers

This new SPB tag reflects the Services API side of the **-nodbtriggers** switch introduced in the *gbak* utility at V.2.1 to prevent database-level and transaction-level triggers from firing during backup and restore. It is intended for use as a member of the **isc_spb_options** set of optional directives that includes items like **isc_spb_bkp_ignore_limbo**, etc.

nBackup Support

Tracker reference: CORE-1758.

The nBackup utility performs two logical groups of operations: locking or unlocking a database and backing it up or restoring it. While there is no rationale for providing a service action for the lock/unlock operations—they can be requested remotely by way of an SQL language request for ALTER DATABASE—a Services API interface to the backup/restore operations is easily justified.

Backup and restore must be run on the host station and the only way to access them was by running nBackup.

The two new service actions now enabling nBackup backup and restore to be requested through the Services API are:

- isc_action_svc_nbak - incremental nbackup
- isc_action_svc_nrest - incremental database restore

The parameter items are:

- isc_spb_nbk_level - backup level (integer)
- isc_spb_nbk_file - backup file name (string)
- isc_spb_nbk_no_triggers - option to suppress database triggers

Usage Examples

Following are a few examples of using the new parameters with the *fbsvcmgr* utility. For simplicity, it is assumed that login has already been established. Each example, though broken to fit the page-width, is a single line command.

Create backup level 0:

```
fbsvcmgr service_mgr action_nbak dbname employee
  nbk_file e.nb0 nbk_level 0
```

Create backup level 1:

```
fbsvcmgr service_mgr action_nbak dbname employee
  nbk_file e.nb1 nbk_level 1
```

Restore database from those files:

```
fbsvcmgr service_mgr action_nrest dbname e.fdb
  nbk_file e.nb0 nbk_file e.nb1
```

Chapter 5

New Reserved Words and Changes

The following keywords have been added, or have changed status, since Firebird 2.1. Those marked with an asterisk (*) are not present in the SQL standard.

Newly Reserved Words

```
SIMILAR
```

Keywords Added as Non-reserved

```
AUTONOMOUS *
BIN_NOT *
CALLER *
CHAR_TO_UUID *
COMMON *
DATA
FIRSTNAME *
GRANTED
LASTNAME *
MIDDLENAME *
MAPPING *
OS_NAME *
SOURCE *
TWO_PHASE *
UUID_TO_CHAR *
```

Chapter 6

Configuration Parameter Additions and Changes

The following changes or additions to `firebird.conf` should be noted:

AuditTraceConfigFile

V. Khorsun

This parameter points to the name and location of the file that the Firebird engine is to read to determine the list of events required for the next system audit trace. By default, the value of this parameter is empty, indicating that no system audit tracing is configured.

> **Note**
>
> The template file file `fbtrace.conf`, found in Firebird's root directory, contains the full list of available events, with format, rules and syntax for composing an audit trace configuration file.

For more information, see the topic System Audit Session in section [Trace and Audit Services] in the chapter about the new administrative features.

Authentication

A. Peshkov

On Windows server platforms, since V.2.1, *Authentication* has been used for configuring the server authentication mode if you need it to be other than the default.

The mode settings for v.2.5 are the same, viz.

- *trusted* makes use of Windows “trusted authentication”. Under the right conditions, this may be the most secure way to authenticate on Windows.
- *native* sets the traditional Firebird server authentication mode, requiring users to log in using a user name and password defined in the security database.
- *mixed* allows both.

Changes in V.2.5

- Under v.2.5, although the modes are unchanged, configuring 'mixed' or 'trusted' mode no longer confers SYSDBA privileges on Windows domain administrators automatically by default. Please read the notes in the Administrative Features chapter regarding the new RDB$ADMIN role in ODS 11.2 databases and auto-mapping SYSDBA privileges to domain administrators.

- The default configuration has been changed from *mixed* to *native*. To have enable user authentication (whether *mixed* or *trusted*, it is now necessary to configure this parameter specifically.

  Tracker reference CORE-2376)

MaxUserTraceLogSize

V. Khorsun

Stores the maximum total size of the temporary files to be created by a user trace session using the new Trace functions in the Services API. The default limit is 10 MB. Use this parameter to raise or lower the maximum total size of the temporary files storing the output.

OldSetClauseSemantics

D. Yemanov

Before Firebird 2.5, the SET clause of the UPDATE statement assigned columns in the user-defined order, with the NEW column values being immediately accessible to the subsequent assignments. This did not conform to the SQL standard, which requires the starting value of the column to persist during execution of the statement.

Now, only the OLD column values are accessible to any assignment in the SET clause.

The *OldSetClauseSemantics* enables you to revert to the legacy behavior via the OldSetClauseSemantics, if required. Values are 1 for the legacy behaviour, 0 (the default) for the corrected behaviour.

> **Warning**
>
> - Changing this parameter affects *all* databases on your server.
> - This parameter is provided as a temporary solution to resolve backward compatibility issues. It will be deprecated in future Firebird versions.

Use Hostname for RemoteBindAddress

A. Peshkov

Tracker entry: CORE-2094)

It is now possible to use the hostname of the host where the Firebird server is running to configure *RemoteBindAddress*, where previously, only an IP address was allowed.

> **Important**
>
> *RemoteBindAddress* can be used to “pin” user connections to a specific NIC card on the host server. Take care that the hostname specified is not associated concurrently with more than one IP address, anywhere! In particular, check the `etc/hosts` file on all stations, including the host station itself.

Chapter 7

Administrative Features

Certain improvements to Firebird's administrative features will be welcomed by many.

New RDB$ADMIN System Role

Alex Peshkov

A new pre-defined system role RDB$ADMIN has been added for transferring SYSDBA privileges to another user. Any user, when granted the role in a particular database, acquires SYSDBA-like rights when attaching to that database with the RDB$ADMIN role specified.

To assign it, SYSDBA should log in to that database and grant the role RDB$ADMIN to the user, in the same way one would grant any other role to a user.

The following example transfers SYSDBA privileges to users named User1 and Admins\ADMINS. The second user in our example is typical for a Windows system user with access via trusted authentication:

```
GRANT RDB$ADMIN TO User1;
GRANT RDB$ADMIN TO "Admins\ADMINS";
```

> **Note**
>
> For Windows trusted authentication, a database can be set up to provide the RDB$ADMIN role to Windows Administrators automatically. This is described in more detail presently.

Windows Domain Administrators

On POSIX hosts, the *root* user always had SYSDBA privileges, but the same was not possible for a domain administrator on Windows until Firebird 2.1. There, a configuration parameter, *Authentication*, was introduced whereby a user logged in as a Windows domain administrator could automatically gain server access with SYSDBA privileges through trusted user authentication. The mechanism for achieving that has changed with the introduction of the new system role and associated behaviour in v.2.5.

Automatically Mapping RDB$ADMIN to a Windows User

The situation has not changed for the *root* user on POSIX but, on Windows, a domain administrator must now be granted the RDB$ADMIN role in order to get SYSDBA access. By default, the SYSDBA must perform this GRANT manually for any user, including a domain administrator. However, the SYSDBA can configure it to happen automatically for Windows Administrators if the *Authentication* parameter in firebird.conf is 'mixed' or 'trusted'.

The default for the Authentication parameter has changed!

By default, under V.2.5, Windows trusted authentication is not available. It must be specifically enabled in `firebird.conf`, by configuring the *Authentication* parameter as either *trusted* or *mixed*.

New ALTER ROLE Statement

A new ALTER ROLE syntax is is implemented for this specialised purpose.

To configure a database to auto-grant the RDB$ADMIN role to Administrators, use the following statement:

```
ALTER ROLE RDB$ADMIN
  SET AUTO ADMIN MAPPING;
```

To revert to the default setting, preventing administrators from getting SYSDBA privileges automatically, issue this statement:

```
ALTER ROLE RDB$ADMIN
  DROP AUTO ADMIN MAPPING;
```

Escalating RDB$ADMIN Scope

Because *security2.fdb* is created as (or should be upgraded to) an ODS 11.2 database, it has the pre-defined RDB $ADMIN role, too. SYSDBA can grant RDB$ADMIN in *security2.fdb* to a user if that user needs the same rights as SYSDBA to administer all other users through *gsec* or the Services API, i.e., create and drop users or alter any user.

The auto-mapping facility described above is also applicable, if required.

Important

If the user attaches with a user database role passed in the DPB (connection parameters), it will not be replaced with RDB$ADMIN, i.e., he/she will not get SYSDBA rights.

Trace and Audit Services

Vlad Khorsun

The new trace and audit facilities in v.2.5 were initially developed from the TraceAPI contributed to us by Nickolay Samofatov that he had developed for the Red Soft Database, a commercial product based on Firebird's code.

Overview of Features

The new trace and audit facilities enable various events performed inside the engine, such as statement execution, connections, disconnections, etc., to be logged and collated for real-time analysis of the corresponding performance characteristics.

A trace takes place in the context of a *trace session*. Each trace session has its own configuration, state and output.

The Firebird engine has a fixed list of events it can trace. It can perform two different sort of traces: a *system audit* and a *user trace*. How the engine forms the list of events for a session depends on which sort of trace is requested.

The System Audit Session

A system audit session is started by the engine itself. To determine which events the session is "interested in", it reads the contents of a *trace configuration file* as it goes to create the session.

A new parameter in `firebird.conf`, *AuditTraceConfigFile* points to the name and location of the file. There can be at most one system audit trace in progress. By default, the value of this parameter is empty, indicating that no system audit tracing is configured.

A configuration file contains list of traced events and points to the placement of the trace log(s) for each event. It is sufficiently flexible to allow different sets of events for different databases to be logged to separate log files. The template file file `fbtrace.conf`, found in Firebird's root directory, contains the full list of available events, with format, rules and syntax for composing an audit trace configuration file.

User Trace Sessions

A user trace session is managed by user, using some new calls to the Services API. There are five new service functions for this purpose:

- start: **isc_action_svc_trace_start**
- stop: **isc_action_svc_trace_stop**
- suspend: **isc_action_svc_trace_suspend**
- resume: **isc_action_svc_trace_resume**
- list all known trace sessions: **isc_action_svc_trace_list**

The syntax for the Services API calls are discussed in the topic New Trace Functions for Applications in the chapter entitled *Changes to the Firebird API and ODS*.

Workings of a User Trace Session

When a user application starts a trace session, it sets a session name (optional) and the session configuration (mandatory). The session configuration is a text file conforming to the rules and syntax modelled in the `fbtrace.conf` template that is in Firebird's root directory, apart from the lines relating to placement of the output.

> **Note**
>
> Such files obviously do not live on the server. It will be the job of the application developer to design a suitable mechanism for storing and retrieving texts for passing in the user trace request.
>
> For example, the command-line fbsvcmgr utility supports a saved-file parameter, *trc_cfg*.

The output of a user session is stored in set of temporary files, each of 1 MB. Once a file has been completely read by the application, it is automatically deleted. By default, the maximum total size of the output is limited to 10 MB. It can be changed to a smaller or larger value using the *MaxUserTraceLogSize* in `firebird.conf`.

Once the user trace session service has been started by the application, the application has to read its output, using calls to *isc_service_query()*. The service could be generating output faster than the application can read it. If the total size of the output reaches the *MaxUserTraceLogSize* limit, the engine automatically suspends the trace session. Once the application has finished reading a file (a 1 MB part of the output) that file is deleted, capacity is returned and the engine resumes the trace session automatically.

At the point where the application decides to stop its trace session, it simply requests a *detach* from the service. Alternatively, the application can use the **isc_action_svc_trace_*** functions to suspend, resume or stop the trace session at will.

Who Can Manage Trace Sessions?

Any user can initiate and manage a trace session. An ordinary user can request a trace only on its own connections and cannot manage trace sessions started by any other users. Administrators can manage any trace session.

Abnormal Endings

If all Firebird processes are stopped, no user trace sessions are preserved. What this means is that if a Superserver or Superclassic process is shut down, any user trace sessions that were in progress, including any that were awaiting a *resume* condition, are fully stopped and a *resume* cannot restart them.

> **Note**
>
> This situation doesn't apply to the Classic server, of course, since each connection involves its own dedicated server instance. Thus, there is no such thing as “shutting down” a Classic server instance. No service instance can outlive the connection that instigated it.

User Trace Sample Configuration Texts

The following samples provide a reference for composing configuration texts for user trace sessions.

a. Trace prepare, free and execution of all statements within connection 12345

```
<database mydatabase.fdb>
 enabled                true
 connection_id          12345
 log_statement_prepare  true
 log_statement_free     true
 log_statement_start    true
 log_statement_finish   true
 time_threshold         0
</database>
```

b. Trace all connections of given user to database mydatabase.fdb, logging executed INSERT, UPDATE and DELETE statements and nested calls to procedures and triggers, and show corresponding PLANs and performance statistics.

```
<database mydatabase.fdb>
 enabled                true
 include_filter         %(INSERT|UPDATE|DELETE)%
```

```
 log_statement_finish   true
 log_procedure_finish   true
 log_trigger_finish     true
 print_plan             true
 print_perf             true
 time_threshold         0
</database>
```

Command-line Requests for User Trace Services

At this stage there is no specialized, stand-alone utility to work with trace services. However, the general Services utility, *fbsvcmgr*, is available for submitting service requests from the command line, as exemplified in the following samples.

a. Start a user trace named “My trace” using a configuration file named `fbtrace.conf` and read its output on the screen:

```
fbsvcmgr service_mgr action_trace_start trc_name "My trace" trc_cfg fbtrace.conf
```

To stop this trace session, press Ctrl+C at the *fbsvcmgr* console prompt. (See also (e), below).

b. List trace sessions:

```
fbsvcmgr service_mgr action_trace_list
```

c. Suspend trace sesson with ID 1

```
fbsvcmgr service_mgr action_trace_suspend trc_id 1
```

d. Resume trace sesson with ID 1

```
fbsvcmgr service_mgr action_trace_resume trc_id 1
```

e. Stop trace sesson with ID 1

```
fbsvcmgr service_mgr action_trace_stop trc_id 1
```

> **Tip**
>
> List sessions (see b.) in another console, look for the ID of a session of interest and use it in the current console to stop the session.

Use Cases

There are three general use cases:

1. **Constant audit of engine activity**

   This is served by system audit trace. Administrator creates or edits the trace configuration file, sets its name via the *AuditTraceConfigFile* setting in firebird.conf and restarts Firebird. Later, the administrator could suspend, resume or stop this session without needing to restart Firebird.

   > **Important**
   >
   > To make audit configuration changes known to the engine, Firebird must be restarted.

2. **On-demand interactive trace of some (or all) activity in some (or all) databases**

   An application (which could be the *fbsvcmgr* utility) starts user trace session, reads its output and shows traced events to the user in real time on the screen. The user can suspend and resume the trace and, finally, stop it.

3. **Engine activity collection for a significant period of time (a few hours or perhaps even a whole day) for later analysis**

   An application starts a user trace session, reading the trace output regularly and saving it to one or more files. The session must be stopped manually, by the same application or by another one. If multiple trace sessions are running, a listing can be requested in order to identify the session of interest.

Monitoring Improvements

Dmitry Yemanov

Firebird 2.5 sees the enhancement of the “MON$” database monitoring features introduced in V.2.1, with new tables delivering data about context variables and memory usage in ODS 11.2 and higher databases. Also, in these databases, it becomes possible to terminate a client connection from another connection through the MON $ structures.

Extended Access for Ordinary Users

The original design allowed non-privileged database users to see monitoring information pertaining only to their CURRENT_CONNECTION. Now they can request information for any attachment that was authenticated using the same user name.

Tracker reference CORE-2233.

> **Notes**
>
> 1. For an application architecture that entails a middleware tier logging in multiple times concurrently with the same user name on behalf of different end users, consideration should be given to the impact on performance and privacy of exposing the monitoring features to the end users.
> 2. The same extension was implemented in V.2.1.2.

New MON$ Metadata for ODS 11.2 Databases

> **Note**
>
> Please refer to the V.2.1 documentation for the ODS 11.1 metadata.

MON$MEMORY_USAGE (current memory usage)

- MON$STAT_ID (statistics ID)
- MON$STAT_GROUP (statistics group)
  0: database
  1: attachment
  2: transaction
  3: statement
  4: call
- MON$MEMORY_USED (number of bytes currently in use)
  High-level memory allocations performed by the engine from its pools.
  Can be useful for tracing memory leaks and for investigating unusual
  memory consumption and the attachments, procedures, etc. that might
  be responsible for it.
- MON$MEMORY_ALLOCATED (number of bytes currently allocated at the OS level)
  Low-level memory allocations performed by the Firebird memory manager.
  These are bytes actually allocated by the operating system, so it enables
  the physical memory consumption to be monitored.

> **Note**
>
> Not all records have non-zero values. On the whole, only MON$DATABASE and memory-bound objects point to non-zero “allocated” values. Small allocations are not allocated at this level, being redirected to the database memory pool instead.

- MON$MAX_MEMORY_USED (maximum number of bytes used by this object)
- MON$MAX_MEMORY_ALLOCATED (maximum number of bytes allocated from
  the operating system by this object)

MON$CONTEXT_VARIABLES (known context variables)

- MON$ATTACHMENT_ID (attachment ID)
  Contains a valid ID only for session-level context variables.
  Transaction-level variables have this field set to NULL.
- MON$TRANSACTION_ID (transaction ID)
  Contains a valid ID only for transaction-level context variables.
  Session-level variables have this field set to NULL.
- MON$VARIABLE_NAME (name of context variable)
- MON$VARIABLE_VALUE (value of context variable)

Memory Usage in MON$STATEMENTS and MON$STATE

Memory usage statistics in MON$STATEMENTS and MON$STATE represent actual CPU consumption.

Tracker reference: CORE-1583)

Usage Notes

Examples

“Top 10” statements ranked according to their memory usage:

```
SELECT FIRST 10
  STMT.MON$ATTACHMENT_ID,
  STMT.MON$SQL_TEXT,
  MEM.MON$MEMORY_USED
FROM MON$MEMORY_USAGE MEM
  NATURAL JOIN MON$STATEMENTS STMT
  ORDER BY MEM.MON$MEMORY_USED DESC
```

To enumerate all session-level context variables for the current connection:

```
SELECT
  VAR.MON$VARIABLE_NAME,
  VAR.MON$VARIABLE_VALUE
FROM MON$CONTEXT_VARIABLES VAR
  WHERE VAR.MON$ATTACHMENT_ID = CURRENT_CONNECTION
```

Terminating a Client

The MON$ structures are, by design, read-only. Thus, user DML operations on them are prohibited. However, a mechanism is built in to allow deleting (only) of records in the MON$STATEMENTS and MON$ATTACHMENTS tables. The effect of this mechanism is to make it possible, respectively, to cancel running statements and, for ODS 11.2 databases, to terminate client sessions.

To cancel all current activity for a specified connection:

```
DELETE FROM MON$STATEMENTS
  WHERE MON$ATTACHMENT_ID = 32
```

To disconnect all clients except the “Me” connection:

```
DELETE FROM MON$ATTACHMENTS
  WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

Note

1. • A statement cancellation attempt becomes a void operation (“no-op”) if the client has no statements currently running.
   • Upon cancellation, the execute/fetch API call returns the *isc_cancelled* error code.
   • Subsequent operations are allowed.
2. • Any active transactions in the connection being terminated will have their activities cancelled immediately and they are rolled back.
   • Once terminated, the client session receives the *isc_att_shutdown* error code.
   • Subsequent attempts to use this connection handle will cause network read/write errors.

Chapter 8

Security Hardening

Windows Platforms

No SYSDBA Auto-mapping (Windows)

In V.2.1, members of administrative Windows groups were mapped to SYSDBA by default. From V.2.5 forward, SYSDBA mapping is controlled on per-database basis using the new SQL command

ALTER ROLE RDB$ADMIN SET/DROP AUTO ADMIN MAPPING

Chapter 9

Data Definition Language (DDL)

V.2.5 brings a few significant additions and enhancements to DDL.

Quick Links

Although this release emphasises architectural changes in the movement towards Firebird 3, a number of improvements and extensions have been implemented, in many cases as a response to feature requests in the Tracker.

Visibility of Procedure Definition Changes on Classic

Dmitry Yemanov

Tracker reference CORE-2052.

One such change addressed the problem of the visibility of altered stored procedures to other connections to the Classic server. Now, such changes are made visible to the entire server as soon as the modifying transaction has completed its commit.

CREATE/ALTER/DROP USER

Alex Peshkov

Tracker reference CORE-696.

In v.2.5, Firebird finally has syntax to enable user accounts on the server to be managed by submitting SQL statements when logged in to a regular database.

Syntax Patterns

A user with SYSDBA privileges can add a new user:

```
CREATE USER <username> {PASSWORD 'password'}
  [FIRSTNAME 'firstname']
```

```
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname'];
```

> **Note**
>
> The PASSWORD clause is required when creating a new user. It should be the initial password for that new user. (The user can change it later, using ALTER USER.)

A user with SYSDBA privileges can change one or more of the password and proper name attributes of an existing user. Non-privileged users can use this statement to alter only their own attributes.

```
ALTER USER <username>
  [PASSWORD 'password']
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname'];
```

> **Note**
>
> At least one of PASSWORD, FIRSTNAME, MIDDLENAME or LASTNAME must be present.
>
> ALTER USER does not allow the <username> to be changed. If a different <username> is required, the old one should be deleted (dropped) and a new one created.

A user with SYSDBA privileges can delete a user:

```
DROP USER <username>;
```

Restrictions

CREATE and DROP statements are available only for the SYSDBA or a user that has been granted the RDB $ADMIN role in the security database. An ordinary user can ALTER his own password and elements of his proper name. An attempt to modify another user will fail.

Examples

```
CREATE USER alex PASSWORD 'test';

ALTER USER alex FIRSTNAME 'Alex' LASTNAME 'Peshkov';

ALTER USER alex PASSWORD 'IdQfA';

DROP USER alex;
```

> **Tip**
>
> Firebird 2.5 does not allow you to set up more than one security database on a server. From V.3.0, it is intended to be possible to have separate security databases for each database. For now, you can be connected to any database on the server (even *employee.fdb*) to update its one-and-only *security2.fdb*.
>
> In future, it will be essential to send these requests from a database that is associated with the security database that is to be affected by them.

Syntaxes for Altering Views

Adriano dos Santos Fernandes

Previously, in order to alter a view definition, it was necessary to save the view definition off-line somewhere and drop the view, before recreating it with its changes. This made things very cumbersome, especially if there were dependencies. V.2.5 introduces syntaxes for ALTER VIEW and CREATE OR ALTER VIEW.

Tracker references are CORE-770 and CORE-1640.

ALTER VIEW

ALTER VIEW enables a view definition to be altered without the need to recreate (drop and create) the old version of the view and all of its dependencies.

CREATE OR ALTER VIEW

With CREATE OR ALTER VIEW, the view definition will be altered (as with ALTER VIEW) if it exists, or created if it does not exist.

Syntax Pattern

```
create [ or alter ] | alter } view <view_name>
  [ ( <field list> ) ]
as <select statement>
```

Example

```
create table users (
    id integer,
    name varchar(20),
    passwd varchar(20)
);

create view v_users as
    select name from users;

alter view v_users (id, name) as
    select id, name from users;
```

ATTENTION: Known Issue

It was discovered that ALTER VIEW (and probably CREATE OR ALTER VIEW) is able to remove a column used in a stored procedure or trigger, without raising a dependency exception at compile time.

This bug (CORE-2386) will be addressed before the next Beta.

Extension for CREATE VIEW

Adriano dos Santos Fernandes

Tracker reference CORE-886.

A selectable stored procedure can now be specified in the FROM clause of a view definition.

Example

```
create view a_view as
select * from a_procedure(current_date);
```

ALTER Mechanism for Computed Columns

Adriano dos Santos Fernandes

Tracker reference CORE-1454.

A column defined as COMPUTED BY <expression> can now be altered using the ALTER TABLE...ALTER COLUMN syntax. This feature can be used only to change the <expression> element of the column definition to a different expression. It cannot convert a computed column to non-computed or vice versa.

Syntax Pattern

```
alter table <table-name>
  alter <computed-column-name>
  [type <data-type>]
  COMPUTED BY (<expression>);
```

Examples

```
create table test (
  n integer,
  dn computed by (n * 2)
);
commit;
alter table test
  alter dn computed by (n + n);
```

Extensions for SQL Permissions

Alex Peshkov

The following extensions have been implemented in the area of SQL permissions (privileges).

GRANTED BY Clause

A GRANTED BY or GRANTED AS clause can now be optionally included in GRANT and REVOKE statements, enabling the grantor to be a user other than the CURRENT_USER (the default).

Syntax Pattern

```
grant <right> to <object>
  [ { granted by | as } [ user ] <username> ]
--
revoke <right> from <object>
  [ { granted by | as } [ user ] <username> ]
```

> **Tip**
>
> GRANTED BY and GRANTED AS are equivalent. GRANTED BY is the form recommended by the SQL standard. We support GRANTED AS for compatibility with some other servers (Informix, for example).

Example

Logged in as SYSDBA:

```
create role r1; -- SYSDBA owns the role
  /* next, SYSDBA grants the role to user1
     with the power to grant it to others */
grant r1 to user1 with admin option;
  /* SYSDBA uses GRANTED BY to exercise
     user1's ADMIN OPTION */
grant r1 to public granted by user1;
```

In isql, we look at the effects:

```
SQL>show grant;
/* Grant permissions for this database */
GRANT R1 TO PUBLIC GRANTED BY USER1
GRANT R1 TO USER1 WITH ADMIN OPTION
SQL>
```

ALTER ROLE

Tracker reference CORE-1660.

The new ALTER ROLE statement has a specialised function to control the assignment of SYSDBA permissions to Windows administrators during trusted authentication. It has no other purpose currently.

For usage details, see the topic in the Administrative Features chapter, entitled Automatically Mapping RDB $ADMIN to a Windows User.

REVOKE ALL

Tracker reference CORE-2113.

When a user is removed from the security database or another authentication source, such as the operating system ACL, any associated cleanup of SQL privileges in databases has to be performed manually. This extension adds the capability to revoke all privileges in one stroke from a particular user or role.

Syntax Pattern

```
REVOKE ALL ON ALL FROM { <user list> | <role list> }
```

Example

Logged in as SYSDBA:

```
# gsec -del guest
# isql employee
fbs bin # ./isql employee
Database:  employee
SQL> REVOKE ALL ON ALL FROM USER guest;
SQL>
```

Default COLLATION Attribute for a Database

Adriano dos Santos Fernandes

Tracker references CORE-1737 and CORE-1803.

An ODS 11.2 or higher database can now have a default COLLATION attribute associated with the default character set, enabling all text column, domain and variable definitions to be created with the same collation order unless a COLLATE clause for a different collation is specified.

The COLLATION clause is optional. If it is omitted, the default collation order for the character set is used.

> **Tip**
>
> Note also that the default collation order for a character set used in a database can now also be changed, thanks to the introduction of syntax for ALTER CHARACTER SET.

Syntax Pattern

```
create database <file name>
    [ page_size <page size> ]
    [ length = <length> ]
    [ user <user name> ]
    [ password <user password> ]
    [ set names <charset name> ]
    [ default character set <charset name>
      [ collation <collation name> ] ]
    [ difference file <file name> ]
```

Example

```
create database 'test.fdb'
    default character set win1252
    collation win_ptbr;
```

ALTER CHARACTER SET Command

Adriano dos Santos Fernandes

Tracker reference CORE-1803.

New syntax introduced in this version, enabling the default collation for a character set to be set for a database.

The default collation is used when table columns are created with a given character set (explicitly, through a CHARACTER SET clause in the column or domain definition, or implicitly, through the default character set attribute of the database) and no COLLATION clause is specified.

> **Note**
>
> String constants also use the default collation of the connection character set.

Syntax Pattern

```
ALTER CHARACTER SET <charset_name>
    SET DEFAULT COLLATION <collation_name>
```

Example

```
create database 'people.fdb'
  default character set win1252;

alter character set win1252
  set default collation win_ptbr;

create table person (
  id integer,
  name varchar(50)  /* will use the database default
                       character set and the win1252
                       default collation */
);

insert into person
  values (1, 'adriano');
insert into person
  values (2, 'ADRIANO');

/* will retrieve both records
   because win_ptbr is case insensitive */
select * from person where name like 'A%';
```

> **Tip**
>
> Another improvement allows the current value of RDB$DEFAULT_COLLATE_NAME in the system table RDB$CHARACTER_SETS to survive the backup/restore cycle.

Chapter 10

Data Manipulation Language (DML)

In this chapter are the additions and improvements that have been added to the SQL data manipulation language subset in Firebird 2.5.

Quick Links

RegEx Search Support using SIMILAR TO

Adriano dos Santos Fernandes

Tracker reference CORE-769.

A new SIMILAR TO predicate is introduced to support regular expressions. The predicate's function is to verify whether a given SQL-standard regular expression matches a string argument. It is valid in any context that accepts Boolean expressions, such as WHERE clauses, CHECK constraints and PSQL IF() tests.

Syntax Patterns

```
<similar predicate> ::=
 <value> [ NOT ] SIMILAR TO <similar pattern> [ ESCAPE <escape character> ]

<similar pattern> ::= <character value expression: regular expression>

<regular expression> ::=
      <regular term>
    | <regular expression> <vertical bar> <regular term>

<regular term> ::=
      <regular factor>
    | <regular term> <regular factor>

<regular factor> ::=
      <regular primary>
    | <regular primary> <asterisk>
    | <regular primary> <plus sign>
    | <regular primary> <question mark>
```

```
    | <regular primary> <repeat factor>

<repeat factor> ::=
    <left brace> <low value> [ <upper limit> ] <right brace>

<upper limit> ::= <comma> [ <high value> ]

<low value> ::= <unsigned integer>

<high value> ::= <unsigned integer>

<regular primary> ::=
      <character specifier>
    | <percent>
    | <regular character set>
    | <left paren> <regular expression> <right paren>

<character specifier> ::=
      <non-escaped character>
    | <escaped character>

<regular character set> ::=
      <underscore>
    | <left bracket> <character enumeration>... <right bracket>
    | <left bracket> <circumflex> <character enumeration>... <right bracket>
    | <left bracket> <character enumeration include>... <circumflex> <character enumeration excl
        <right bracket>

<character enumeration include> ::= <character enumeration>

<character enumeration exclude> ::= <character enumeration>

<character enumeration> ::=
      <character specifier>
    | <character specifier> <minus sign> <character specifier>
    | <left bracket> <colon> <character class identifier> <colon> <right bracket>

<character specifier> ::=
      <non-escaped character>
    | <escaped character>

<character class identifier> ::=
      ALPHA
    | UPPER
    | LOWER
    | DIGIT
    | SPACE
    | WHITESPACE
    | ALNUM
```

Note

1. <non-escaped character> is any character except <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore>, <percent>, <question mark>, <left brace> and <escape character>.

2. <escaped character> is the <escape character> succeeded by one of <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore>, <percent>, <question mark>, <left brace> or <escape character>.

Table 10.1. Character class identifiers

Identifier	Description	Note
ALPHA	All characters that are simple latin letters (a-z, A-Z)	Includes latin letters with accents when using accent-insensitive collation
UPPER	All characters that are simple latin upper-case letters (A-Z)	Includes lowercase latters when using case-insensitive collation
LOWER	All characters that are simple latin lower-case letters (a-z)	Includes uppercase latters when using case-insensitive collation
DIGIT	All characters that are numeric digits (0-9)	
SPACE	All characters that are the space character (ASCII 32)	
WHITESPACE	All characters that are whitespaces (vertical tab (9), newline (10), horizontal tab (11), carriage return (13), formfeed (12), space (32))	
ALNUM	All characters that are simple latin letters (ALPHA) or numeric digits (DIGIT)	

Usage Guide

Return true for a string that matches <regular expression> or <regular term>:

```
  <regular expression> <vertical bar> <regular term>
```

```
'ab' SIMILAR TO 'ab|cd|efg'   -- true
'efg' SIMILAR TO 'ab|cd|efg'  -- true
'a' SIMILAR TO 'ab|cd|efg'    -- false
```

Match zero or more occurrences of <regular primary>: <regular primary> <asterisk>

```
'' SIMILAR TO 'a*'     -- true
'a' SIMILAR TO 'a*'    -- true
'aaa' SIMILAR TO 'a*'  -- true
```

Match one or more occurrences of <regular primary>: <regular primary> <plus sign>

```
'' SIMILAR TO 'a+'     -- false
'a' SIMILAR TO 'a+'    -- true
'aaa' SIMILAR TO 'a+'  -- true
```

Match zero or one occurrence of <regular primary>: <regular primary> <question mark>

```
'' SIMILAR TO 'a?'     -- true
'a' SIMILAR TO 'a?'    -- true
'aaa' SIMILAR TO 'a?'  -- false
```

Match exact <low value> occurrences of <regular primary>: <regular primary> <left brace> <low value> <right brace>

```
'' SIMILAR TO 'a{2}'     -- false
'a' SIMILAR TO 'a{2}'    -- false
'aa' SIMILAR TO 'a{2}'   -- true
'aaa' SIMILAR TO 'a{2}'  -- false
```

Match <low value> or more occurrences of <regular primary>: <regular primary> <left brace> <low value> <comma> <right brace>:

```
'' SIMILAR TO 'a{2,}'     -- false
'a' SIMILAR TO 'a{2,}'    -- false
'aa' SIMILAR TO 'a{2,}'   -- true
'aaa' SIMILAR TO 'a{2,}'  -- true
```

Match <low value> to <high value> occurrences of <regular primary> <regular primary> <left brace> <low value> <comma> <high value> <right brace>:

```
'' SIMILAR TO 'a{2,4}'       -- false
'a' SIMILAR TO 'a{2,4}'      -- false
'aa' SIMILAR TO 'a{2,4}'     -- true
'aaa' SIMILAR TO 'a{2,4}'    -- true
'aaaa' SIMILAR TO 'a{2,4}'   -- true
'aaaaa' SIMILAR TO 'a{2,4}'  -- false
```

Match any (non-empty) character: <underscore>

```
'' SIMILAR TO '_'    -- false
'a' SIMILAR TO '_'   -- true
'1' SIMILAR TO '_'   -- true
'a1' SIMILAR TO '_'  -- false
```

Match a string of any length (including empty strings): <percent>

```
'' SIMILAR TO '%'         -- true
'az' SIMILAR TO 'a%z'     -- true
'a123z' SIMILAR TO 'a%z'  -- true
'azx' SIMILAR TO 'a%z'    -- false
```

Group a complete <regular expression> to use as one single <regular primary> as a sub-expression: <left paren> <regular expression> <right paren>

```
'ab' SIMILAR TO '(ab){2}'    -- false
```

```
    'aabb' SIMILAR TO '(ab){2}'  -- false
    'abab' SIMILAR TO '(ab){2}'  -- true
```

Match a character identical to one of <character enumeration>: <left bracket> <character enumeration>... <right bracket>

```
    'b' SIMILAR TO '[abc]'    -- true
    'd' SIMILAR TO '[abc]'    -- false
    '9' SIMILAR TO '[0-9]'    -- true
    '9' SIMILAR TO '[0-8]'    -- false
```

Match a character not identical to one of <character enumeration>: <left bracket> <circumflex> <character enumeration>... <right bracket>

```
    'b' SIMILAR TO '[^abc]'    -- false
    'd' SIMILAR TO '[^abc]'    -- true
```

Match a character identical to one of <character enumeration include> but not identical to one of <character enumeration exclude>: <left bracket> <character enumeration include>... <circumflex> <character enumeration exclude>...

```
    '3' SIMILAR TO '[[:DIGIT:]^3]'    -- false
    '4' SIMILAR TO '[[:DIGIT:]^3]'    -- true
```

Match a character identical to one character included in <character class identifier>. Refer to the table of Character Class Identifiers, above. May be used with <circumflex> to invert the logic (see above): <left bracket> <colon> <character class identifier> <colon> <right bracket>

```
    '4' SIMILAR TO '[[:DIGIT:]]'  -- true
    'a' SIMILAR TO '[[:DIGIT:]]'  -- false
    '4' SIMILAR TO '[^[:DIGIT:]]' -- false
    'a' SIMILAR TO '[^[:DIGIT:]]' -- true
```

Examples

```
create table department (
  number numeric(3) not null,
  name varchar(25) not null,
  phone varchar(14)
  check (phone similar to '\([0-9]{3}\) [0-9]{3}\-[0-9]{4}' escape '\')
);

insert into department
  values ('000', 'Corporate Headquarters', '(408) 555-1234');
insert into department
  values ('100', 'Sales and Marketing', '(415) 555-1234');
insert into department
  values ('140', 'Field Office: Canada', '(416) 677-1000');

insert into department
  values ('600', 'Engineering', '(408) 555-123'); -- check constraint violation
```

```
select * from department
  where phone not similar to '\([0-9]{3}\) 555\-%' escape '\';
```

Hex Literal Support

Bill Oliver
Adriano dos Santos Fernandes

Tracker reference CORE-1760.

Support for hexadecimal numeric and binary string literals has been introduced.

Syntax Patterns

```
<numeric hex literal> ::=
    { 0x | 0X } <hexit> [ <hexit>... ]

<binary string literal> ::=
    { x | X } <quote> [ { <hexit> <hexit> }... ] <quote>

<digit> ::=
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<hexit> ::=
    <digit> | A | B | C | D | E | F | a | b | c | d | e | f
```

Numeric Hex Literals

- The number of <hexit> in the string cannot exceed 16.
- If the number of <hexit> is greater than eight, the constant data type is a signed BIGINT. If it is eight or less, the data type is a signed INTEGER.

> **Tip**
>
> That means 0xF0000000 is -268435456 and 0x0F0000000 is 4026531840.

Binary String Literals

The resulting string is defined as a CHAR(*n*/2) CHARACTER SET OCTETS, where *n* is the number of <hexit>.

Examples

```
select 0x10, cast('0x0F0000000' as bigint)
  from rdb$database;
select x'deadbeef'
  from rdb$database;
```

New UUID Conversion Functions

Adriano dos Santos Fernandes

Tracker references CORE-1656 and CORE-1682.

Two new built-in functions, UUID_TO_CHAR and CHAR_TO_UUID, enable conversion between a UUID in the form of a CHAR(16) OCTETS string and the RFC4122-compliant form.

CHAR_TO_UUID()

The function CHAR_TO_UUID() converts the CHAR(32) ASCII representation of a UUID (XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX) to the CHAR(16) OCTETS representation, optimized for storage.

Syntax Model

```
CHAR_TO_UUID( <string> )
```

Example

```
select char_to_uuid('93519227-8D50-4E47-81AA-8F6678C096A1')
  from rdb$database;
```

UUID_TO_CHAR()

The function UUID_TO_CHAR() converts a CHAR(16) OCTETS UUID (as returned by the GEN_UUID() function) to the CHAR(32) ASCII representation (XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX).

Syntax Model

```
UUID_TO_CHAR( <string> )
```

Example

```
select uuid_to_char(gen_uuid())
  from rdb$database;
```

SOME_COL = ? OR ? IS NULL Predication

Adriano dos Santos Fernandes

Tracker reference CORE-2298

By popular request, particularly from Delphi programmers, support has been implemented for a predication that is able to “OR” test both the equivalence between a column and a parameter and whether the value passed to the

parameter is NULL. This construct is often desired as a way to avoid the need to prepare one query to request a filtered result set and another for the same query without the filter.

Users of Delphi and other programming interfaces that apply client-side object names to parameters wanted the ability for the DSQL engine to recognise a usage like the following:

```
WHERE col1 = :param1 OR :param1 IS NULL
```

At the API level, the language interface translates the query to

```
WHERE col1 = ? OR ? IS NULL
```

That presented two problems:

1. While the programmer treated the parameter **:param1** as though it were a single variable with two references, the API could not: it is presented with two *parameters*
2. The second parameter is of an unknown data type and the program has no way to assign to it

What was needed to solve this problem was to introduce a new data type to handle the "? is NULL" condition and teach Firebird to do the right thing when it received such a request.

The implementation works like this. To resolve the first problem, the request must supply *two* parameters (for our Delphi example):

```
WHERE col1 = :param1 OR :param2 IS NULL
```

- If "param1" is not NULL, the language interface is required to assign the correct value for the first parameter, set the **XSQLVAR.sqlind** to NOT NULL and leave XSQLVAR.sqldata NULL.
- If "param2" is NULL, the language interface is required to set the XSQLVAR.sqlind of *both* parameters to NULL and leave the XSQLVAR.sqldata NULL.

In other words, for the parameter (?) in **? IS NULL**
:

- XSQLVAR.sqlind should be set in accordance with NULL/NON-NULL state of the parameter. This is the type of parameter that is described by the new constant SQL_NULL.
- The XSQLVAR.sqldata of a SQL_NULL type of parameter should always be passed by the client application as a NULL pointer and should never be accessed.

> **NULL Specified in the Output Set**
>
> When NULL is specified as an output constant (**select NULL from ...**), it continues to be described as CHAR(1), rather than by SQL_NULL. That may change in a future version.

Extension to LIST() Function

Adriano dos Santos Fernandes

Tracker reference CORE-1453

A string expression is now allowed as the delimiter argument of the LIST() function.

Example

```
SELECT
  DISCUSSION_ID,
  LIST(COMMMENT, ASCII_CHAR(13))
FROM COMMENTS
  GROUP BY DISCUSSION_ID;
```

Optimizer Improvements

Changes in optimizer logic that address recognised problems include:

CROSS JOIN Logic (D. Yemanov)
When a CROSS JOIN involved an empty table, the optimizer had no special logic to detect that the query was futile and return the empty set immediately. That shortcut logic has now been implemented.

Tracker reference CORE-2200.

> **Note**
>
> The same change was implemented in V.2.1.2.

Derived Tables (A. dos Santos Fernandes)
The limit on the number of contexts available when using derived tables has been raised.

Tracker reference CORE-2029.

Timing of DEFAULT Evaluation (A. dos Santos Fernandes)
Under rare conditions, the early evaluation of a DEFAULT value or expression defined for a column might give rise to a confused situation regarding the evaluation of an input value supplied for that column. The possibility was addressed by deferring the evaluation of DEFAULT and not actually performing the evaluation at all unless it was actually necessary.

Tracker reference CORE-1842.

Index Use for NOT IN Searches (A. dos Santos Fernandes)
Better performance has been achieved for the NOT IN predicate by enabling the use of an index.

Tracker reference CORE-1137.

Undo Log Memory Consumption (D. Yemanov)
Excessive memory consumption by the Undo log after a lengthy series of updates in a single transaction has been avoided.

Tracker reference CORE-1477.

Other Improvements

Other changes to smooth out the little annoyances include:

FREE_IT Error Detection (A. dos Santos Fernandes)

Previously, a UDF declared with FREE_IT would crash if the pointer returned had not been allocated by the **ib_util_malloc()** function. Now, such a condition is detected, an exception is thrown and the pointer remains in its allocated state.

Tracker reference CORE-1937.

"Expression evaluation not supported" message improved (C. Valderrama)

A number of secondary GDS codes were introduced to provide more details about an operation that fails with an "Expression evaluation not supported" exception, for example:

```
'Argument for @1 in dialect 1 must be string or numeric'
'Strings cannot be added to or subtracted from DATE or TIME types'
'Invalid data type for subtraction involving DATE, TIME or TIMESTAMP types'
 etc.
```

These detailed messages follow the GDS code for the **isc_expression_eval_err (expression evaluation not supported)** error in the status vector.

Tracker reference CORE-1799.

Chapter 11

Procedural SQL (PSQL)

Several significant changes appear in Firebird's procedural language (PSQL), the language set for triggers, stored procedures and dynamic executable blocks, especially with regard to new extensions to the capabilities of EXECUTE STATEMENT. This release also heralds the arrival of the “autonomous transaction”.

Quick Links

Autonomous Transactions

Adriano dos Santos Fernandes

Tracker reference CORE-1409.

This new implementation allows a piece of code to run in an autonomous transaction within a PSQL module. It can be handy for a situation where you need to raise an exception but do not want the database changes to be rolled back.

The new transaction is initiated with the same isolation level as the one from which it is launched. Any exception raised in a block within the autonomous transaction will cause changes to be rolled back. If the block runs through until its end, the transaction is committed.

> **Warning**
>
> Because the autonomous transaction is independent from the one from which is launched, you need to use this feature with caution to avoid deadlocks.

Syntax Pattern

```
IN AUTONOMOUS TRANSACTION
DO
  <simple statement | compound statement>
```

Example of Use

```
create table log (
  logdate timestamp,
  msg varchar(60)
);

create exception e_conn 'Connection rejected';

set term !;

create trigger t_conn on connect
as
begin
  if (current_user = 'BAD_USER') then
  begin
    in autonomous transaction
    do
    begin
      insert into log (logdate, msg) values (current_timestamp, 'Connection rejected');
    end

    exception e_conn;
  end
end!

set term ;!
```

Borrow Database Column Type for a PSQL Variable

Adriano dos Santos Fernandes

Tracker reference CORE-1356.

This feature extends the implementation in v.2 whereby domains became available as “data types” for declaring variables in PSQL. Now it is possible to borrow the data type of a column definition from a table or view for this purpose.

Syntax Pattern

```
    data_type ::=
         <builtin_data_type>
       | <domain_name>
       | TYPE OF <domain_name>
       | TYPE OF COLUMN <table or view>.<column>
```

> **Note**
>
> TYPE OF COLUMN gets only the type of the column. Any constraints or default values defined for the column are ignored.

Examples

```
CREATE TABLE PERSON (
  ID INTEGER,
```

```
  NAME VARCHAR(40)
  );

CREATE PROCEDURE SP_INS_PERSON (
  ID TYPE OF COLUMN PERSON.ID,
  NAME TYPE OF COLUMN PERSON.NAME
  )
  AS
DECLARE VARIABLE NEW_ID TYPE OF COLUMN PERSON.ID;
BEGIN
  INSERT INTO PERSON (ID, NAME)
    VALUES (:ID, :NAME)
    RETURNING ID INTO :NEW_ID;
END
```

New Extensions to EXECUTE STATEMENT

Unusually for our release notes, we begin this chapter with the full, newly extended syntax for the EXECUTE STATEMENT statement in PSQL and move on afterwards to explain the various new features and their usage.

```
[FOR] EXECUTE STATEMENT <query_text> [(<input_parameters>)]
    [ON EXTERNAL [DATA SOURCE] <connection_string>]
    [WITH {AUTONOMOUS | COMMON} TRANSACTION]
    [AS USER <user_name>]
    [PASSWORD <password>]
    [WITH CALLER PRIVILEGES]
    [INTO <variables>]
```

Note

The order of the optional clauses is not fixed so, for example, a statement based on the following model would be just as valid:

```
    [ON EXTERNAL [DATA SOURCE] <connection_string>]
    [WITH {AUTONOMOUS | COMMON} TRANSACTION]
    [AS USER <user_name>]
    [PASSWORD <password>]
    [WITH CALLER PRIVILEGES]
```

Clauses cannot be duplicated.

Context Issues

If there is no ON EXTERNAL DATA SOURCE clause present, EXECUTE STATEMENT is normally executed within the CURRENT_CONNECTION context. This will be the case if the AS USER clause is omitted, or it is present with its <user_name> argument equal to CURRENT_USER.

However, if <user_name> is not equal to CURRENT_USER, then the statement is executed in a separate connection, established without Y-Valve and remote layers, inside the same engine instance.

Note

In the absence of an AS USER <user_name> clause, CURRENT_USER is the default.

Authentication

Where server authentication is needed for a connection that is different to CURRENT_CONNECTION, e.g., for executing an EXECUTE STATEMENT command on an external datasource, the AS USER and PASSWORD clauses are required. However, under some conditions, the PASSWORD may be omitted and the effects will be as follows:

1. On Windows, for the CURRENT_CONNECTION (i.e., no external data source), trusted authentication will be performed *if it is active* and the AS USER parameter is missing, null or equal to CURRENT_USER.

2. If the external data source parameter is present and its <connection_string> refers to the same database as the CURRENT_CONNECTION, the effective user account will be that of the CURRENT_USER.

3. If the external data source parameter is present and its <connection_string> refers to a different database than the one CURRENT_CONNECTION is attached to, the effective user account will be the operating system account under which the Firebird process is currently running.

In any other case where the PASSWORD clause is missing, only *isc_dpb_user_name* will be presented in the DPB (attachment parameters) and native authentication will be attempted.

Transaction Behaviour

The new syntax has an optional clause for setting the appropriate transaction behaviour: WITH AUTONOMOUS TRANSACTION and WITH COMMON TRANSACTION. WITH COMMON TRANSACTION is the default and does not need to be specified. Transaction lifetimes are bound to the lifetime of CURRENT_TRANSACTION and are committed or rolled back in accordance with the CURRENT_TRANSACTION.

The behaviour for WITH COMMON TRANSACTION is as follows:

a. Causes any transaction in an external data source to be started with the same parameters as CURRENT_TRANSACTION; otherwise

b. Executes the statement inside the CURRENT_TRANSACTION; or

c. May use another transaction that is started internally in CURRENT_CONNECTION.

The WITH AUTONOMOUS TRANSACTION setting starts a new transaction with the same parameters as CURRENT_TRANSACTION. That transaction will be committed if the statement is executed without exceptions or rolled back if the statement encounters an error.

Inherited Access Privileges

Vladyslav Khorsun

Tracker reference CORE-1928.

By design, the original implementation of EXECUTE STATEMENT isolated the executable code from the access privileges of the calling stored procedure or trigger, falling back to the privileges available to the

CURRENT_USER. In general, the strategy is wise, since it reduces the vulnerability inherent in providing for the execution of arbitrary statements. However, in hardened environments, or where privacy is not an issue, it could present a limitation.

The introduction of the optional clause WITH CALLER PRIVILEGES now makes it possible to have the executable statement inherit the access privileges of the calling stored procedure or trigger. The statement is prepared using any additional privileges that apply to the calling stored procedure or trigger. The effect is the same as if the statement were executed by the stored procedure or trigger directly.

> **Important**
>
> The WITH CALLER PRIVILEGES option is not compatible with the ON EXTERNAL DATA SOURCE option.

External Queries from PSQL

Vladyslav Khorsun

Tracker reference CORE-1853.

EXECUTE STATEMENT now supports queries against external databases by inclusion of the ON EXTERNAL DATA SOURCE clause with its <connection_string> argument.

The <connection_string> Argument

The format of <connection_string> is the usual one that is passed through the API function *isc_attach_database()*, viz.

```
[<host_name><protocol_delimiter>]database_path
```

Character Set

The connection to the external data source uses the same character set as is being used by the CURRENT_CONNECTION context.

Access Privileges

If the external data source is on another server then the clauses AS USER <user_name> and PASSWORD <password> will be needed.

The clause WITH CALLER PRIVILEGES is a no-op if the external data source is on another server.

MORE INFORMATION REQUIRED. ROLES?

> **Note**
>
> Use of a two-phase transaction for the external connection is not available in V.2.5.

EXECUTE STATEMENT with Dynamic Parameters

Vladyslav Khorsun
Alex Peshkov

Tracker reference CORE-1221.

The new extensions provide the ability to prepare a statement with dynamic input parameters (placeholders) in a manner similar to a parameterised DSQL statement. The actual text of the query itself can also be passed as a parameter.

Syntax Conventions

The mechanism employs some conventions to facilitate the run-time parsing and to allow the option of “naming” parameters in a style comparable with the way some popular client wrapper layers, such as Delphi, handle DSQL parameters. The API's own convention, of passing unnamed parameters in a predefined order, is also supported. However, named and unnamed parameters cannot be mixed.

The New Binding Operator

At this point in the implementation of the dynamic parameter feature, to avoid clashes with equivalence tests, it was necessary to introduce a new assignment operator for binding run-time values to named parameters. The new operator mimics the Pascal assignment operator:“:=”.

Syntax for Defining Parameters

```
    <input_parameters> ::=
      <named_parameter> | <input_parameters>, <named_parameter>

    <named_parameter> ::=
      <parameter name> := <expression>
```

Example for **named input parameters**

For example, the following block of PSQL defines both <query_text> and named <input_parameters> (<named_parameter>):

```
EXECUTE BLOCK AS
  DECLARE S VARCHAR(255);
  DECLARE N INT = 100000;
  BEGIN
  /* Normal PSQL string assignment of <query_text> */
    S = 'INSERT INTO TTT VALUES (:a, :b, :a)';

    WHILE (N > 0) DO
    BEGIN
    /* Each loop execution applies both the string value
       and the values to be bound to the input parameters */
```

```
      EXECUTE STATEMENT (:S) (a := CURRENT_TRANSACTION, b := CURRENT_CONNECTION)
      WITH COMMON TRANSACTION;
      N = N - 1;
    END
  END
```

Example for **unnamed input parameters**

A similar block using a set of unnamed input parameters instead and passing constant arguments directly:

```
EXECUTE BLOCK AS
  DECLARE S VARCHAR(255);
  DECLARE N INT = 100000;
  BEGIN
    S = 'INSERT INTO TTT VALUES (?, ?, ?)';

    WHILE (N > 0) DO
    BEGIN
      EXECUTE STATEMENT (:S) (CURRENT_TRANSACTION, CURRENT_CONNECTION, CURRENT_TRANSACTION);
      N = N - 1;
    END
  END
```

> **Note**
>
> Observe that, if you use both <query_text> and <input_parameters> then the <query_text> must be enclosed in parentheses, viz.
>
> ```
> EXECUTE STATEMENT (:sql) (p1 := 'abc', p2 := :second_param) ...
> ```

Examples Using EXECUTE STATEMENT

The following examples offer a sampler of ways that the EXECUTE STATEMENT extensions might be applied in your applications.

Test Connections and Transactions

A couple of tests you can try to compare variations in settings:

Test a) :Execute this block few times in the same transaction - it will create three new connections to the current database and reuse it in every call. Transactions are also reused.

```
EXECUTE BLOCK
  RETURNS (CONN INT, TRAN INT, DB VARCHAR(255))
AS
  DECLARE I INT = 0;
  DECLARE N INT = 3;
  DECLARE S VARCHAR(255);
BEGIN
  SELECT A.MON$ATTACHMENT_NAME FROM MON$ATTACHMENTS A
```

```
   WHERE A.MON$ATTACHMENT_ID = CURRENT_CONNECTION
    INTO :S;

  WHILE (i < N) DO
  BEGIN
    DB = TRIM(CASE i - 3 * (I / 3)
      WHEN 0 THEN '\\.\' WHEN 1 THEN 'localhost:' ELSE '' END) || :S;

    FOR EXECUTE STATEMENT
      'SELECT CURRENT_CONNECTION, CURRENT_TRANSACTION
      FROM RDB$DATABASE'
      ON EXTERNAL :DB
      AS USER CURRENT_USER PASSWORD 'masterkey' -- just for example
      WITH COMMON TRANSACTION
      INTO :CONN, :TRAN
    DO SUSPEND;

    i = i + 1;
  END
END
```

Test b) : Execute this block few times in the same transaction - it will create three new connections to the current database on every call.

```
EXECUTE BLOCK
  RETURNS (CONN INT, TRAN INT, DB VARCHAR(255))
AS
  DECLARE I INT = 0;
  DECLARE N INT = 3;
  DECLARE S VARCHAR(255);
BEGIN
  SELECT A.MON$ATTACHMENT_NAME
    FROM MON$ATTACHMENTS A
  WHERE A.MON$ATTACHMENT_ID = CURRENT_CONNECTION
   INTO :S;

  WHILE (i < N) DO
  BEGIN
    DB = TRIM(CASE i - 3 * (I / 3)
      WHEN 0 THEN '\\.\'
      WHEN 1 THEN 'localhost:'
      ELSE '' END) || :S;

    FOR EXECUTE STATEMENT
    'SELECT CURRENT_CONNECTION, CURRENT_TRANSACTION FROM RDB$DATABASE'
      ON EXTERNAL :DB
      WITH AUTONOMOUS TRANSACTION -- note autonomous transaction
      INTO :CONN, :TRAN
    DO SUSPEND;

    i = i + 1;
  END
END
```

Input Evaluation Demo

Demonstrating that input expressions evaluated only once:

```
EXECUTE BLOCK
  RETURNS (A INT, B INT, C INT)
AS
BEGIN
  EXECUTE STATEMENT (
    'SELECT CAST(:X AS INT),
            CAST(:X AS INT),
            CAST(:X AS INT)
       FROM RDB$DATABASE')
      (x := GEN_ID(G, 1))
    INTO :A, :B, :C;

  SUSPEND;
END
```

Insert Speed Test

Recycling our earlier examples for input parameter usage for comparison with the non-parameterised form of EXECUTE STATEMENT:

```
RECREATE TABLE TTT (
  TRAN INT,
  CONN INT,
  ID INT);

-- Direct inserts:

EXECUTE BLOCK AS
  DECLARE N INT = 100000;
BEGIN
  WHILE (N > 0) DO
  BEGIN
    INSERT INTO TTT VALUES (CURRENT_TRANSACTION, CURRENT_CONNECTION, CURRENT_TRANSACTION);
    N = N - 1;
  END
END

-- Inserts via prepared dynamic statement
-- using named input parameters:

EXECUTE BLOCK AS
  DECLARE S VARCHAR(255);
  DECLARE N INT = 100000;
BEGIN
  S = 'INSERT INTO TTT VALUES (:a, :b, :a)';

  WHILE (N > 0) DO
  BEGIN
    EXECUTE STATEMENT (:S)
      (a := CURRENT_TRANSACTION, b := CURRENT_CONNECTION)
    WITH COMMON TRANSACTION;
    N = N - 1;
  END
END

-- Inserts via prepared dynamic statement
-- using unnamed input parameters:
```

```
EXECUTE BLOCK AS
DECLARE S VARCHAR(255);
DECLARE N INT = 100000;
BEGIN
  S = 'INSERT INTO TTT VALUES (?, ?, ?)';

  WHILE (N > 0) DO
  BEGIN
    EXECUTE STATEMENT (:S) (CURRENT_TRANSACTION, CURRENT_CONNECTION, CURRENT_TRANSACTION);
    N = N - 1;
  END
END
```

Chapter 12

International Language Support (INTL)

Adriano dos Santos Fernandes

Some improvements appear in this release to tighten and enhance Firebird's handling capabilities for international language environment.

Default COLLATION Attribute for a Database

Databases of ODS 11.2 and higher can now optionally be created with a default collation associated with the default character set. For details, please see Default COLLATION Attribute for a Database in the DDL chapter.

ALTER CHARACTER SET Command

DDL syntax has been introduced to enable the default collation for a character set to be set at database level. For details, please see ALTER CHARACTER SET Command in the DDL chapter.

Connection Strings & Character Sets

Capability has been implemented in the API database connection (DPB) area to interoperate with the character set and/or code page of server and client, to avoid the previous problems that could occur when file names contained non-ASCII characters.

Refer to the topic Connection Strings & Character Sets in the chapter *Changes to the Firebird API and ODS*. Even if you are not normally interested in the API, this topic will be a worthwhile read if you have been bothered with such issues.

Other Improvements

Malformed UNICODE_FSS Characters Disallowed

Tracker reference CORE-1600.

Malformed characters are no longer allowed in data for UNICODE_FSS columns.

Repair Switches for Malformed Strings

New restore switches were added to the *gbak* utility code for the purpose of repairing malformed UNICODE_FSS data and metadata by restoring a backup of the affected database. Details are in the *gbak* section of the Utilities chapter.

Numeric Sort Attributes

Tracker reference: CORE-1945)

For UNICODE collations only, a custom attribute NUMERIC-SORT has been enabled for specifying the order by which to sort numerals.

Format & Usage

```
NUMERIC-SORT={0 | 1}
```

The default, 0, sorts numerals in alphabetical order. For example:

```
1
10
100
2
20
```

1 sorts numerals in numerical order. For example:

```
1
2
10
20
100
```

Example

```
create collation unicode_num for utf8
from unicode 'NUMERIC-SORT=1';
```

Character Sets and Collations

UNICODE_CI_AI

Tracker reference CORE-824.

UNICODE_CI_AI: case-insensitive, accent-insensitive collation added for UNICODE.

WIN_1258

Tracker reference CORE-2185.

Added alias WIN_1258 for WIN1258 character set, for consistency with other WIN* character sets.

SJIS and EUCJ Character Sets

Tracker reference CORE-2103.

Strings in SJIS and EUCJ character sets are now verified for well-formedness.

Chapter 13

Command-line Utilities

Retrieve Password from a File or Prompt

Alex Peshkov

Any command-line utility that takes a **-password** parameter is vulnerable to password sniffing, especially when the utility is run from a script. Since v.2.1, the [PASSWORD] argument has displayed in the process list on POSIX platforms as an asterisk (*), which was an improvement on showing it in clear.

As a second stage towards hiding the password from unauthorised eyes, this release enables it to be retrieved from a file or (on POSIX) from STDIN.

New -fetch_password Switch

Firebird 2.5 introduces the new switch *-fet[ch_password]* as an optional replacement for *-pa[ssword]* for all command-line utilities that take a password for authentication purposes. The switch may be progressively abbreviated from the right, conforming to the established rules.

PLEASE NOTE

1. The exception to the rules is the *qli* utility, for which only *-F* is valid.
2. The new switch *cannot* be applied to substitute for the *-pw* switch of the *gsec* utility.

Usage of -fetch_password

The switch requires one parameter, an unquoted string that is the file path for the file containing the password. If the call is not made by a system user with Superuser/Administrator privileges, the location must be accessible by the system user making the call.

For example,

```
isql -user sysdba -fet passfile server:employee
```

extracts the first line of from a file named “passfile” in the current working directory and loads it into the [PASSWORD] argument of the call.

The filename can be specified as *stdin*:

```
isql -user sysdba -fet stdin server:employee
```

If stdin is the terminal, a prompt is presented—

```
Enter password:
```

—requiring the operator to type in the password.

> **Tip**
>
> On POSIX, the operator will also be prompted if s/he specifies
>
> ```
>    -fetch /dev/tty
> ```
>
> This technique could be useful if, for example, you needed to restore from *stdin* (all one line):
>
> ```
> bunzip2 -c emp.fbk.bz2 | gbak -c stdin /db/new.fdb
>    -user sysdba -fetch /dev/tty
> ```

gsec and fbsvcmgr

Since v.2.1, domain administrators have had full access to the user management functions. This version adds the **ALTER ROLE** operator to enable SYSDBA-privileged access to user databases by the appointed OS administinstrator, by way of the RDB$ADMIN role.

However, it was still not possible for such a user to attach directly to the security database, except by way of an embedded connection, which resulted in a functional incompatibility with v.2.1.

New -mapping Switch for gsec

The new **-mapping** switch is used to enable or disable the OS user's association with the RDB$ADMIN role in the security database. Its format is:

```
mapping {set|drop}
```

Mapping Tags for fbsvcmgr

The corresponding new tag items for the Service Parameter Block are **isc_action_svc_set_mapping** and **isc_action_svc_drop_mapping**. The appropriate support for these tag items has been added to the *fbsvcmgr* utility.

gbak

Repair Switches for Malformed Strings

Adriano dos Santos Fernandes

Tracker reference CORE-1831.

The *gbak* utility has two new restore switches intended to repair malformed UNICODE_FSS character data and metadata, respectively, when restoring the backup of an affected database.

Switch Syntax

```
-FIX_FSS_D(ATA) <charset> -- fix malformed UNICODE_FSS data
-FIX_FSS_M(ETADATA) <charset> -- fix malformed UNICODE_FSS metadata
```

Preserve Character Set Default Collation

Adriano dos Santos Fernandes

An improvement allows the current value of RDB$DEFAULT_COLLATE_NAME in the system table RDB $CHARACTER_SETS to survive the backup/restore cycle.

Tracker reference CORE-789.

nBackup

An improvement has been done for POSIX versions to address a problem whereby the full backup tool of the *nBackup* incremental backup utility would hog I/O resources when backing up large databases, bringing production work to a standstill. Now, nBackup tries to read from the operating system cache before attempting to read from disk, thus reducing the I/O load substantially.

> **Note**
>
> The “cost” may be a 10 to 15 percent increase in the time taken to complete the full backup under high-load conditions.

Tracker reference CORE-2316.

isql

SQLSTATE instead of SQLCODE

Claudio Valderrama

isql now returns the SQLSTATE completion code in diagnostics, instead of the now deprecated SQLCODE. For more information, see the topic Support for SQLSTATE Completion Codes in the chapter *Changes to the Firebird API and ODS*.

gpre (Precompiler)

Some Updates

Stephen Boyd
Adriano dos Santos Fernandes

Tracker reference CORE-1527.

GPRE now supports the IS NOT DISTINCT predicate and CASE/NULLIF/COALESCE/SUBSTRING functions, as well as the whole set of CURRENT_* context variables.

Deprecated Features with Future Impact on Utilities

In anticipation of the dropping of the intrinsic function **isc_ddl** from the Firebird 3 codebase, certain features currently available in the *gdef* and *gpre* tools are deprecated—meaning that, whilst they may work in V.2.5, they will fail in Firebird 3. More details can be found in the Compatibility chapter.

Chapter 14

Installation Notes

> **Installation And Migration Guide**
>
> The latest version of Installation and Migration Guide for Firebird versions 2.0.x and 2.1.x is still applicable to the v.2.5 series. If a copy of this document is not present in your /doc/ directory, you can download it from the Firebird website.

Some improvements have been done to solve issues that could arise with the binary installation packages.

Linux (POSIX)

Alex Peshkov

(CORE-2195): the Linux Classic installation scripts were reviewed to improve the assignment of ownership and access rights to documentation and other components.

(CORE-2392): Cleanups of installation scripts for all active POSIX ports for Superserver and Superclassic were done to address a problem with the Guardian on these platforms.

Windows

Vlad Khorsun

Tracker entry: CORE-2243

> **Note**
>
> Because the changes took effect from V.2.1.2, this discussion also appears as a special topic in the *V.2 Installation and Migration document*.

Managing MSCV8 Assemblies

Firebird 2.5 is built by the Microsoft MSVC8 compiler in Visual Studio 2005. Because all the Firebird binaries are built to use dynamic linking, they all require run-time libraries.

To avoid the dll-hell issue Microsoft introduced new rules for the distribution of components that may be shared by multiple applications. From Windows XP forward, shared libraries—such as the Visual C++ and Visual C runtimes `msvcp80.dll`, `msvcr80.dll` and `mscvcm80.dll`—must be distributed as shared or as private assemblies.

- The Microsoft MSI Installer installs shared assemblies into the common special folder `SxS` for use by multiple applications.
- Private assemblies are distributed with applications and should be put into the application folder. Use of the `\system32` folder for assemblies is now prohibited on the XP, Server2003 and Vista platform families.

Installing Runtimes as a Shared Assembly

To install the runtimes as a shared assembly, the deployment system must have MSI 3.0 installed and the user must have administrative privileges. Often, this is not possible with an application being deployed with Firebird Embedded: it must be installed ready-to-run. In that case, do not plan to install the runtimes as a shared assembly.

Installing Runtimes as a Private Assembly

To install the MSVC8 run-time libraries as a private assembly its contents—the three DLLs mentioned above and the assembly's manifest file, `Microsoft VC80.CRT.manifest`—must be put into every folder where a dependent binary (.exe or .dll) resides, because of built-in checks for the folders that are the expected location of the runtimes that are equivalent to the compile-time libraries that were used.

A typical installation of Firebird Embedded would thus require three complete copies of the MSVC8 run-time assembly: one in the application folder and one each into the \intl and \udf folders. To avoid the issue of bloating the installation, some changes were done for V.2.1.2 in the way some of the Firebird binaries are built. (See also *Tracker entry CORE-2243*).

These are the changes that enable Firebird Embedded to work even if the application structure does not incorporate the MSVC8 runtime assembly:

a. The libraries ib_util.dll, fbudf.dll, ib_udf.dll, fbintl.dll are built without any embedded manifest. The effect is to avoid having the loader search for a MSVC8 assembly in the same folder as corresponding DLL. For this to work, the host process must have already loaded the MSVC8 run-time via manifest before any attempt is made to load these secondary DLL's.

b. fbembed.dll now has code to create and activate the activation context from its own manifest before loading any secondary DLL that might be required.

Notes

a. It is highly recommended to use the Microsoft redistribution package to install the MSVC8 run-time! The executable installer `vcredist_x86.exe` or `vcredist_x64.exe` (as appropriate to your kit selection) should be present in the zip kits for the full installation and the Embedded version. If not, it can be downloaded from the Microsoft download site.

b. Third party UDFs must satisfy *one of the following requirements* if a MSVC8 run-time assembly is installed as private assembly. When compiling the UDF library, the MSVC8 runtime *EITHER*:

- is NOT used
- is used but the build is done without the embedded manifest
- is used and the build is done with the embedded manifest—the default option in the MSVC IDE. In this case the MSVC8 assembly must be in the same folder as the UDF library

Chapter 15

Compatibility Issues

Dmitry Yemanov

For migrating v.2.0.x or v.2.1.x databases to Firebird 2.5, a number of incompatibilities that are likely to affect existing databases or existing applications should be noted. It is not recommended that you begin a migration until you have resolved these.

Effects of Unicode Metadata

If you have not previously updated text objects within the metadata of your databases to be in character set UTF8, restoring a database until V.2.5 will fail with “malformed string” errors. To resolve this it is necessary to pay attention to the files in the `/misc/upgrade/metadata` directory of your installation and to use the new `-fix_fss_data and -fix_fss_metadata` switches in the *gbak* command line.

Configuration Parameters Removed

The deprecated configuration parameters `OldParameterOrdering` and `CreateInternalWindow` are no longer supported and have been removed from `firebird.conf`.

Two parameters that allowed tuning of the Lock Manager in previous versions are not required with the new Lock Manager implementation and have been removed. They are `LockSemCount` and `LockSignal`.

SQL Language Changes

It will be necessary to pay attention to some changes in the SQL language implementation.

Reserved Words

Some new reserved keywords are introduced. The full list is available in the chapter New Reserved Words and Changes. Any identifiers using those words in DSQL statements or PSQL code will need to be changed, either by renaming the identifiers concerned or, for dialect 3 databases, by double-quoting them.

Execution Results

Some changes will now cause exceptions during run-time execution of queries, including those that are run during the execution of the *gbak* utility code (backups and restores).

Malformed String Errors

Well-formedness checks are now performed on UNICODE_FSS strings and text blobs. If new or existing UNICODE_FSS is malformed, it will now cause exceptions at execution time.

Logic Change in SET Clause

Previously, when the SET clause of the UPDATE statement assigned new values to columns, the new value replaced the old value immediately. If the same column was assigned or assigned to more than once, the current value would be that of the assignment most recently done. In other words, previously, assignment order mattered.

To bring Firebird in line with the standard, from this version forward, only the original value of a column is accessible to any assignment in the SET clause.

For a period, it is possible to revert to the legacy behavior by setting the temporary parameter OldSetClauseSemantics in `firebird.conf`. This parameter will be deprecated and removed in future releases.

Utilities

Be on the watch for the effects of the following changes to the Firebird command-line utilities.

fb_lock_print

Because v.2.5 maintains separate lock structures for each database on the server, *fb_lock_print* now requires a database path name in order to print the lock table. Include the new switch `-d <path name>` in the command line to specify *the filesystem path* to the database you wish to analyse.

Deprecated Features with Future Impact

In anticipation of the dropping of the intrinsic function **isc_ddl** from the Firebird 3 codebase, certain features currently available in the *gdef* and *gpre* tools are deprecated—meaning that, whilst they may work in V.2.5, they will fail in Firebird 3.

- *gdef* will no longer be supported at all. Instead, *isql* should be used, with regular DDL commands.
- For *gpre* pre-processing, replace all DDL operations with

```
EXEC SQL
  EXECUTE IMMEDIATE "..."
```

- In all custom applications, calls to **isc_ddl** must be replaced with SQL DDL statement requests.

API Changes

Notice the following changes to the application programming interface (API) that is implemented in the client libraries.

Rejection of Inconsistent TPB Options

The API functions **isc_start_transaction()** and **isc_start_multiple()** will now reject combinations of transaction parameter buffer (TPB) items that do not "belong together".

For example, a non-zero *wait timeout* is inconsistent with the *no wait* option; and *no record version* is inconsistent with any *transaction isolation mode* other than ReadCommitted. Now, instead of making some arbitrary (and possibly incorrect) assumption about the inherent ambiguities, the engine will reject such combinations as invalid.

For more information, see the topic Transaction Diagnostics in the chapter *Changes in the Firebird Engine*.

Addition of SQL_NULL Constant

New SQL_NULL constant was introduced to enable the predication **OR ? IS NULL** to be recognised and processed with the expected outcome and without engendering the "Data type unknown" exception. This affects how the XSQLVAR structures are populated for such queries. For information, refer to the topic SOME_COL = ? OR ? IS NULL Predication in the DML chapter.

Security Hardening

The following changes should be noted.

No SYSDBA Auto-mapping (Windows)

In V.2.1, members of administrative Windows groups were mapped to SYSDBA by default. From V.2.5 forward, SYSDBA mapping is controlled on per-database basis using the new SQL command

ALTER ROLE RDB$ADMIN SET/DROP AUTO ADMIN MAPPING

For more details, refer to the chapter on Security.

Default Authentication Method (Windows)

In V.2.1, where support for Windows trusted authentication was introduced, the default authentication method applied was *mixed*, i.e., the DPB or SPB would accept either native Firebird logins or trusted user logins. Thus, he *Authentication* parameter in `firebird.conf` showed *mixed* as the default.

From V.2.5 forward, the default is *native*. To have *mixed* or *trusted*, it is now necessary to configure this parameter specifically.

Tracker reference CORE-2376)

Chapter 16

Bugs Fixed

Firebird 2.5 Beta 1

The following bugs were reported as fixed in the Beta 1 release:

Core Engine/DSQL

(CORE-2389) Wrong matching of SIMILAR TO expression with brackets.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2356) On Windows the listener process of Classic Server was unable to create the necessary resources after restart if any worker process was present.

fixed by V. Khorsun

~ ~ ~

(CORE-2355) Incorrect handling of LOWER/UPPER when result string shrinks in terms of byte length.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2351) It was not possible to create a database whose <file specification> was an alias, even though the alias existed.

fixed by A. Peshkov

~ ~ ~

(CORE-2349) The “Invalid SQLDA” error was being falsely thrown.

fixed by V. Khorsun

~ ~ ~

(CORE-2348) More database corruption problems showed up resulting from transaction numbers overflowing 32-bit signed integer.

fixed by V. Khorsun

~ ~ ~

(CORE-2347) “Deadlock. Page type <N> lock conversion denied” errors were reported under a concurrent “connect-work-disconnect” style of load.

fixed by D. Yemanov

~ ~ ~

(CORE-2341) Hidden variables could conflict with output parameters, causing assertions, unexpected errors or even incorrect results.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2340) Bugcheck 258 (page slot not empty) could occur under high concurrent load.

fixed by V. Khorsun

~ ~ ~

(CORE-2339) Incorrect results were being returned for derived expressions based on aggregation and computation.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2265) Grouping by function was not working properly.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2320) A complex recursive query did not always return all rows.

fixed by V. Khorsun

~ ~ ~

(CORE-2317) **select * from (select cast(....** was returning null.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2313) INF_* functions could invalidate the whole output buffer with isc_info_truncated at the beginning, due to a boundary condition.

fixed by C. Valderrama

~ ~ ~

(CORE-2311) A WITH RECURSIVE query could cause memory leakage.

fixed by V. Khorsun

~ ~ ~

(CORE-2308) SIMILAR TO was producing random results with [x-y] expressions.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2300) The second evaluation of SUBSTRING() would throw an unexpected *arithmetic exception, numeric overflow, or string truncation* error.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2293) The wrong dependent object type (RELATION) was being stored in RDB$DEPENDENCIES for views.

fixed by V. Khorsun

~ ~ ~

(CORE-2289) The wrong name was being reported for the referenced primary key when a foreign key violation occurred during foreign key creation.

fixed by V. Khorsun

~ ~ ~

(CORE-2264) Doing ALTER DOMAIN on a domain with dependencies could leave a transaction handle in an inconsistent state and cause segmentation faults.

fixed by A. dos Santos Fernandes, D. Yemanov

~ ~ ~

(CORE-2258) Selecting UPPER (<blob>) from a UNION was causing an internal error.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2257) The error *Bugcheck 167 (invalid send request)* could occur while altering dependent procedures.

fixed by D. Yemanov

~ ~ ~

(CORE-2255) The error *String right truncation* could occur while altering a view with a join.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2242) The engine was incorrectly populating integer containers in the blob parameter buffer (BPB) with integers in machine-local format, causing errors on Big Endian platforms.

fixed by A. Peshkov

~ ~ ~

(CORE-2241) If an ALTER TABLE ALTER COLUMN.. operation was performed on a table in the course of a bulk insert operation, minor index corruption could occur causing subsequent queries to return the wrong number of records. The bug was traced to legacy code in BTR\compress_root().

fixed by V. Khorsun

~ ~ ~

(CORE-2237) An unresolved assertion was exhibited at src\jrd\intl.cpp, line 569.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2231) A *Bugcheck 179 (decompression overran buffer)* was thrown when the v.2.5 server attempted to read from the table RDB$TRIGGER_MESSAGES in a ODS 10.x database.

fixed by D. Yemanov

~ ~ ~

(CORE-2230) Input parameters for EXECUTE BLOCK were not being domain-checked.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2202) RDB$VIEW_RELATIONS was not being cleaned up when ALTER VIEW was processed.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2186) In the Windows embedded server, *fbintl.dll* was not being unloaded after the **isc_dsql_execute_immediate()** during the processing for CREATE DATABASE.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2183) An error was being thrown when a server shutdown was started while an “execute statement” request was open.

fixed by V. Khorsun

~ ~ ~

(CORE-2182) It was not possible to drop an existing UDF whose name was duplicated by the name of a new built-in function.

fixed by D. Yemanov

~ ~ ~

(CORE-2179) Deadlocks were exhibited when trying to shut down the server whilst an “execute statement” request was open.

fixed by A. Peshkov

~ ~ ~

(CORE-2176) COALESCE and GROUP BY were returning unexpected wrong results.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2174) The DATEADD() and DATEDIFF() functions were causing an assert in **TimeStamp::decode()**.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2154) A “request synchronization error” would occur when calling **isc_dsql_sql_info()** with the **isc_info_sql_records** parameter after the last record had been fetched with EXECUTE PROCEDURE.

fixed by V. Khorsun

~ ~ ~

(CORE-2153) The “|” option was causing the SIMILAR TO predicate to hang.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2140) Error messages after substitution of parameters contained '\n' characters instead of the actual line break.

fixed by C. Valderrama

~ ~ ~

(CORE-2138) If a stored procedure with an EXECUTE STATEMENT against an external database failed at runtime, the external database would remain attached.

fixed by V. Khorsun

~ ~ ~

(CORE-2132) Indexed retrieval could not be chosen if a stored procedure call was used in the comparison predicate.

fixed by D. Yemanov

~ ~ ~

(CORE-2117) Incorrect ROW_COUNT values were being returned with indexed retrieval and subqueries.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2115) For some long queries the query plan could go missing.

fixed by D. Yemanov

~ ~ ~

(CORE-2101) A *Bugcheck 249 (pointer page vanished)* error would be thrown when an attempt was made to fetch beyond the end-of-stream mark of an open PSQL cursor.

fixed by D. Yemanov

~ ~ ~

(CORE-2098) It was not possible to create a view that selected from a global temporary table.

fixed by V. Khorsun

~ ~ ~

(CORE-2081) RDB$DB_KEY in a subquery expression would incorrectly return NULL.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2078) If selective non-indexed predicates were involved in a join, the join plan was not optimized as well as it could be.

fixed by D. Yemanov

~ ~ ~

(CORE-2075) Parts of the RDB$DB_KEY of views could be inverted when using outer joins.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2073) Expression indices bug: incorrect result for the inverted boolean.

fixed by D. Yemanov

~ ~ ~

(CORE-2069) Incorrect view expansion when RDB$DB_KEY was used inside a view body.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2068) Comparision would return a wrong result with the **IN <subquery_expression>** operand if the <subquery_expression> argument involved the RDB$DB_KEY.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2067) ?? GROUP BY and RDB$DB_KEY problems ??

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2066) ?? Conversion of SQL_TEXT / SQL_VARCHAR to SQL_TIMESTAMP / SQL_TIME / SQL_DATE ??

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2062) Classic server exhibited a lock file remapping error.

fixed by A. Peshkov

~ ~ ~

(CORE-2053) Computed expressions could suffer from poor optimization if used inside the RETURNING clause of the INSERT statement.

fixed by D. Yemanov

~ ~ ~

(CORE-2045) A v.2.1 regression was picked up, whereby references to non-existent system fields with `blr_field` were not being resolved to NULL, whereas a parallel change involving `blr_fld` was exhibiting the proper corrective behaviour.

fixed by dos Santos Fernandes

~ ~ ~

(CORE-2044) ?? Incorrect result for UPDATE OR INSERT ... RETURNING OLD and non-nullable columns ???

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2041) UPDATE OR INSERT with GEN_ID() was causing the generator to step by 3.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2039) Domain-level CHECK constraints were processing NULL values wrongly.

fixed by D. Yemanov

~ ~ ~

(CORE-2038) The new EXECUTE STATEMENT implementation would assert or throw an error if used both before and after Commit Retaining or Rollback Retaining.

fixed by D. Yemanov

~ ~ ~

(CORE-2036) The order of parameters in an EXECUTE BLOCK statement was being reversed if the block was called from EXECUTE STATEMENT.

fixed by V. Khorsun

~ ~ ~

(CORE-2031) ??? NULL in the first record in a condition on RDB$DB_KEY ???

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2027) The buffer size for ORDER BY expressions involving system fields was being calculated wrongly.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2026) ??? Problem with a read-only marked database ???

fixed by V. Khorsun

~ ~ ~

(CORE-2022) CREATE USER was not being supported by the EXECUTE BLOCK statement.

fixed by A. Peshkov

~ ~ ~

(CORE-2008) ??? NOT NULL flag for procedure parameters in the system schema ???

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2002) A conversion error from a UDF result would leave a memory leak if the result was marked with FREE_IT.

fixed by C. Valderrama

~ ~ ~

(CORE-2001) When trying to show a conversion error, the message *arithmetic exception or string truncation* was sometimes appearing instead.

fixed by C. Valderrama

~ ~ ~

(CORE-2000) The Lock Manager could report false deadlocks under high load.

fixed by V. Khorsun

~ ~ ~

(CORE-1994) An “invalid database handle” could occur while executing a CREATE USER statement.

fixed by D. Yemanov

~ ~ ~

(CORE-1986) Altering the name of a domain was causing dependencies on the domain to be dropped.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1985) The Lock Manager code could periodically cause 100% CPU load.

fixed by V. Khorsun

~ ~ ~

(CORE-1984) The Lock Manager could falsely report a deadlock if one of the alleged participants was waiting with a permitted timeout.

fixed by V. Khorsun

~ ~ ~

(CORE-1980) Sweeper could consume 100% of CPU indefinitely.

fixed by V. Khorsun

~ ~ ~

(CORE-1970) Lock conversion denied (215) errors could occur.

fixed by V. Khorsun

~ ~ ~

(CORE-1962) The function **EXTRACT (MILLISECONDS FROM aTimeStampOrTime)** was returning incorrect results.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1958) A *Bugcheck 179 (decompression overran buffer)* consistency check error would be thrown when attempts were made to update the same record multiple times in a transaction.

fixed by D. Yemanov

~ ~ ~

(CORE-1957) Long access control lists (ACLs) were being truncated, causing privileges to disappear.

fixed by A. Peshkov

~ ~ ~

(CORE-1943) A statement that aggregated on a RAND() expression would return infinite rows.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1938) Bugcheck 243 (missing pointer page) was being thrown on preparing or executing statements that referred to a table being dropped or recreated by another connection.

fixed by D. Yemanov

~ ~ ~

(CORE-1936) The built-in function **LOG(base, number)** was not checking parameters and would deliver NAN values for out-of-range input instead of excepting.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1935) SIMILAR TO character classes were not being recognised correctly.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1914) If a problem occurred during table creation, the database could be left in an inconsistent state.

fixed by A. Peshkov

~ ~ ~

(CORE-1812) For some date/time expressions in dialect 1, indexes were not being used when they should have been.

fixed by D. Yemanov

~ ~ ~

(CORE-1650) An improbable case was demonstrated whereby something like **SELECT GEN_ID(..) FROM RDB$DATABASE** with a **GROUP BY** operation would cause rows to be generated infinitely.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1607) A correlated subquery that depended on a UNION stream would be poorly optimized.

fixed by D. Yemanov

~ ~ ~

(CORE-1606) ??? Ability to insert child record if parent record is locked but foreign key target unchanged ???

fixed by A. Potapchenko, V. Khorsun

~ ~ ~

(CORE-1575) Multiple updates to a table in a single transaction would throw up a serious memory bug.

fixed by D. Yemanov

~ ~ ~

(CORE-1544) When a user application created “temporary” stored procedures in run-time for some run-time purpose, the internal generator for the RDB$PROCEDURES.RDB$PROCEDURE_ID column could easily overflow the 32K limit (a signed SMALLINT) in its internal generator and cause a “numeric overflow” exception on trying to create the new stored procedure.

The fix wraps around the generated value at the 32K boundary, allowing reuse of existing gaps in the ID numbering. A similar fix was applied to RDB$GENERATORS and RDB$EXCEPTIONS as well.

fixed by D. Yemanov

~ ~ ~

(CORE-1343) ?? simple case and a subquery ??

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1246) Outer joins with derived tables returned incorrect results.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1245) Outer joins with views returned incorrect results.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-903) Firebird's behaviour with regard to multiple assignments referring to the same column in the SET clause of an UPDATE statement did not comply with the standard.

fixed by D. Yemanov

~ ~ ~

(CORE-501) COALESCE exhibited an optimization problem.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-216) Privileges could be lost if a database contained too many.

fixed by A. Peshkov

~ ~ ~

Server/Client Crashes

(CORE-2372) The server would crash inside CMP_release() while releasing an already freed collation resource.

fixed by V. Khorsun

~ ~ ~

(CORE-2368) An access violation would follow the call to isc_cancel_events() if the event was not found.

fixed by V. Khorsun

~ ~ ~

(CORE-2248) During port validation, the server was crashing in server.cpp/loopThread().

fixed by V. Khorsun

~ ~ ~

(CORE-2222) Storing a text blob with a transliterating blob filter could cause an access violation in the engine.

fixed by V. Khorsun

~ ~ ~

(CORE-2158) Client and embedded libraries could crash while being unloaded.

fixed by A. Peshkov

~ ~ ~

(CORE-2137) A database restore could crash the server when the configuration parameter *DummyPacketInterval* was set explicitly.

fixed by D. Yemanov

~ ~ ~

(CORE-2121) The Client library could crash in the course of an operation involving BLOBs.
c

fixed by A. Peshkov

~ ~ ~

(CORE-2071) The client library would crash if isc_dsql_prepare() was called with a NULL statement text.

fixed by A. Peshkov

~ ~ ~

(CORE-2064) A server process could crash during an exit while under high load.

fixed by A. Peshkov

~ ~ ~

(CORE-2061) ALTER VIEW WITH CHECK OPTION could crash the engine.

fixed by D. Yemanov

~ ~ ~

(CORE-2042) ?? Connection lost to a database when using AUTONOMOUS TRANSACTION ??

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1983) An out-of-memory condition in the operating system would cause an access violation.

fixed by A. Peshkov

~ ~ ~

(CORE-1965) The Lock Manager would crash with an invalid lock ID under concurrent DDL load.

fixed by D. Yemanov

~ ~ ~

(CORE-1894) Circular dependencies between computed fields would crash the server.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1963) The server could crash on commit when granting/revoking privileges from multiple connections simultaneously.

fixed by D. Yemanov

~ ~ ~

(CORE-1506) Server crashes with isc_dsql_execute_immediate() and a zero-length string.

fixed by A. Peshkov

~ ~ ~

(CORE-210) The Classic server would crash if the same stored procedure was being altered in two separate processes.

fixed by D. Yemanov

~ ~ ~

Remote Interface/API

(CORE-2307) API information requests were returning incomplete values in the results.

fixed by D. Yemanov

~ ~ ~

(CORE-2262) Abrupt termination of a client connection could occur.

fixed by A. Peshkov

~ ~ ~

(CORE-2234) Sometimes, terminated worker processes in Classic on Windows were still considered to be alive after termination, due to improper checks on the Firebird server's part. The same bug could cause the Firebird server to misbehave with prolonged deadlocks when under load.

fixed by V. Khorsun

~ ~ ~

(CORE-2151) When a temporary directory path had spaces within it, it was (wrongly) being truncated at the rightmost space.

fixed by V. Khorsun

~ ~ ~

(CORE-2033) The symbol _Unwind_GetIP in the client library was being left unresolved due to a missing static library linkage.

fixed by A. Peshkov

~ ~ ~

(CORE-2018) Only a single client could access a read-only database.

fixed by V. Khorsun

~ ~ ~

(CORE-2016) A client's attempt to use the XNET protocol causes it to hang if the attachment or the database has been shut down.

fixed by D. Yemanov

~ ~ ~

(CORE-1972) A non-SYSDBA user was able to change the Forced Writes mode of any database, along with several other database characteristics that should be restricted to the SYSDBA. This long-standing, legacy loophole in the handling of DPB parameters could lead to database corruptions or give ordinary users access to SYSDBA-only operations.

The changes could affect several existing applications, database tools and connectivity layers (drivers, components).

Same fix was backported to v.2.1.2 and v.2.0.5.

fixed by A. Peshkov

~ ~ ~

POSIX-specific

(CORE-2221) On POSIX platforms, any attachment to any database would fail after the access rights for security2.fdb were modified from 0660 to 0666.

fixed by P. Beach, A. Peshkov

~ ~ ~

(CORE-2093) SuperServer startup would fail on Solaris 64-bit.

fixed by A. Peshkov

~ ~ ~

Windows-specific

(CORE-2108) When using the Windows local protocol (XNET), the next available map number was calculated incorrectly, thus allowing the server to try to reuse a map number that already existed. If the “new” map's timestamp was equal to the timestamp of the pre-existing map, it would cause the `get_free_slot()` function to fail.

fixed by V. Khorsun

~ ~ ~

(CORE-2107) Establishing a TCP\IP connection to the Windows Classic Server could fail under high load.

fixed by V. Khorsun

~ ~ ~

(CORE-1923) Successful execution of **instsvc.exe remove** was returning 1 as its completion code, instead of 0.

fixed by D. Yemanov

~ ~ ~

MacOSX-specific

(CORE-2102) Firebird 2.5 would not build on MacOS (Darwin).

fixed by A. Peshkov

~ ~ ~

(CORE-2065) The MacOSX installation package was in violation of platform rules by not including the client library in the dynamic loader search paths.

fixed by P. Beach

~ ~ ~

Database Monitoring/Administration

(CORE-2209) Monitoring requests in high load conditions could become very slow and even block other activity during that time.

fixed by D. Yemanov

~ ~ ~

(CORE-2171) The column MON$CALLER_ID of table MON$CALL_STACK was reporting invalid IDs.

fixed by D. Yemanov

~ ~ ~

(CORE-2017) I/O statistics for stored procedures were not being kept account of in the monitoring tables.

fixed by D. Yemanov

~ ~ ~

(CORE-1944) On Big Endian platforms, the monitoring tables contained wrong data.

fixed by A. Peshkov

~ ~ ~

Security

(CORE-2087) When the configuration parameter *RemoteBindAddress* specified a hostname instead of an IP address, or specified a non-existent IP address, it would be silently ignored and the server would bind to all interfaces, without any notification in firebird.log or the system log. This was considered a potential security risk if the system had ports open to the Internet. Now, an invalid or unavailable IP address will be resolved to localhost (127.0.0.1).

fixed by A. Peshkov

~ ~ ~

(CORE-2055) Buffer overflow in Firebird client library.

fixed by A. Peshkov

~ ~ ~

International Language Support

(CORE-2278) Conversion from and to CP943C was incorrect on RISC machines.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2227) Problems were occurring in some environments when creating triggers that referred to column names with accented characters.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2123) ?? Problem with getting UNICODE_FSS data in the CP943C connection charset ??

fixed by A. dos Santos Fernandes, D. Kovalenko

~ ~ ~

(CORE-2122) ?? Translation of large text blobs between UNICODE_FSS / UTF8 and other charsets ??

fixed by A. dos Santos Fernandes, D. Kovalenko

~ ~ ~

(CORE-2095) Bug in CVJIS_eucj_to_unicode().

fixed by A. dos Santos Fernandes, D. Kovalenko

~ ~ ~

(CORE-2019) A UTF-8 conversion error (string truncation) was being thrown unexpectedly.

fixed by dos Santos Fernandes

~ ~ ~

(CORE-1989) A column with UNICODE_CI collation for UTF8 could not be used in a foreign key constraint.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1927) The procedure **sp_register_character_set** could generate a negative value for RDB $CHARACTER_SETS.RDB$CHARACTER_SET_ID.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1690) A condition in tables with UTF8 text was causing the error *Arithmetic exception, numeric overflow, or string truncation*.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1596) Bug in CsConvert::convert

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1432) The collation attribute of columns was not being propagated between record formats.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-316) A database with multi-byte characters in its name could not be opened.

fixed by A. dos Santos Fernandes

~ ~ ~

Services Manager

(CORE-1982) Simultaneous backups or restores invoked through the Services API could interfere with one another.

fixed by A. dos Santos Fernandes

~ ~ ~

Command-line Utilities

fb_lock_print

(CORE-2354) “fb_lock_print -ia” output was not being flushed to the file between iterations.

fixed by A. Peshkov

~ ~ ~

isql

(CORE-2270) When run in a *zlogin* console, *isql* would consume all memory and crash.

fixed by J. Swierczynski, A. Peshkov

~ ~ ~

gbak Backup/Restore Utility

(CORE-2291) The error *Bugcheck 284 (cannot restore singleton select data)* would be thrown when...??

fixed by V. Khorsun

~ ~ ~

(CORE-2285) A database with a large number of grants could become corrupted after a backup/restore.

fixed by A. Peshkov

~ ~ ~

(CORE-2245) A database with long exception messages defined would exhibit errors when being restored from a backup.

fixed by C. Valderrama

~ ~ ~

(CORE-2223) gbak was encountering several bugs when operating on the access control lists (ACLs) that store SQL privileges.

fixed by A. Peshkov

~ ~ ~

(CORE-2214) Security classes were being restored incorrectly.

fixed by A. dos Santos Fernandes

~ ~ ~

nbackup Utility

(CORE-2266) nBackup's database locking was not working correctly, causing database file growth to continue when database writes should have been in suspension.

fixed by V. Khorsun

~ ~ ~

(CORE-1696) Deadlock would occur in the Lock Manager when the NBackup utility was in use.

fixed by R. Simakov

~ ~ ~

gfix

(CORE-2271) The *gfix* utility had a legacy bug that exhibited itself during the database validation/repair routines on large databases. The privilege level of the user running these routines was being checked too late in the operation, thus allowing a non-privileged user (i.e., not SYSDBA or Owner) to start a validation operation. Once the privilege check occurred, the database validation could halt in mid-operation and thus be left unfinished, resulting in logical corruption that might not have been there otherwise.

fixed by A. Peshkov

~ ~ ~

(CORE-1961) A *Bugcheck 210 (page in use during flush)* consistency check error would be thrown during database validation.

fixed by D. Yemanov, R. Simakov

~ ~ ~

qli Query Utility for GDML

(CORE-2247) In the QLI utility, message and descriptor buffers were not properly aligned.

fixed by A. Peshkov

~ ~ ~

Miscellaneous Bugs

(CORE-2282) Truncating UDFs were broken for negative numbers below -1.

fixed by C. Valderrama

~ ~ ~

(CORE-2281) Rounding UDFs were broken for negative numbers.

fixed by C. Valderrama

~ ~ ~

Firebird 2.5 Alpha 1

The following bugs were reported as fixed in the Alpha 1 release:

Core Engine/DSQL

(CORE-1421) SuperServer could not shut down immediately if the shutdown request followed a failed login attempt.

fixed by A. Peshkov

~ ~ ~

(CORE-1907) Dropping and adding a domain constraint in the same transaction would leave incorrect dependencies.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1905) Hash sign (#) in filenames in aliases.conf was being handled incorrectly.

fixed by C. Valderrama

~ ~ ~

(CORE-1887) Newly created databases had wrong access rights.

fixed by A. Peshkov

~ ~ ~

(CORE-1869) Roles granting/revoking logic differed between v.2.0 and v.2.1.

fixed by A. Peshkov

~ ~ ~

(CORE-1841) If some VIEW used derived tables and long table names/aliases, it was possible to overflow RDB$VIEW_RELATIONS.RDB$CONTEXT_NAME.

fixed by V. Khorsun

~ ~ ~

(CORE-xxxx) Text

fixed by A. Peshkov

~ ~ ~

(CORE-1840) Each DDL execution would cause a small memory leak.

fixed by D. Yemanov

~ ~ ~

(CORE-1838) SET STATISTICS INDEX on an index for a GTT could wrongly change the index id by the maximum available number for the database page size.

fixed by V. Khorsun

~ ~ ~

(CORE-1830) Possible index corruption with multiple updates of the same record in the same transaction with savepoints being used.

fixed by V. Khorsun

~ ~ ~

(CORE-1817) The RelaxedAliasChecking parameter was having no effect with regard to RDB$DB_KEY.

fixed by V. Khorsun

~ ~ ~

(CORE-1811) Parser reacted incorrectly to the unquoted usage of the keyword “VALUE”.

fixed by D. Yemanov

~ ~ ~

(CORE-1798) RDB$DB_KEY was being evaluated as NULL in INSERT ... RETURNING.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1797) OLD/NEW.RDB$DB_KEY returned incorrect result in triggers.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1784) Error with EXECUTE PROCEDURE inside EXECUTE STATEMENT.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1777) Conflicting table reservation specifications were being allowed in the TPB.

fixed by C. Valderrama

~ ~ ~

(CORE-1775) Security checking was performing poorly during prepare.

fixed by V. Khorsun

~ ~ ~

(CORE-1770) Bugcheck 291 in DDL.

fixed by A. Peshkov

~ ~ ~

(CORE-1735) Behavour problem with SET DEFAULT action argument in referential integrity triggers.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1731) Sometimes engine would hang for several minutes, using 1000% CPU load but with no I/O activity.

fixed by V. Khorsun

~ ~ ~

(CORE-1730) Problems arose if one of the directories specified in the TempDirectories config setting was not available.

fixed by D. Yemanov

~ ~ ~

(CORE-1724) Common table expressions could not be used in computed columns nor in quantified predicates (IN / ANY / ALL).

fixed by V. Khorsun

~ ~ ~

(CORE-1694) Bug in CREATE/ALTER database trigger, where comments were in Russian.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1693) Error in EXECUTE STATEMENT inside CONNECT / TRANSACTION START triggers.

fixed by A. dos Santos Fernandes, D. Yemanov

~ ~ ~

(CORE-1689) “There are <n> dependencies” error message shows the wrong count of dependent objects

fixed by C. Valderrama

~ ~ ~

(CORE-1357) DummyPacketInterval mechanism was broken.

fixed by D. Yemanov

~ ~ ~

(CORE-1307) Switch -s of fb_inet_server was not being processed correctly

fixed by A. Peshkov

~ ~ ~

(CORE-479) Grants would overwrite previous entries in RDB$SECURITY_CLASSES.

fixed by A. Peshkov

~ ~ ~

Server Crashes

(CORE-1930) Possible server crash if procedure was altered to have no outputs and dependent procedures were not recompiled

fixed by V. Khorsun

~ ~ ~

(CORE-1919) Memory corruptions in EXECUTE STATEMENT could crash the server.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1884) Random crashes using stored procedures with expressions as default values for input parameters.

fixed by V. Khorsun

~ ~ ~

(CORE-1839) Server could crash when sorting by a field that was calculated using a recursive CTE.

fixed by A. Peshkov

~ ~ ~

(CORE-1793) Server crashes at prepare of query with unused parameterized CTE.

fixed by V. Khorsun

~ ~ ~

(CORE-1512) Server crashes due to the wrong parsing of the DEFAULT clause.

fixed by D. Yemanov

~ ~ ~

POSIX-specific

(CORE-1909) Garbage in firebird.log on linux/amd64

fixed by A. Peshkov

~ ~ ~

(CORE-1885) CREATE COLLATION caused lost connection under Posix.

fixed by A. dos Santos Fernandes, A. Peshkov

~ ~ ~

(CORE-1854) Value of CURRENT_USER might not be in upper case when using Unix OS authentication.

fixed by A. Peshkov

~ ~ ~

(CORE-1826) changeRunUser.sh and restoreRootRunUser.sh scripts were not changing the run user in the init.d scripts.

fixed by A. Peshkov

~ ~ ~

(CORE-1818) Temporary files used for temporary page spaces were not deleted after use on Posix platforms.

fixed by A. Peshkov

~ ~ ~

(CORE-1807) Server was being assigned to a non-canonical port after abnormal termination.

fixed by A. Peshkov

~ ~ ~

(CORE-1766) Owner and group of isc_monitor1 file on Linux classic server were incorrect.

fixed by A. Peshkov

~ ~ ~

(CORE-1671) atexit() calls in client libraries cause segfaults if the libraries were used in dlopen'ed modules.

fixed by A. Peshkov

~ ~ ~

Windows-specific

(CORE-1820) Setup program was failing to detect a running server.

fixed by P. Reeves, D. Yemanov

~ ~ ~

(CORE-1105, CORE-1390, CORE-1566 & CORE-1639) Aliases would not work properly for XNET connections.

fixed by D. Yemanov

~ ~ ~

Data Manipulation Language

(CORE-1910) Invalid fields were accepted in the insert clause for MERGE.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1859) Arithmetic overflow or division by zero could occur in MAX() function.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1828) Error with ABS() function in dialect 1.

fixed by A. dos Santos Fernandes

~ ~ ~

Remote Interface/API

(CORE-1868) Client library was crashing inside isc_dsql_free_statement().

fixed by A. Peshkov

~ ~ ~

(CORE-1763) The client library was not setting the options SO_KEEPALIVE nor TCP_NODELAY for the socket at connection.

fixed by V. Khorsun

~ ~ ~

(CORE-1755 and)CORE-1756 A couple of crash scenarios could occur in isc_start_transaction().

fixed by D. Kovalenko, A. Peshkov

~ ~ ~

(CORE-1726) Failure in isc_service_start().

fixed by A. Peshkov

~ ~ ~

(CORE-1079) Every attach of fbclient/fbembed library to the host process would leak 64KB of memory.

fixed by A. Peshkov

~ ~ ~

International Language Support

(CORE-1802) Some issues were reported concerning maximum key size using the PXW_CSY collation.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1774) A problem appeared with collate ES_ES_CI_AI.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1254) Problem with DISTINCT and insensitive collations.

fixed by A. dos Santos Fernandes

~ ~ ~

Database Monitoring/Administration

(CORE-1890) Database monitoring process could hang under high load.

fixed by D. Yemanov

~ ~ ~

(CORE-1881) Database monitoring could crash the server or badly affect its page locking logic.

fixed by D. Yemanov

~ ~ ~

(CORE-1728) Database monitoring would not work after a fresh Linux install.

fixed by A. Peshkov

~ ~ ~

Security

(CORE-1845) Some standard calls would show the server installation directory to regular users.

fixed by A. Peshkov

~ ~ ~

(CORE-1810) Some issues appeared concerning usernames with '.' characters.

fixed by A. Peshkov

~ ~ ~

Command-line Utilities

isql

(CORE-1891) SHOW VIEW would show nonsense information for view fields with expressions.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1875) Errors in scripts with CURRENT_DATE.

fixed by V. Khorsun

~ ~ ~

(CORE-1862) Extracted script was unusable with interdependent selectable procedures in FB 2.1

fixed by C. Valderrama

~ ~ ~

(CORE-1782) Isql would crash when fetching data for a column having an alias longer than 30 characters.

fixed by D. Yemanov

~ ~ ~

(CORE-1749) DDL statement with AUTODDL ON was not showing statistics.

fixed by D. Yemanov

~ ~ ~

(CORE-1507) ISQL linecount facility in scripts goes out of sync after an INPUT command.

fixed by C. Valderrama

~ ~ ~

(CORE-1363) ISQL would crash when the string converted from a double was longer than 23 bytes.

fixed by C. Valderrama

~ ~ ~

gsec

(CORE-1680) Gsec DISPLAY was showing only a few of the first users when the security databases contained more than 50 users.

fixed by A. Peshkov

~ ~ ~

gbak

(CORE-1911) Backup and restore were not thread-safe when using the Services API.

fixed by C. Valderrama

~ ~ ~

(CORE-1843) Gbak with Service Manager would not allow paths with spaces.

fixed by A. Peshkov

~ ~ ~

(CORE-1703) Delays/lockups when the gbak output was redirected to another process.

fixed by D. Yemanov

~ ~ ~

nbackup

(CORE-1876) Incremental backups with NBACKUP were broken in v.2.1.

fixed by N. Samofatov

~ ~ ~

Chapter 17

Firebird 2.5 Project Teams

Table 17.1. Firebird Development Teams

Developer	Country	Major Tasks
Dmitry Yemanov	Russian Federation	Full-time database engineer/implementor, core team leader
Alex Peshkov	Russian Federation	Full-time security features coordinator; buildmaster; porting authority
Claudio Valderrama	Chile	Code scrutineer; bug-finder and fixer; ISQL enhancements; UDF fixer, designer and implementor
Vladyslav Khorsun	Ukraine	Full-time DB engineer, SQL feature designer/implementor
Arno Brinkman	The Netherlands	Indexing and Optimizer enhancements; new DSQL features
Adriano dos Santos Fernandes	Brazil	New international character-set handling; text and text BLOB enhancements; new DSQL features; code scrutineering
Nickolay Samofatov	Russian Federation	Engine contributions
Roman Simakov	Russian Federation	Engine contributions
Bill Oliver	U.S.A.	Vulcan fork development, engine contributions
Paul Beach	France	Release Manager; HP-UX builds; MacOS Builds; Solaris Builds
Pavel Cisar	Czech Republic	QA tools designer/coordinator
Philippe Makowski	France	QA tester
Paul Reeves	France	Win32 installers and builds
Roman Rokytskyy	Germany	Jaybird implementor and co-coordinator
Evgeny Putilin	Russian Federation	Java stored procedures implementation
Jiri Cincura	Czech Republic	Developer and coordinator of .NET providers
Vladimir Tsvigun	Ukraine	Developer and coordinator of ODBC/JDBC driver for Firebird
Stephen Boyd	Canada	GPRE contributions

Developer	Country	Major Tasks
Paul Vinkenoog	The Nether-lands	Coordinator, Firebird documentation project; documentation writer and tools developer/implementor
Norman Dunbar	U.K.	Documentation writer
Pavel Menshchikov	Russian Federation	Documentation translator
Tomneko Hayashi	Japan	Documentation translator
Umberto (Mimmo) Masotti	Italy	Documentation translator
Helen Borrie	Australia	Release notes editor; Chief of Thought Police
also		
Université du Littoral Côte d'Opale Masters students	France	QA tests development

Appendix A: SQLSTATE

SQLSTATE Codes & Messages

In this Appendix are all of the SQLSTATE codes currently supported:

1. The 5-character SQLSTATE code returned by the status array consists of SQL CLASS (2 characters) and SQL SUBCLASS (3 characters)
2. Where existent and known, 1:1 mappings to the deprecated SQLCODE are included.
3. In many cases, SQLCODE:SQLSTATE mappings are not 1:1, which is intentional on the part of the SQL Standards committee. It has been their aim, for many years, that the use of the SQLCODE be deprecated entirely.

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
SQLCLASS 00 (Success)		
00000	Success	
SQLCLASS 01 (Warning)		
01000	General Warning	
01001	Cursor operation conflict	
01002	Disconnect error	
01003	NULL value eliminated in set function	
01004	String data, right-truncated	
01005	Insufficient item descriptor areas	
01006	Privilege not revoked	
01007	Privilege not granted	
01008	Implicit zero-bit padding	
01100	Statement reset to unprepared	
01101	Ongoing transaction has been committed	
01102	Ongoing transaction has been rolled back	
SQLCLASS 02 (No Data)		
02000	No data found or no rows affected	
SQLCLASS 07 (Dynamic SQL error)		
07000	Dynamic SQL error	
07001	Wrong number of input parameters	
07002	Wrong number of output parameters	
07003	Cursor specification cannot be executed	

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
07004	USING clause required for dynamic parameters	
07005	Prepared statement not a cursor-specification	
07006	Restricted data type attribute violation	
07007	USING clause required for result fields	
07008	Invalid descriptor count	
07009	Invalid descriptor index	
SQLCLASS 08 (Connection Exception)		
08001	Client unable to establish connection	
08002	Connection name in use	
08003	Connection does not exist	
08004	Server rejected the connection	
08006	Connection failure	
08007	Transaction resolution unknown	
SQLCLASS 0A (Feature Not Supported)		
0A000	Feature Not Supported	
SQLCLASS 0B (Invalid Transaction Initiation)		
0B000	Invalid transaction initiation	
SQLCLASS 0L (Invalid Grantor)		
0L000	Invalid grantor	
SQLCLASS 0P (Invalid Role Specification)		
0P000	Invalid role specification	
SQLCLASS 0U (Attempt to Assign to Non-Updatable Column)		
0U000	Attempt to assign to non-updatable column	
SQLCLASS 0V (Attempt to Assign to Ordering Column)		
0V000	Attempt to assign to Ordering column	
SQLCLASS 20 (Case Not Found For Case Statement)		
20000	Case not found for case statement	
SQLCLASS 21 (Cardinality Violation)		
21000	Cardinality violation	
21S01	Insert value list does not match column list	
21S02	Degree of derived table does not match column list	

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
SQLCLASS 22 (Data Exception)		
22000	Data exception	
22001	String data, right truncation	
22002	Null value, no indicator parameter	
22003	Numeric value out of range	
22004	Null value not allowed	
2205	Error in assignment	
2206	Null value in field reference	
2207	Invalid datetime format	
22008	Datetime field overflow	
22009	Invalid time zone displacement value	
2200A	Null value in reference target	
2200B	Escape character conflict	
2200C	Invalid use of escape character	
2200D	Invalid escape octet	
2200E	Null value in array target	
2200F	Zero-length character string	
2200G	Most specific type mismatch	
22010	Invalid indicator parameter value	
22011	Substring error	
22012	Division by zero	
22014	Invalid update value	
22015	Interval field overflow	
22018	Invalid character value for cast	
22019	Invalid escape character	
2201B	Invalid regular expression	
2201C	Null row not permitted in table	
22020	Invalid limit value	
22021	Character not in repertoire	
22022	Indicator overflow	
22023	Invalid parameter value	

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
22024	Character string not properly terminated	
22025	Invalid escape sequence	
22026	String data, length mismatch	
22027	Trim error	
22028	Row already exists	
2202D	Null instance used in mutator function	
2202E	Array element error	
2202F	Array data, right truncation	
SQLCLASS 23 (Integrity Constraint Violation)		
23000	Integrity constraint violation	
SQLCLASS 24 (Invalid Cursor State)		
24000	Invalid cursor state	
24504	The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row	
SQLCLASS 25 (Invalid Transaction State)		
25000	Invalid transaction state	
25	xxxx	
25S01	Transaction state	
25S02	Transaction is still active	
25S03	Transaction is rolled back	
SQLCLASS 26 (Invalid SQL Statement Name)		
26000	Invalid SQL statement name	
SQLCLASS 27 (Triggered Data Change Violation)		
27000	Triggered data change violation	
SQLCLASS 28 (Invalid Authorization Specification)		
28000	Invalid authorization specification	
SQLCLASS 2B (Dependent Privilege Descriptors Still Exist)		
2B000	Dependent privilege descriptors still exist	
SQLCLASS 2C (Invalid Character Set Name)		
2C000	Invalid character set name	
SQLCLASS 2D (Invalid Transaction Termination)		

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
2D000	Invalid transaction termination	
SQLCLASS 2E (Invalid Connection Name)		
2E000	Invalid connection name	
SQLCLASS 2F (SQL Routine Exception)		
2F000	SQL routine exception	
2F002	Modifying SQL-data not permitted	
2F003	Prohibited SQL-statement attempted	
2F004	Reading SQL-data not permitted	
2F005	Function executed no return statement	
SQLCLASS 33 (Invalid SQL Descriptor Name)		
33000	Invalid SQL descriptor name	
SQLCLASS 34 (Invalid Cursor Name)		
34000	Invalid cursor name	
SQLCLASS 35 (Invalid Condition Number)		
35000	Invalid condition number	
SQLCLASS 36 (Cursor Sensitivity Exception)		
36001	Request rejected	
36002	Request failed	
SQLCLASS 37 (Invalid Identifier)		
37000	Invalid identifier	
37001	Identifier too long	
SQLCLASS 38 (External Routine Exception)		
38000	External routine exception	
SQLCLASS 39 (External Routine Invocation Exception)		
39000	External routine invocation exception	
SQLCLASS 3B (Invalid Save Point)		
3B000	Invalid save point	
SQLCLASS 3C (Ambiguous Cursor Name)		
3C000	Ambiguous cursor name	
SQLCLASS 3D (Invalid Catalog Name)		
3D000	Invalid catalog name	

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
3D001	Catalog name not found	
SQLCLASS 3F (Invalid Schema Name)		
3F000	Invalid schema name	
SQLCLASS 40 (Transaction Rollback)		
40000	Ongoing transaction has been rolled back	
40001	Serialization failure	
40002	Transaction integrity constraint violation	
40003	Statement completion unknown	
SQLCLASS 42 (Syntax Error or Access Violation)		
42000	Syntax error or access violation	
42702	Ambiguous column reference	
42725	Ambiguous function reference	
42818	The operands of an operator or function are not compatible	
42S01	Base table or view already exists	
42S02	Base table or view not found	
42S11	Index already exists	
42S12	Index not found	
42S21	Column already exists	
42S22	Column not found	
SQLCLASS 44 (With Check Option Violation)		
44000	WITH CHECK OPTION Violation	
SQLCLASS 45 (Unhandled User-defined Exception)		
45000	Unhandled user-defined exception	
SQLCLASS 54 (Program Limit Exceeded)		
54000	Program limit exceeded	
54001	Statement too complex	
54011	Too many columns	
54023	Too many arguments	
SQLCLASS HY (CLI-specific Condition)		
HY000	CLI-specific condition	
HY001	Memory allocation error	

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
HY003	Invalid data type in application descriptor	
HY004	Invalid data type	
HY007	Associated statement is not prepared	
HY008	Operation canceled	
HY009	Invalid use of null pointer	
HY010	Function sequence error	
HY011	Attribute cannot be set now	
HY012	Invalid transaction operation code	
HY013	Memory management error	
HY014	Limit on the number of handles exceeded	
HY015	No cursor name available	
HY016	Cannot modify an implementation row descriptor	
HY017	Invalid use of an automatically allocated descriptor handle	
HY018	Server declined the cancellation request	
HY019	Non-string data cannot be sent in pieces	
HY020	Attempt to concatenate a null value	
HY021	Inconsistent descriptor information	
HY024	Invalid attribute value	
HY055	Non-string data cannot be used with string routine	
HY090	Invalid string length or buffer length	
HY091	Invalid descriptor field identifier	
HY092	Invalid attribute identifier	
HY095	Invalid FunctionId specified	
HY096	Invalid information type	
HY097	Column type out of range	
HY098	Scope out of range	
HY099	Nullable type out of range	
HY100	Uniqueness option type out of range	
HY101	Accuracy option type out of range	
HY103	Invalid retrieval code	
HY104	Invalid LengthPrecision value	

SQLSTATE Code	Mapped Message	Maps to SQLCODE..
HY105	Invalid parameter type	
HY106	Invalid fetch orientation	
HY107	Row value out of range	
HY109	Invalid cursor position	
HY110	Invalid driver completion	
HY111	Invalid bookmark value	
HYC00	Optional feature not implemented	
HYT00	Timeout expired	
HYT01	Connection timeout expired	
SQLCLASS XX (Internal Error)		
XX000	Internal error	
XX001	Data corrupted	
XX002	Index corrupted	

Appendix B: Licence Notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the “License”); you may only use this Documentation if you comply with the terms of this Licence. Copies of the Licence are available at http://www.firebirdsql.org/pdfmanual/pdl.pdf (PDF) and http://www.firebirdsql.org/manual/pdl.html (HTML).

The Original Documentation is entitled *Firebird 2.5 Release Notes*.

The Initial Writer of the Original Documentation is: Helen Borrie. Persons named in attributions are Contributors.