Firebird 2.1 Release Notes

Helen Borrie (Collator/Editor)

6 May 2008 - Document v.0210_56 - for Firebird 2.1

Firebird 2.1 Release Notes

6 May 2008 - Document v.0210_56 - for Firebird 2.1

Helen Borrie (Collator/Editor)

Table of Contents

List of Tables

Chapter 1

General Notes

Firebird 2.1 is a full version release that builds on the architectural changes introduced in the V.2.0 series. Thanks to all who have field-tested the Alphas and Betas during 2007 and the first quarter of 2008 we have a release that is bright with new features and improvements, including the long-awaited global temporary tables, a catalogue of new run-time monitoring mechanisms, database triggers and the injection of dozens of internal functions into the SQL language set.

About These Notes

This edition of the v.2.1 release notes is a merger of the notes developed over the course of developments and releases from V.2.0 to this v.2.1 release. Note that the separate v.2.0.x documentation is not distributed with the v.2.1 binaries.

The Installation, Migration/Compatibility and Bug Fixes sections have been removed from the release notes document and are now distributed in the `$fbroot$/doc/` subdirectory as separate documents. Like the release notes, they cover both the v.2.0.x and v.2.1 release series.

To help smoothe the transition from older versions, it will be essential to study both the release notes and the migration/installation guide thoroughly. We encourage you to take your time testing this release with your applications and stressing it with real-world data and loads. Some legacy queries might not work, or might not perform as they did previously, due to a number of logic corrections that have been implemented. Temporary workarounds for legacy applications in such situations are available in some cases. We prefer that you research such issues in the documentation before you consider posting support questions or bug reports about them.

From the QA Team

Although this is a designated stable release, intended for production environments, it does introduce much that is new. We encourage you to see what you can achieve with these new features and let us know about any deficiency, sooner rather than later.

You are enthusiastically invited to post to the firebird-devel list good descriptions of any bugs or beasts you encounter, or post bug reports directly to our Issue Tracker. Regular sub-releases will follow, whose quality and timeliness depend heavily on the reports that come in “from the field”.

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 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 release.

Additional Documentation

You will find README documents for many of the new v.2 and v.2.1 features in your installation kit, installed by default in the /doc/ sub-directory.

An automated "Release Notes" page in the Tracker provides access to lists and links for all of the Tracker tickets associated with this version and its various builds. Use this link.

For your convenience, the many bug-fixes and regressions fixed during the development of Firebird 2.0.x and 2.1 are listed in descending chronological order in the separate Bugfixes document.

—*Firebird 2 Project Team*

Chapter 2

New in Firebird 2

New Features Implemented

This chapter summarises the new features implemented in Firebird 2, encompassing both v.2.1 and the v.2.0.x series.

On-Disk Structure

Databases created or restored under Firebird 2 have an on-disk structure (ODS) of 11 or higher.

- Firebird 2.1 creates databases with an ODS of 11.1. It can read databases of lower ODS but most of its new features will be unavailable to such databases.
- Firebird 2.0.x servers create databases with an ODS of 11 (sometimes expressed as 11.0). If you wish to have the full range of v.2.1 features available, you should upgrade ODS 11 and lower databases by backing them up and restoring them under v.2.1.

Database Triggers

(v.2.1) Newly implemented “database triggers” are user-defined PSQL modules that can be designed to fire in various connection-level and transaction- level events. See Database Triggers.

SQL and Objects

Global Temporary Tables

(v.2.1) SQL standards-compliant global temporary tables have been implemented. These pre-defined tables are instantiated on request for connection-specific or transaction-specific use with non-persistent data, which the Firebird engine stores in temporary files. See Global Temporary Tables.

Common Table Expressions, Recursive DSQL Queries

(v.2.1) Standards-compliant common table expressions, which make dynamic recursive queries possible, are introduced. See Common Table Expressions.

RETURNING Clause

(v.2.1) Optional RETURNING clause for all singleton operations update, insert and delete operations. See RETURNING Clause.

UPDATE OR INSERT Statements

(v.2.1) Now you can write a statement that is capable of performing either an update to an existing record or an insert, depending on whether the targeted record exists. See UPDATE OR INSERT Statement.

MERGE Statement

(v.2.1) New statement syntax that performs either an update to an existing record if a condition is met or an insert if the condition is not met. See MERGE Statement.

LIST() function

(v.2.1) A new aggregate function LIST(<SOMETHING>) retrieves all of the SOMETHINGs in a group and aggregates them into a comma-separated list. See LIST Function.

Lots of New Built-in Functions

(v.2.1) Built-in functions replacing many of the UDFs from the Firebird- distributed UDF libraries. For a full list with examples, see Built-in Functions.

"Short" BLOBs Can Masquerade as Long VARCHARs

(v.2.1) At various levels of evaluation, the engine now treats text BLOBs that are within the 32,765-byte size limit as though they were varchars. Now functions like cast, lower, upper, trim and substring will work with these BLOBs, as well as concatenation and assignment to string types. See Text BLOB Compatibility.

Procedural SQL

Domains for Defining PSQL Variables and Arguments

(v.2.1) PSQL local variables and input and output arguments for stored procedures can now be declared using domains in lieu of canonical data types. See Domains in PSQL.

COLLATE in Stored Procedures and Parameters

(v.2.1) Collations can now be applied to PSQL variables and arguments. See COLLATE in Stored Procedures.

Enhancement to PSQL error stack trace

V. Khorsun

Feature request CORE-970

(v.2.1) A PSQL error stack trace now shows line and column numbers.

Security

Windows Security to Authenticate Users

(v.2.1) Windows “Trusted User” security can be applied for authenticating Firebird users on a Windows host. See Windows Trusted User Security.

International Language Support

The CREATE COLLATION Command

(v.2.1) The DDL command `CREATE COLLATION` has been introduced for implementing a collation, obviating the need to use the script for it. See CREATE COLLATION statement.

Unicode Collations Anywhere

(v.2.1) Two new Unicode collations can be applied to any character set using a new mechanism. See UNICODE Collations.

Platform Support

Ports to Windows 2003 64-bit

D. Yemanov

Feature request CORE-819 and CORE-682

(v.2.1) 64-bit Windows platform (AMD64 and Intel EM64T) ports of Classic, Superserver and Embedded models.

Administrative

Database Monitoring via SQL

(v.2.1) Implementation of run-time database snapshot monitoring (transactions, tables, etc.) via SQL over some new virtualized system tables. See Monitoring Tables.

Included in the set of tables is one named MON$DATABASE that provides a lot of the database header information that could not be obtained previously via SQL: such details as the on-disk structure (ODS) version, SQL dialect, sweep interval, OIT and OAT and so on.

It is possible to use the information from the monitoring tables to cancel a rogue query. See Cancel a Running Query.

More Context Information

Context information providing the server engine version has been added, for retrieving via SELECT calls to the RDB$GET_CONTEXT function. See More Context Information.

New Command-line Utility fbsvcmgr

(V.2.1) The new utility *fbsvcmgr* provides a command-line interface to the Services API, enabling access to any service that is implemented in Firebird.

Although there are numerous database administration tools around that surface the Services API through graphical interfaces, the new tool addresses the problem for admins needing to access remote Unix servers in broad networks through a text-only connection. Previously, meeting such a requirement needed a programmer. Details here.

Remote Interface

(v.2.1) The remote protocol has been slightly improved to perform better in slow networks once drivers are updated to utilise the changes. Testing showed that API round trips were reduced by about 50 percent, resulting in about 40 per cent fewer TCP round trips. See Remote Interface Improvement.

Derived Tables

A. Brinkman

Implemented support for derived tables in DSQL (subqueries in FROM clause) as defined by SQL200X. A derived table is a set, derived from a dynamic SELECT statement. Derived tables can be nested, if required, to build complex queries and they can be involved in joins as though they were normal tables or views.

More details under Derived Tables in the DML chapter.

PSQL Now Supports Named Cursors

D. Yemanov

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

Reimplemented Protocols on Windows

D. Yemanov

Two significant changes have been made to the Windows-only protocols.-

Local Protocol--XNET

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

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

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

Benefits of the XNET Protocol over IPServer

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

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

Performance

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

Disadvantages

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

Change to WNET ("NetBEUI") Protocol

WNET (a.k.a. NetBEUI) protocol no longer performs client impersonation.

In all previous Firebird versions, remote requests via WNET are performed in the context of the *client security token*. Since the server serves every connection according to its client security credentials, this means that, if the client machine is running some OS user from an NT domain, that user should have appropriate permissions to access the physical database file, UDF libraries, etc., on the server filesystem. This situation is contrary to what is generally regarded as proper for a client-server setup with a protected database.

Such impersonation has been removed in Firebird 2.0. WNET connections are now truly client-server and behave the same way as TCP ones, i.e., with no presumptions with regard to the rights of OS users.

Reworking of Garbage Collection

V. Khorsun

Since Firebird 1.0 and earlier, the Superserver engine has performed *background garbage collection*, maintaining information about each new record version produced by an UPDATE or DELETE statement. As soon as the

old versions are no longer “interesting”, i.e. when they become older than the Oldest Snapshot transaction (seen in the *gstat -header* output) the engine signals for them to be removed by the garbage collector.

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

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

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

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

- cooperative - garbage collection will be performed only in cooperative mode (like Classic) and the engine will not track old record versions. This reverts GC behaviour to that of IB 5.6 and earlier. It is the only option for Classic.
- background - garbage collection will be performed only by background threads, as is the case for Firebird 1.5 and earlier. User table-scan requests will not remove unused record versions but will cause the GC thread to be notified about any page where an unused record version is detected. The engine will also remember those page numbers where UPDATE and DELETE statements created back versions.
- combined (the installation default for Superserver) - both background and cooperative garbage collection are performed.

> **Note**
>
> The Classic server ignores this parameter and always works in “cooperative” mode.

Porting of the Services API to Classic is Complete

N. Samofatov

Porting of the Services API to Classic architecture is now complete. All Services API functions are now available on both Linux and Windows Classic servers, with no limitations. Known issues with *gsec* error reporting in previous versions of Firebird are eliminated.

Lock Timeout for WAIT Transactions

A. Karyakin, D. Yemanov

All Firebird versions provide two transaction wait modes: *NO WAIT* and *WAIT*. NO WAIT mode means that lock conflicts and deadlocks are reported immediately, while WAIT performs a blocking wait which times out only when the conflicting concurrent transaction ends by being committed or rolled back.

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

Timeout intervals are specified per transaction, using the new TPB constant *isc_tpb_lock_timeout* in the API or, in DSQL, the *LOCK TIMEOUT <value>* clause of the SET TRANSACTION statement.

New Implementation of String Search Operators

N. Samofatov

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

Reworking of Updatable Views

D. Yemanov

A reworking has been done to resolve problems with views that are implicitly updatable, but still have update triggers. This is an important change that will affect systems written to take advantage of the undocumented [mis]behaviour in previous versions.

For details, see the notes in the Compatibility chapter of the separate Installation Notes document.

Additional Database Shutdown Modes Introduced

N. Samofatov

Single-user and full shutdown modes are implemented using new *[state]* parameters for the `gfix -shut` and `gfix -online` commands.

Syntax Pattern

```
gfix <command> [<state>] [<options>]
<command>> ::= {-shut | -online}
<state> ::= {normal | multi | single | full}
<options> ::= {-force <timeout> | -tran | -attach}
```

- *normal* state = online database
- *multi* state = multi-user shutdown mode (the legacy one, unlimited attachments of SYSDBA/owner are allowed)
- *single* state = single-user shutdown (only one attachment is allowed, used by the restore process)
- *full* state = full/exclusive shutdown (no attachments are allowed)

For more details, refer to the section on Gfix New Shutdown Modes, in the Utilities chapter.

For a list of shutdown state flag symbols and an example of usage, see Shutdown State in the API.

UDFs Improved re NULL Handling

C. Valderrama

Signalling SQL NULL

- Ability to signal SQL NULL via a NULL pointer (see Signal SQL NULL in UDFs).
- External function library ib_udf upgraded to allow the string functions ASCII_CHAR, LOWER, LPAD, LTRIM, RPAD, RTIM, SUBSTR and SUBSTRLEN to return NULL and have it interpreted correctly.

  The script `ib_udf_upgrade.sql` can be applied to pre-v.2 databases that have these functions declared, to upgrade them to work with the upgraded library. This script should be used only when you are using the new ib_udf library with Firebird v2 and operation requests are modified to anticipate nulls.

Run-time Checking for Concatenation Overflow

D. Yemanov

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

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

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

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

Changes to Synchronisation Logic

N. Samofatov

1. Lock contention in the lock manager and in the SuperServer thread pool manager has been reduced significantly
2. A rare race condition was detected and fixed, that could cause Superserver to hang during request processing until the arrival of the next request
3. Lock manager memory dumps have been made more informative and OWN_hung is detected correctly
4. Decoupling of lock manager synchronization objects for different engine instances was implemented

Support for 64-bit Platforms

A. Peshkov, N. Samofatov

Firebird 2.0 will support 64-bit platforms.

Record Enumeration Limits Increased

N. Samofatov

40-bit (64-bit internally) record enumerators have been introduced to overcome the ~30GB table size limit imposed by 32-bit record enumeration.

Debugging Improvements

Various Contributors

Improved Reporting from Bugchecks

BUGCHECK log messages now include file name and line number. (A. Brinkman)

Updated Internal Structure Reporting

Routines that print out various internal structures (DSQL node tree, BLR, DYN, etc) have been updated. (N. Samofatov)

New Debug Logging Facilities

Thread-safe and signal-safe debug logging facilities have been implemented. (N. Samofatov)

Diagnostic Enhancement

Syslog messages will be copied to the user's tty if a process is attached to it. (A. Peshkov)

Improved Connection Handling on POSIX Superserver

A. Peshkov

Posix SS builds now handle SIGTERM and SIGINT to shutdown all connections gracefully. (A. Peshkov)

PSQL Invariant Tracking Reworked

N. Samofatov

Invariant tracking in PSQL and request cloning logic were reworked to fix a number of issues with recursive procedures, for example SF bug #627057.

Invariant tracking is the process performed by the BLR compiler and the optimizer to decide whether an "invariant" (an expression, which might be a nested subquery) is independent from the parent context. It is used to perform one-time evaluations of such expressions and then cache the result.

If some invariant is not determined, we lose in performance. If some variant is wrongly treated as invariant, we see wrong results.

Example

```
select * from rdb$relations
  where rdb$relation_id <
    ( select rdb$relation_id from rdb$database )
```

This query performs only one fetch from rdb$database instead of evaluating the subquery for every row of rdb $relations.

ROLLBACK RETAIN Syntax Support

D. Yemanov

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

See ROLLBACK RETAIN Syntax in the chapter about DML.

No More Registry Search on Win32 Servers

D. Yemanov

The root directory lookup path has changed so that server processes on Windows no longer use the Registry.

> **Important**
>
> The command-line utilities still check the Registry.

More Optimizer Improvements

A. Brinkman

Better cost-based calculation has been included in the optimizer routines.

Chapter 3

Changes to the Firebird API and ODS

API (Application Programming Interface)

Some needed changes have been performed in the Firebird API. They include.-

Cleanup of ibase.h

D. Yemanov, A. Peshkov

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

Lock Timeout for WAIT Transactions

A. Karyakin, D. Yemanov

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

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

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

isc_dsql_sql_info() Now Includes Relation Aliases

D. Yemanov

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

Enhancement to isc_blob_lookup_desc()

A. dos Santos Fernandes

isc_blob_lookup_desc() now also describes blobs that are outputs of stored procedures

API Identifies Client Version

N. Samofatov

The macro definition `FB_API_VER` is added to ibase.h to indicate the current API version. The number corresponds to the appropriate Firebird version.

The current value of FB_API_VER is 20 (two-digit equivalent of 2.0). This macro can be used by client applications to check the version of ibase.h its being compiled with.

Additions to the isc_database_info() Structure

V. Khorsun

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

isc_info_active_tran_count

Returns the number of currently active transactions.

isc_info_creation_date

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

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

Additions to the isc_transaction_info() Structure

V. Khorsun

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

isc_info_tra_oldest_interesting

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

isc_info_tra_oldest_active

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

isc_info_tra_oldest_snapshot

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

> **Note**
>
> This value is used as the threshold ("high-water mark") for garbage collection.

isc_info_tra_isolation

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

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

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

isc_info_tra_access

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

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

isc_info_tra_lock_timeout

Returns the lock timeout set for the current transaction.

Improved Services API

The following improvements have been added to the Services API:

Parameter isc_spb_trusted_auth

(V.2.1, ODS 11.1) `isc_spb_trusted_auth` applies only to Windows and is used to force Firebird to use Windows trusted authentication for the requested service.

Parameter isc_spb_dbname

(V.2.1, ODS 11.1) For any services related to the security database, provides the ability to supply the name of the security database when invoking a security service remotely. It is equivalent to supplying the `-database` switch when calling the *gsec* utility remotely.

Task Execution Optimized

D. Yemanov

Services are now executed as threads rather than processes on some threadable CS builds (currently 32- bit Windows and Solaris).

New Function for Delivering Error Text

C. Valderrama

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

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

Accommodation of New Shutdown <state> Parameters

D. Yemanov

API Access to database shutdown is through flags appended to the isc_dpb_shutdown parameter in the DBP argument passed to isc_attach_database(). The symbols for the <state> flags are:

```
#define isc_dpb_shut_cache              0x1
#define isc_dpb_shut_attachment         0x2
#define isc_dpb_shut_transaction        0x4
#define isc_dpb_shut_force              0x8
#define isc_dpb_shut_mode_mask         0x70

#define isc_dpb_shut_default            0x0
#define isc_dpb_shut_normal            0x10
#define isc_dpb_shut_multi             0x20
#define isc_dpb_shut_single            0x30
#define isc_dpb_shut_full              0x40
```

Example of Use in C/C++

```
char dpb_buffer[256], *dpb, *p;
ISC_STATUS status_vector[ISC_STATUS_LENGTH];
isc_db_handle handle = NULL;
```

```
dpb = dpb_buffer;

*dpb++ = isc_dpb_version1;

const char* user_name = “SYSDBA”;
const int user_name_length = strlen(user_name);
*dpb++ = isc_dpb_user_name;
*dpb++ = user_name_length;
memcpy(dpb, user_name, user_name_length);
dpb += user_name_length;

const char* user_password = “masterkey”;
const int user_password_length = strlen(user_password);
*dpb++ = isc_dpb_password;
*dpb++ = user_password_length;
memcpy(dpb, user_password, user_password_length);
dpb += user_password_length;

// Force an immediate full database shutdown
*dpb++ = isc_dpb_shutdown;
*dpb++ = isc_dpb_shut_force | isc_dpb_shut_full;

const int dpb_length = dpb - dpb_buffer;

isc_attach_database(status_vector,
                    0, “employee.db”,
                    &handle,
                    dpb_length, dpb_buffer);

if (status_vector[0] == 1 && status_vector[1])
{
  isc_print_status(status_vector);
}
else
{
  isc_detach_database(status_vector, &handle);
}
```

ODS (On-Disk Structure) Changes

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

New ODS Number

Firebird 2.0 creates databases with an ODS (On-Disk Structure) version of 11.

Size limit for exception messages increased

V. Khorsun

Maximum size of exception messages raised from 78 to 1021 bytes.

New Description Field for Generators

C. Valderrama

Added RDB$DESCRIPTION to RDB$GENERATORS, so now you can include description text when creating generators.

New Description Field for SQL Roles

C. Valderrama

Added RDB$DESCRIPTION and RDB$SYSTEM_FLAG to RDB$ROLES to allow description text and to flag user-defined roles, respectively.

"ODS Type" Recognition

N. Samofatov

Introduced a concept of ODS type to distinguish between InterBase and Firebird databases.

Smarter DSQL Error Reporting

C. Valderrama

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

New Column in RDB$Index_Segments

D. Yemanov, A. Brinkman

A new column RDB$STATISTICS has been added to the system table RDB$INDEX_SEGMENTS to store the per-segment selectivity values for multi-key indexes.

> **Note**
>
> The column of the same name in RDB$INDICES is kept for compatibility and still represents the total index selectivity, that is used for a full index match.

Chapter 4

Global Improvements in Firebird 2.1

Some global improvements and changes have been implemented in Firebird 2.1, as engine development moves towards the architectural changes planned for Firebird 3.

> **Note**
>
> Unless otherwise indicated, these improvements apply from v.2.1 forward.

Forced Writes on Linux Now Works!

A. Peshkov

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

Its implementation on Linux was very simple - invoke **fcntl(dbFile, F_SETFL, O_SYNC)**.

Yet databases on Linux were sometimes corrupted anyway.

Forensics

Speed tests on Linux showed that setting O_SYNC on a file has no effect at all on performance! Fine, fast operating system we may think? Alas, no, it's a documented bug in the Linux kernel!

According to the Linux manual, “On Linux this command (i.e. fcntl(fd, F_SETFL, flags)) can only change the O_APPEND, O_ASYNC, O_DIRECT, O_NOATIME, and O_NONBLOCK flags”. Though it is not documented in any place known to me, it turns out that an attempt to set any flag other than those listed in the manual (such as O_SYNC, for example) won't work but it does not cause fcntl() to return an error, either.

For Firebird and for InterBase versions since Day One, it means that Forced Writes has never worked on Linux. It certainly works on Windows. It seems likely that this is not a problem that affects other operating systems, although we cannot guarantee that. To make sure, you can check whether the implementation of **fcntl()** on your OS is capable of setting the O_SYNC flag.

The technique used currently, introduced in the Beta 2 release of Firebird 2.1, is to re-open the file. It should guarantee correct operation on any OS, provided the **open()** system call works correctly in this respect. It appears that no such problems are reported.

The Firebird developers have no idea why such a bug would remain unfixed almost two years after getting into the Linux kernel's bug-tracker. Apparently, in Linux, a documented bug evolves into a feature...

Instant Fix for an Older Firebird

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

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

Databases on Raw Devices

A. Peshkov

File system I/O can degrade performance severely when a database in Forced Writes mode grows rapidly. On Linux, which lacks the appropriate system calls to grow the database efficiently, performance with Forced Writes can be as much as three times slower than with asynchronous writes.

When such conditions prevail, performance may be greatly enhanced by bypassing the file system entirely and restoring the database directly to a raw device. A Firebird database can be recreated on any type of block device.

Moving a Database to a Raw Device

Moving your database to a raw device can be as simple as restoring a backup directly to an unformatted partition in the local storage system. For example,

```
gbak -c my.fbk /dev/sda7
```

will restore your database on the third logical disk in the extended partition of your first SCSI or SATA hard-drive (disk0).

> **Note**
>
> The database does not have a “database name” other than the device name itself. In the example given, the name of the database is '/dev/sda7'.

Special Issues for nbak/nbackup

The physical backup utility **nbackup** must be supplied with an explicit file path and name for its difference file, in order to avoid this file being written into the /dev/ directory. You can achieve this with the following statement, using isql:

```
# isql /dev/sda7
```

```
SQL> alter database add difference file '/tmp/dev_sda7';
```

To keep the size of the nbak copy within reasonable bounds, it is of benefit to know how much storage on the device is actually occupied. The '-s' switch of nbackup will return the size of the database *in database pages*:

```
# nbackup -s -l /dev/sda7
77173
```

Don't confuse the result here with the block size of the device. The figure returned—77173—is the number of pages occupied by the database. Calculate the physical size (in bytes) as (number of pages * page size). If you are unsure of the page size, you can query it from the database header using gstat -h:

```
 # gstat -h /dev/sda7
 Database "/dev/sda7"
 Database header page information:
        Flags               0
        Checksum            12345
        Generation          43
        Page size           4096  <———
        ODS version         11.1
  . . . . . .
```

Examples of nbackup Usage with a Raw Device

1. A backup can be performed in a script, using the output from the '-s' switch directly. For example,

```
# DbFile=/dev/sda7
# DbSize=`nbackup -L $DbFile -S` || exit 1
# dd if=$DbFile ibs=4k count=$DbSize | # compress and record DVD
# nbackup -N $DbFile
```

2. A physical backup using nbackup directly from the command line:

```
# nbackup -B 0 /dev/sda7 /tmp/lvl.0
```

Further Advice about Raw Devices

Although no other specific issues are known at this point about the use of raw device storage for databases, keep in mind that

- the growth and potential growth of the database is less obvious to end-users than one that lives as a file within a file system. If control of the production system's environment is out of your direct reach, be certain to deploy adequate documentation for any monitoring that will be required!

- the very Windows-knowledgeable might want to try out the concept of raw device storage on Windows systems. It has not been a project priority to explore how it might be achieved on that platform. However, if you think you know a way to do it, please feel welcome to test the idea in your Windows lab and report your observations—good or bad or indifferent—back to the firebird-devel list.

- **Tip**

  Maintain your raw devices in aliases.conf. That way, in the event of needing to reconfigure the storage hardware, there will be no need to alter any connection strings in your application code.

Remote Interface Improvements

V. Khorsun, D. Yemanov

Feature request CORE-971

The remote protocol has been slightly improved to perform better in slow networks. In order to achieve this, more advanced packets batching is now performed, along with some buffer transmission optimizations. In a real world test scenario, these changes showed about 50 per cent fewer API round trips, thus incurring about 40 per cent fewer TCP roundtrips.

In Firebird 2.1 the remote interface limits the packet size of the response to various isc_XXX_info calls to the real used length of the contained data, whereas before it sent the full specified buffer back to the client buffer, even if only 10 bytes were actually filled. Firebird 2.1 remote interface sends back only 10 bytes in this case.

Some of our users should see a benefit from the changes, especially two-tier clients accessing databases over the Internet.

The changes can be summarised as

a. Batched packets delivery. Requires both server and client of version v2.1, enabled upon a successful protocol handshake. Delays sending packets of certain types which can be deferred for batched transfer with the next packet. (Allocate/deallocate statement operations come into this category, for example.)

b. Pre-fetching some pieces of information about a statement or request and caching them on the client side for (probable) following API calls. Implemented on the client side only, but relies partly on the benefits of reduced round trips described in (a).

   It works with any server version, even possibly providing a small benefit for badly written client applications, although best performance is not to be expected if the client is communicating with a pre-V.2.1 server.

c. Reduced information responses from the engine (no trailing zeroes). As the implementation is server-side only, it requires a V.2.1 server and any client. Even old clients will work with Firebird 2.1 and see some benefit from the reduction of round trips, although the old remote interface, unlike the new, will still send back big packets for isc_dsql_prepare().

d. Another round-trip saver, termed “defer execute”, whereby SELECT requests will be held at the point just before execution of the `isc_dsql_execute` until the next API call on the same statement. The benefit of the saved round-trip becomes most visible where there is a bunch of SELECT requests whose result set fits into one or two network packets.

   This enhancement takes effect only if both client and server are v.2.1 or higher.

> **Note**
>
> A faintly possible side-effect is that, if `isc_dsql_execute` should happen to fail with a certain exception, this exception is returned to the client in the response to the API call that was *actually* responsible; i.e., instead of being returned by `isc_dsql_execute` it would be returned by `isc_dsql_fetch`, `isc_dsql_info`, or whichever API call actually dispatched the `op_execute` call.
>
> In most cases, the side-effect would be transparent: it might show up in a case where some error occurred with default values for PSQL parameters or variables and would be noticed as an exception array where the exceptions were delivered in an unusual sequence.

The changes work with either TCP/IP or NetBEUI. They are backward-compatible, so existing client code will not be broken. However, when you are using a driver layer that implements its own interpretation of the remote protocol—such as the Jaybird JDBC and the FirebirdClient .NET drivers—your existing code will not enable the enhancements unless you usedrivers are updated.

API Changes

XSQLVAR

A. dos Santos Fernandes

The identifier of the connection character set or, when the connection character set is NONE, the BLOB character set, is now passed in the `XSQLVAR::sqlscale` item of text BLOBs.

Optimization

Optimization for Multiple Index Scans

V. Khorsun

Feature request CORE-1069

An optimization was done for index scanning when more than one index is to be scanned with AND conjunctions.

Optimize sparse bitmap operations

V. Khorsun

Feature request CORE-1070

Optimization was done for sparse bitmap operations (set, test and clear) when values are mostly consecutive.

Configuration and Tuning

Increased Lock Manager Limits & Defaults

D. Yemanov

Feature requests CORE-958 and CORE-937

- the **maximum number of hash slots** is raised from 2048 to 65,536. Because the actual setting should be a prime number, the exact supported maximum is 65,521 (the biggest prime number below 65,536). The minimum is 101.
- the new **default number of hash slots** is 1009
- the default **lock table size** has been increased to 1 Mb on all platforms

Page sizes of 1K and 2K Deprecated

D. Yemanov

Feature request CORE-969

Page sizes of 1K and 2K are deprecated as inefficient.

> **Note**
>
> The small page restriction applies to new databases only. Old ones can be attached to regardless of their page size.

Enlarge Disk Allocation Chunks

V. Khorsun

Feature request CORE-1229

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

For databases using ODS 11.1 and higher, Firebird servers from v.2.1 onward use a different algorithm for allocating disk space, to address two recognised problems associated with the existing approach:

1. **Corruptions resulting from out-of-space conditions on disk**

   The indeterminate order of writes can give rise to a situation that, at a point where the page cache contains a large number of dirty pages and Firebird needs to request space for a new page in the process of writing them out, there is insufficient disk space to fulfil the request. Under such conditions it often happens that the administrator decides to shut down the database in order to make some more disk space available, causing the remaining dirty pages in the cache to be lost. This leads to serious corruptions.

2. **File fragmentation**

   Allocating disk space in relatively small chunks can lead to significant fragmentation of the database file at file system level, impairing the performance of large scans, as during a backup, for example.

The Solution

The solution is to introduce some rules and rationales to govern page writes according to the state of available disk space, as follows.-

a. Each newly allocated page writes to disk immediately before returning to the engine. If the page cannot be written then the allocation does not happen: the PIP bit remains uncleared and the appropriate I/O error is raised. Corruption cannot arise, since it is guaranteed that all dirty pages in cache have disk space allocated and can be written safely.

   Because this change adds an extra write for each newly-allocated page, some performance penalty is to be expected. To mitigate the effect, writes of newly-allocated pages are performed in batches of up to 128 Kb and Firebird keeps track of the number of these “initialized” pages in the PIP header.

> **Note**
>
> A page that has been allocated, released and re-allocated is already “space in hand”, meaning that no further verification is required in order to “initialize” it. Hence, a newly allocated page is subjected to this double-write only if it is a block that has never been allocated before.

b. To address the issue of file fragmentation, Firebird now uses the appropriate call to the API of the file system to *preallocate* disk space in relatively large chunks.

   Preallocation also gives room to avoid corruptions in the event of an “out of disk space” condition. Chances are that the database will have enough space preallocated to continue operating until the administrator can make some disk space available.

> **Important**
>
> **Windows Only (for Now)**
>
> Currently, only Windows file systems publish such API calls, which means that, for now, this aspect of the solution is supported only in the Windows builds of Firebird. However, similar facilities have recently been added to the Linux API, allowing the prospect that a suitable API function call will appear in such popular file systems as *ext3* in future.

DatabaseGrowthIncrement Configuration Parameter

For better control of disk space preallocation, the new parameter *DatabaseGrowthIncrement* has been added to `firebird.conf`. It represents the upper limit for the preallocation chunk size in bytes.

> **Important**
>
> Please be sure to read the details regarding this configuration, under DatabaseGrowthIncrement in the chapter entitled “New Configuration Parameters and Changes”.

Bypass Filesystem Caching on Superserver

V. Khorsun

Feature requests CORE-1381 and CORE-1480

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

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

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

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

> **Note**
>
> For Classic, there is no escaping filesystem caching.

For details of the **MaxFileSystemCache** parameter, see MaxFileSystemCache.

Other Global Improvements

Garbage Collector Rationalisation

V. Khorsun

Feature request CORE-1071

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

Immediate Release of External Files

V. Khorsun

Feature request CORE- 961

The engine will now release external table files as soon as they are no longer in use by user requests.

Synchronization of DSQL metadata cache objects in Classic server

A. dos Santos Fernandes

Feature request CORE-976

No details.

BLOB Improvements

A. dos Santos Fernandes

Feature request CORE-1169

Conversion of temporary blobs to the destination blob type now occurs when materializing.

Type Flag for Stored Procedures

D. Yemanov

Feature request CORE-779

Introduced a type flag for stored procedures, adding column RDB$PROCEDURE_TYPE to the table RDB $PROCEDURES. Possible values are:

- 0 or NULL -
legacy procedure (no validation checks are performed)

- 1 -
selectable procedure (one that contains a SUSPEND statement)

- 2 -
executable procedure (no SUSPEND statement, cannot be selected from)

Help for Getting Core Dumps on Linux

A. Peshkov

Feature request CORE-1558

The configuration parameter BugcheckAbort provides the capability to make the server stop trying to continue operation after a bugcheck and instead, to call **abort()** immediately and dump a core file. Since a bugcheck usually occurs as a result of a problem the server does not recognise, continuing operation with an unresolved problem is not usually possible anyway, and the core dump can provide useful debug information.

In the more recent Linux distributions the default setups no longer dump core automatically when an application crashes. Users often have troubles trying to get them working. Differing rules for Classic and Superserver, combined with a lack of consistency between the OS setup tools from distro to distro, make it difficult to help out with any useful “general rule”.

Code has been added for Classic and Superserver on Linux to bypass these problems and automate generation of a core dump file when an abort() on BUGCHECK occurs. The Firebird server will make the required 'cwd' (change

working directory) to an appropriate writable location (/tmp) and set the core file size limit so that the 'soft' limit equals the 'hard' limit.

> **Note**
>
> In a release version, the automated core-dumping is active only when the BugcheckAbort parameter in firebird.conf is set to true (1). In a debug version, it is always active.
>
> If you need to enable the facility, don't forget that the server needs to be restarted to activate a parameter change.

Chapter 5

Data Definition Language (DDL)

In this chapter are the additions and improvements that have been added to the SQL data definition language subset in the Firebird 2 development cycle. Those marked as introduced in v.2.1 are available only to ODS 11.1 and higher databases.

Quick Links

Database Triggers

Adriano dos Santos Fernandes

(v.2.1) A *database trigger* is a PSQL module that is executed when a connection or transaction event occurs. The events and the timings of their triggers are as follows.-

`CONNECT`

- Database connection is established
- A transaction is started
- Triggers are fired; uncaught exceptions roll back the transaction, disconnect the attachment and are returned to the client
- The transaction is committed

`DISCONNECT`

- A transaction is started

- Triggers are fired; uncaught exceptions roll back the transaction, disconnect the attachment and are swallowed
- The transaction is committed
- The attachment is disconnected

TRANSACTION START
: Triggers are fired in the newly-created user transaction; uncaught exceptions are returned to the client and the transaction is rolled back.

TRANSACTION COMMIT
: Triggers are fired in the committing transaction; uncaught exceptions roll back the trigger's savepoint, the commit command is aborted and the exception is returned to the client.

> **Note**
>
> For two-phase transactions, the triggers are fired in the “prepare”, not in the commit.

TRANSACTION ROLLBACK
: Triggers are fired during the roll-back of the transaction. Changes done will be rolled back with the transaction. Exceptions are swallowed

Syntax

```
<database-trigger> ::=
  {CREATE | RECREATE | CREATE OR ALTER}
    TRIGGER <name>
    [ACTIVE | INACTIVE]
    ON <event>
    [POSITION <n>]
  AS
    BEGIN
      ...
    END

<event> ::=
  CONNECT
    | DISCONNECT
    | TRANSACTION START
    | TRANSACTION COMMIT
    | TRANSACTION ROLLBACK
```

Rules and Restrictions

1. Database triggers type cannot be changed.
2. Permission to create, recreate, create or alter, or drop database triggers is restricted to the database owner and SYSDBA.

Utilities Support for Database Triggers

New parameters were added to gbak, nbackup and isql to suppress database triggers from running. They are available only to the database owner and SYSDBA:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Global Temporary Tables

Vlad Khorsun

(v.2.1) Global temporary tables (GTTs) are tables that are stored in the system catalogue with permanent metadata, but with temporary data. Data from different connections (or transactions, depending on the scope) are isolated from each other, but the metadata of the GTT are shared among all connections and transactions.

There are two kinds of GTT:

- with data that persists for the lifetime of connection in which the specified GTT was referenced; and
- with data that persists only for the lifetime of the referencing transaction.

Syntax and Rules for GTTs

```
CREATE GLOBAL TEMPORARY TABLE
  ...
  [ON COMMIT <DELETE | PRESERVE> ROWS]
```

Creates the metadata for the temporary table in the system catalogue.

The clause ON COMMIT sets the kind of temporary table:

ON COMMIT PRESERVE ROWS
: Data left in the given table after the end of the transaction remain in database until the connection ends.

ON COMMIT DELETE ROWS
: Data in the given table are deleted from the database immediately after the end of the transaction. ON COMMIT DELETE ROWS is used by default if the optional clause ON COMMIT is not specified.

CREATE GLOBAL TEMPORARY TABLE
: is a regular DDL statement that is processed by the engine the same way as a CREATE TABLE statement is processed. Accordingly, it not possible to create or drop a GTT within a stored procedure or trigger.

Relation Type

GTT definitions are distinguished in the system catalogue from one another and from permanent tables by the value of `RDB$RELATIONS.RDB$RELATION_TYPE`:

- A GTT with ON COMMIT PRESERVE ROWS option has RDB$RELATION_TYPE = 4

  A GTT with ON COMMIT DELETE ROWS option has RDB$RELATION_TYPE = 5.

> **Note**
>
> For the full list of values, see RDB$TYPES.

Structural Feature Support

The same structural features that you can apply to regular tables (indexes, triggers, field-level and table level constraints) are also available to a GTT, with certain restrictions on how GTTs and regular tables can interrelate.-

a. references between persistent and temporary tables are forbidden

b. A GTT with ON COMMIT PRESERVE ROWS cannot have a reference on a GTT with ON COMMIT DELETE ROWS

c. A domain constraint cannot have a reference to any GTT.

Implementation Notes

An instance of a GTT—a set of data rows created by and visible within the given connection or transaction—is created when the GTT is referenced for the first time, usually at statement prepare time. Each instance has its own private set of pages on which data and indexes are stored. The data rows and indexes have the same physical storage layout as permanent tables.

When the connection or transaction ends, all pages of a GTT instance are released immediately. It is similar to what happens when a DROP TABLE is performed, except that the metadata definition is retained, of course. This is much quicker than the traditional row-by-row delete + garbage collection of deleted record versions.

> **Note**
>
> This method of deletion does not cause DELETE triggers to fire, so do not be tempted to define Before or After Delete triggers on the false assumption that you can incorporate some kind of “last rites” that will be execute just as your temporary data breathes its last!

The data and index pages of all GTT instances are placed in separate temporary files. Each connection has its own temporary file created the first time the connection references some GTT.

> **Note**
>
> These temporary files are always opened with Forced Writes = OFF, regardless of the database setting for Forced Writes.

No limit is placed on the number of GTT instances that can coexist. If you have N transactions active simultaneously and each transaction has referenced some GTT then you will have N instances of the GTT.

Views Enhancements

D. Yemanov

A couple of enhancements were made to view definitions in v.2.1.-

Use Column Aliases in CREATE VIEW

Feature requestCORE-831

(v.2.1) Column aliases can now be processed as column names in the view definition.

Example

```
CREATE VIEW V_TEST AS
  SELECT ID,
         COL1 AS CODE,
         COL2 AS NAME
  FROM TAB;
```

SQL2003 compliance for CREATE TRIGGER

A. dos Santos Fernandes

Feature request CORE-711

(v.2.1) Alternative syntax is now available for CREATE TRIGGER that complies with SQL2003.

Syntax Patterns

Existing Form

```
create trigger t1
  FOR atable
  [active] before insert or update
as
  begin
    ...
  end
```

SQL2003 Form

```
create trigger t2
  [active] before insert or update
  ON atable
as
  begin
    ...
  end
```

Note the different positions of the clause identifying the table and the different keywords pointing to the table identifier (existing: FOR; SQL2003: ON).

Both syntaxes are valid and are available also for all CREATE TRIGGER, RECREATE TRIGGER and CREATE OR ALTER TRIGGER statements.

SQL2003 Compliant Alternative for Computed Fields

D. Yemanov

Feature request CORE-1386

(v.2.1) SQL-compliant alternative syntax **GENERATED ALWAYS AS** was implemented for defining a computed field in CREATE/ALTER TABLE.

Syntax Pattern

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

It is fully equivalent semantically with the legacy form:

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

Example

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

CREATE SEQUENCE

D. Yemanov

SEQUENCE has been introduced as a synonym for GENERATOR, in accordance with SQL-99. SEQUENCE is a syntax term described in the SQL specification, whereas GENERATOR is a legacy InterBase syntax term. Use of the standard SEQUENCE syntax in your applications is recommended.

A sequence generator is a mechanism for generating successive exact numeric values, one at a time. A sequence generator is a named schema object. In dialect 3 it is a BIGINT, in dialect 1 it is an INTEGER.

Syntax patterns

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

Examples

1.

```
CREATE SEQUENCE S_EMPLOYEE;
```

2.

```
ALTER SEQUENCE S_EMPLOYEE RESTART WITH 0;
```

See also the notes about NEXT VALUE FOR.

Warning

ALTER SEQUENCE, like SET GENERATOR, is a good way to screw up the generation of key values!

REVOKE ADMIN OPTION

D. Yemanov

SYSDBA, the database creator or the owner of an object can grant rights on that object to other users. However, those rights can be made inheritable, too. By using WITH GRANT OPTION, the grantor gives the grantee the right to become a grantor of the same rights in turn. This ability can be removed by the original grantor with REVOKE GRANT OPTION FROM user.

However, there's a second form that involves roles. Instead of specifying the same rights for many users (soon it becomes a maintenance nightmare) you can create a role, assign a package of rights to that role and then grant the role to one or more users. Any change to the role's rights affect all those users.

By using WITH ADMIN OPTION, the grantor (typically the role creator) gives the grantee the right to become a grantor of the same role in turn. Until FB v2, this ability couldn't be removed unless the original grantor fiddled with system tables directly. Now, the ability to grant the role can be removed by the original grantor with REVOKE ADMIN OPTION FROM user.

SET/DROP DEFAULT Clauses for ALTER TABLE

C. Valderrama

Domains allow their defaults to be changed or dropped. It seems natural that table fields can be manipulated the same way without going directly to the system tables.

Syntax Pattern

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

Note

- Array fields cannot have a default value.
- If you change the type of a field, the default may remain in place. This is because a field can be given the type of a domain with a default but the field itself can override such domain. On the other hand, the field can be given a type directly in whose case the default belongs logically to the field (albeit the information is kept on an implicit domain created behind scenes).

Syntaxes for Changing Exceptions

D. Yemanov

The DDL statements RECREATE EXCEPTION and CREATE OR ALTER EXCEPTION (feature request SF #1167973) have been implemented, allowing either creating, recreating or altering a custom exception, depending on whether it already exists.

RECREATE EXCEPTION

RECREATE EXCEPTION is exactly like CREATE EXCEPTION if the exception does not already exist. If it does exist, its definition will be completely replaced, if there are no dependencies on it.

CREATE OR ALTER EXCEPTION

CREATE OR ALTER EXCEPTION will create the exception if it does not already exist, or will alter the definition if it does, without affecting dependencies.

ALTER EXTERNAL FUNCTION

C. Valderrama

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

COMMENT Statement

C. Valderrama

The COMMENT statement has been implemented for setting metadata descriptions.

Syntax Pattern

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

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

```
<basic_type>:
   DOMAIN
   TABLE
   VIEW
   PROCEDURE
   TRIGGER
   EXTERNAL FUNCTION
   FILTER
   EXCEPTION
   GENERATOR
   SEQUENCE
   INDEX
   ROLE
   CHARACTER SET
   COLLATION
   SECURITY CLASS[1]
```

[1]not implemented, because this type is hidden.

Extensions to CREATE VIEW Specification

D. Yemanov

FIRST/SKIP and ROWS syntaxes and PLAN and ORDER BY clauses can now be used in view specifications.

From Firebird 2.0 onward, views are treated as fully-featured SELECT expressions. Consequently, the clauses FIRST/SKIP, ROWS, UNION, ORDER BY and PLAN are now allowed in views and work as expected.

Syntax

For syntax details, refer to Select Statement & Expression Syntax in the chapter about DML.

RECREATE TRIGGER Statement Implemented

D. Yemanov

The DDL statement RECREATE TRIGGER statement is now available in DDL. Semantics are the same as for other RECREATE statements.

Usage Enhancements

The following changes will affect usage or existing, pre-Firebird 2 workarounds in existing applications or databases to some degree.

Creating Foreign Key Constraints No Longer Requires Exclusive Access

V. Horsun

Now it is possible to create foreign key constraints without needing to get an exclusive lock on the whole database.

Changed Logic for View Updates

Apply NOT NULL constraints to base tables only, ignoring the ones inherited by view columns from domain definitions.

Descriptive Identifiers for BLOB Subtypes

A. Peshkov, C. Valderrama

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

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

The alternative new syntax is:

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

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

Initially they are binary, text and others mostly for internal usage, but an adventurous user could write a new mnemonic in rdb$types and use it, since it is parsed only at declaration time. The engine keeps the numerical value. Remember, only negative subtype values are meant to be defined by users.

To get the predefined types, do

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

RDB$TYPE   RDB$TYPE_NAME                RDB$SYSTEM_FLAG
========= ============================ =================

        0 BINARY                                     1
        1 TEXT                                       1
        2 BLR                                        1
        3 ACL                                        1
        4 RANGES                                     1
        5 SUMMARY                                    1
        6 FORMAT                                     1
        7 TRANSACTION_DESCRIPTION                    1
        8 EXTERNAL_FILE_DESCRIPTION                  1
```

Examples

Original declaration:

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

Alternative declaration:

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

Declaring a name for a user defined blob subtype (remember to commit after the insertion):

```
SQL> insert into rdb$types
CON> values('RDB$FIELD_SUB_TYPE', -100, 'XDR', 'test type', 0);
SQL> commit;
SQL> declare filter pesh2 input_type xdr output_type text
CON> entry_point 'p2' module_name 'p';
SQL> show filter pesh2;
BLOB Filter: PESH2
        Input subtype: -100 Output subtype: 1
        Filter library is p
        Entry point is p2
```

Chapter 6

Data Manipulation Language (DML)

In this chapter are the additions and improvements that have been added to the SQL data manipulation language subset in the Firebird 2 development cycle. Those marked as introduced in v.2.1 are available only to ODS 11.1 and higher databases.

> **Important**
>
> A new configuration parameter, named *RelaxedAliasChecking* was added to the **firebird.conf** in Firebird 2.1 to permit a slight relaxation of the Firebird 2.0.x restrictions on mixing relation aliases and table names in a query (see DSQL Parsing of Table Names is Stricter, below).
>
> This parameter will not be a permanent fixture in Firebird but is intended as a migration aid for those needing time to adjust existing code. More information under RelaxedAliasChecking in the chapter "New Configuration Parameters".

Quick Links

- New [NOT] DISTINCT Predicate
- NULL Comparison Rule Relaxed
- NULLs Ordering Changed
- UNION Sets in Subquery Constructs
- Extended Context Variables
- Query Plans Improvements
- GROUP or ORDER by Alias Name
- GROUP BY Arbitrary Expressions
- Order * Sets by Implicit Degree Number
- NEXT VALUE FOR
- RETURNING Clause for INSERT Statements
- **Articles**
  1. Select Statement & Expression Syntax
  2. Data Type of an Aggregation Result

Common Table Expressions

Vlad Khorsun
Based on work by Paul Ruizendaal for Fyracle project

(v.2.1) A *common table expression* (CTE) is like a view that is defined locally within a main query. The engine treats a CTE like a derived table and no intermediate materialisation of the data is performed.

Benefits of CTEs

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

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

The memory and CPU overhead of a recursive CTE is much less than that of an equivalent recursive stored procedure.

Recursion Limit

Currently the recursion depth is limited to a hard-coded value of 1024.

Syntax and Rules for CTEs

```
select :
  select_expr for_update_clause lock_clause
select_expr :
  with_clause select_expr_body order_clause rows_clause
            | select_expr_body order_clause rows_clause
```

```
with_clause :
  WITH RECURSIVE with_list | WITH with_list
with_list :
  with_item | with_item ',' with_list
with_item :
  symbol_table_alias_name derived_column_list
    AS '(' select_expr ')'
select_expr_body :
  query_term
  | select_expr_body UNION distinct_noise query_term
             | select_expr_body UNION ALL query_term
```

A less formal representation:

```
WITH [RECURSIVE]
 CTE_A [(a1, a2, …)]
 AS ( SELECT … ),

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

Rules for Non-Recursive CTEs

- Multiple table expressions can be defined in one query
- Any clause legal in a SELECT specification is legal in table expressions
- Table expressions can reference one another
- References between expressions should not have loops
- Table expressions can be used within any part of the main query or another table expression
- The same table expression can be used more than once in the main query
- Table expressions (as subqueries) can be used in INSERT, UPDATE and DELETE statements
- Table expressions are legal in PSQL code
- WITH statements can not be nested

Example of a non-recursive CTE

```
WITH
  DEPT_YEAR_BUDGET AS (
    SELECT FISCAL_YEAR, DEPT_NO,
        SUM(PROJECTED_BUDGET) AS BUDGET
      FROM PROJ_DEPT_BUDGET
    GROUP BY FISCAL_YEAR, DEPT_NO
  )
SELECT D.DEPT_NO, D.DEPARTMENT,
```

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

WHERE EXISTS (
  SELECT * FROM PROJ_DEPT_BUDGET B
  WHERE D.DEPT_NO = B.DEPT_NO)
```

Rules for Recursive CTEs

- A recursive CTE is self-referencing (has a reference to itself)
- A recursive CTE is a UNION of recursive and non-recursive members:
  - At least one non-recursive member (anchor) must be present
  - Non-recursive members are placed first in the UNION
  - Recursive members are separated from anchor members and from one another with UNION ALL clauses, i.e.,

    non-recursive member (anchor)
    UNION [ALL | DISTINCT]
    non-recursive member (anchor)
    UNION [ALL | DISTINCT]
    non-recursive member (anchor)
    UNION ALL
    recursive member
    UNION ALL
    recursive member

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

Example of a recursive CTE

```
WITH RECURSIVE
  DEPT_YEAR_BUDGET AS
  (
    SELECT FISCAL_YEAR, DEPT_NO,
        SUM(PROJECTED_BUDGET) AS BUDGET
      FROM PROJ_DEPT_BUDGET
    GROUP BY FISCAL_YEAR, DEPT_NO
  ),

  DEPT_TREE AS
  (
    SELECT DEPT_NO, HEAD_DEPT, DEPARTMENT,
        CAST('' AS VARCHAR(255)) AS INDENT
      FROM DEPARTMENT
     WHERE HEAD_DEPT IS NULL

    UNION ALL

    SELECT D.DEPT_NO, D.HEAD_DEPT, D.DEPARTMENT,
    H.INDENT || '  '
      FROM DEPARTMENT D
      JOIN DEPT_TREE H
        ON D.HEAD_DEPT = H.DEPT_NO
  )

  SELECT D.DEPT_NO,
 D.INDENT || D.DEPARTMENT AS DEPARTMENT,
 B_1993.BUDGET AS B_1993,
 B_1994.BUDGET AS B_1994,
 B_1995.BUDGET AS B_1995,
 B_1996.BUDGET AS B_1996

  FROM DEPT_TREE D
    LEFT JOIN DEPT_YEAR_BUDGET B_1993
      ON D.DEPT_NO = B_1993.DEPT_NO
      AND B_1993.FISCAL_YEAR = 1993

    LEFT JOIN DEPT_YEAR_BUDGET B_1994
      ON D.DEPT_NO = B_1994.DEPT_NO
      AND B_1994.FISCAL_YEAR = 1994

    LEFT JOIN DEPT_YEAR_BUDGET B_1995
      ON D.DEPT_NO = B_1995.DEPT_NO
      AND B_1995.FISCAL_YEAR = 1995

    LEFT JOIN DEPT_YEAR_BUDGET B_1996
      ON D.DEPT_NO = B_1996.DEPT_NO
      AND B_1996.FISCAL_YEAR = 1996
```

The LIST Function

Oleg Loa
Dmitry Yemanov

(v.2.1) This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values.

Format

```
<list function> ::=
  LIST '(' [ {ALL | DISTINCT} ] <value expression> [',' <delimiter value>
   ] ')'

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

Syntax Rules

1. If neither ALL nor DISTINCT is specified, ALL is implied.

2. If <delimiter value> is omitted, a comma is used to separate the concatenated values.

Other Notes

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

2. The result value is of type BLOB with SUB_TYPE TEXT for all cases except list of BLOB with different subtype.

3. Ordering of values within a group is implementation-defined.

Examples

```
/* A */
  SELECT LIST(ID, ':')
  FROM MY_TABLE

/* B */
  SELECT TAG_TYPE, LIST(TAG_VALUE)
  FROM TAGS
  GROUP BY TAG_TYPE
```

The RETURNING Clause

Dmitry Yemanov
Adriano dos Santos Fernandes

(v.2.1) The purpose of this SQL enhancement is to enable the column values stored into a table as a result of the `INSERT`, `UPDATE OR INSERT`, `UPDATE` and `DELETE` statements to be returned to the client.

The most likely usage is for retrieving the value generated for a primary key inside a BEFORE-trigger. The RETURNING clause is optional and is available in both DSQL and PSQL, although the rules differ slightly.

In DSQL, the execution of the operation itself and the return of the set occur in a single protocol round trip.

Because the RETURNING clause is designed to return a singleton set in response to completing an operation on a single record, it is not valid to specify the clause in a statement that inserts, updates or deletes multiple records.

> **Note**
>
> In DSQL, the statement always returns the set, even if the operation has no effect on any record. Hence, at this stage of implementation, the potential exists to return an “empty” set. (This may be changed in future.)

Syntax Patterns

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

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

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

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

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

Rules for Using a RETURNING Clause

1. The INTO part (i.e. the variable list) is allowed in PSQL only, for assigning the output set to local variables. It is rejected in DSQL.

2. The presence of the RETURNING clause causes an INSERT statement to be described by the API as `isc_info_sql_stmt_exec_procedure` rather than `isc_info_sql_stmt_insert`. Existing connectivity drivers should already be capable of supporting this feature without special alterations.

3. The RETURNING clause ignores any explicit record change (update or delete) that occurs as a result of the execution of an AFTER trigger.

4. OLD and NEW context variables can be used in the RETURNING clause of UPDATE and INSERT OR UPDATE statements.

5. In UPDATE and INSERT OR UPDATE statements, field references that are unqualified or qualified by table name or relation alias are resolved to the value of the corresponding NEW context variable.

Examples

1.

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

2.

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

3.

```
DELETE FROM T1
  WHERE F1 = 1
    RETURNING F2;
```

4.

```
UPDATE T1
    SET F2 = F2 * 10
      RETURNING OLD.F2, NEW.F2;
```

UPDATE OR INSERT Statement

Adriano dos Santos Fernandes

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

Syntax Pattern

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

Examples

1.

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

2.

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

3.

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

4.

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

Usage Notes

1. When MATCHING is omitted, the existence of a primary key is required.
2. INSERT and UPDATE permissions are needed on <table or view>.

3. If the RETURNING clause is present, then the statement is described as `isc_info_sql_stmt_exec_procedure` by the API; otherwise, it is described as `isc_info_sql_stmt_insert`.

> **Note**
>
> A “multiple rows in singleton select” error will be raised if the RETURNING clause is present and more than one record matches the search condition.

MERGE Statement

Adriano dos Santos Fernandes

(v.2.1) This syntax has been introduced to enable a record to be either updated or inserted, according to whether or not a stated condition is met. The statement is available in both DSQL and PSQL.

Syntax Pattern

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

<merge when matched> ::=
  WHEN MATCHED THEN
    UPDATE SET <assignment list>

<merge when not matched> ::=
  WHEN NOT MATCHED THEN
    INSERT [ <left paren> <column list> <right paren> ]
    VALUES <left paren> <value list> <right paren>
```

Rules for MERGE

1. At least one of <merge when matched> and <merge when not matched> should be specified

2. Neither should be specified more than once.

> **Note**
>
> A right join is made between the INTO and USING tables using the condition. UPDATE is called when a matching record exists in the left (INTO) table, otherwise INSERT is called.
>
> If no record is returned from the join, INSERT is not called.

Example

```
MERGE INTO customers c
  USING (SELECT * FROM customers_delta WHERE id > 10) cd
  ON (c.id = cd.id)
```

```
    WHEN MATCHED THEN
      UPDATE SET
        name = cd.name
    WHEN NOT MATCHED THEN
      INSERT (id, name)
      VALUES (cd.id, cd.name)
```

New JOIN Types

Adriano dos Santos Fernandes

(v.2.1) Two new JOIN types are introduced: the NAMED COLUMNS join and its close relative, the NATURAL join.

Syntax and Rules

```
<named columns join> ::=
  <table reference> <join type> JOIN <table reference>
    USING ( <column list> )

<natural join> ::=
 <table reference> NATURAL <join type> JOIN <table primary>
```

Named columns join

1. All columns specified in <column list> should exist in the tables at both sides.
2. An equi-join (<left table>.<column> = <right table>.<column>) is automatically created for all columns (ANDed).
3. The USING columns can be accessed without qualifiers—in this case, the result is equivalent to COALESCE(<left table>.<column>, <right table>.<column>).
4. In “SELECT *”, USING columns are expanded once, using the above rule.

Natural join

1. A “named columns join” is automatically created with all columns common to the left and right tables.
2. If there is no common column, a CROSS JOIN is created.

Examples

```
/* 1 */
select * from employee
  join department
  using (dept_no);

/* 2 */
select * from employee_project
```

```
  natural join employee
  natural join project;
```

CROSS JOIN

D. Yemanov

(V.2.0.x) CROSS JOIN is now supported. Logically, this syntax pattern:

```
   A CROSS JOIN B
```

is equivalent to either of the following:

```
   A INNER JOIN B ON 1 = 1
```

or, simply:

```
  FROM A, B
```

INSERT with Defaults

D. Yemanov

Feature request

(v.2.1) It is now possible to INSERT without supplying values, if Before Insert triggers and/or declared defaults are available for every column and none is dependent on the presence of any supplied 'NEW' value.

Example

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

BLOB Subtype 1 Compatibility with VarChar

A. dos Santos Fernandes

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

Full Equality Comparisons Between BLOBs

(v.2.0.x) Comparison can be performed on the entire content of a text BLOB.

RDB$DB_KEY Returns NULL in Outer Joins

A. dos Santos Fernandes

Feature request CORE-979

(v.2.1) By some anomaly, the physical RDB$DB_KEY has always returned a value on every output row when specified in an outer join, thereby making a test predicated on the assumption that a non-match returns NULL in all fields return False when it ought to return True. Now, RDB$DB_KEY returns NULL when it should do so.

Sorting on BLOB and ARRAY Columns is Restored

Dmitry Yemanov

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

That change was reversed in the RC2 pre-release version, not because it was wrong but because so many users complained that it broke the behaviour of legacy applications.

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

Built-in Functions

(v.2.1) Some existing built-in functions have been enhanced, while a large number of new ones has been added.

New Built-in Functions

Adriano dos Santos Fernandes
Oleg Loa
Alexey Karyakin

A number of built-in functions has been implemented in V.2.1 to replace common UDFs with the same names. The built-in functions will not be used if the UDF of the same name is declared in the database.

> **Note**
>
> The choice between UDF and built-in function is decided when compiling the statement. If the statement is compiled in a PSQL module whilst the UDF is available in the database, then the module will continue to require the UDF declaration to be present until it is next recompiled.
>
> The new built-in function *DECODE()* does not have an equivalent UDF in the libraries that are distributed with Firebird.

The functions are detailed in Appendix A.

> **Note**
>
> Several of these built-in functions were already available in Firebird 2/ODS 11, viz., LOWER(), TRIM(), BIT_LENGTH(), CHAR_LENGTH() and OCTET_LENGTH().

Enhancements to Functions

A. dos Santos Fernandes

EXTRACT(WEEK FROM DATE)
Feature request CORE-663

The EXTRACT() function is extended to support the ISO-8601 ordinal week numbers. For example:

```
EXTRACT (WEEK FROM date '30.09.2007')
```

returns 39

Specify the Scale for TRUNC()
Feature request CORE-1340

In Beta 1 the implementation of the TRUNC() function supported only one argument, the value to be truncated. From Beta 2, an optional second argument can be supplied to specify the scale of the truncation. For example:

```
select
  trunc(987.65, 1),
  trunc(987.65, -1)
  from rdb$database;
```

returns 987.60, 980.00

For other examples of using TRUNC() with and without the optional scale argument, refer to the alphabetical listing of functions in Appendix A.

Milliseconds Handling for EXTRACT(), DATEADD() and DATEDIFF()
Feature request CORE-1387

From v.2.1 Beta 2, EXTRACT(), DATEADD() and DATEDIFF() can operate with milliseconds (represented as an integer number). For example:

```
EXTRACT ( MILLISECOND FROM timestamp '01.01.2000 01:00:00.1234' )
```

returns 123

```
DATEADD ( MILLISECOND, 100, timestamp '01.01.2000 01:00:00.0000' )
DATEDIFF ( MILLISECOND, timestamp '01.01.2000 02:00:00.0000', timestamp '01.01.2000 01:00:00.
```

For more explanatory examples of using DATEADD() and DATEDIFF(), refer to the alphabetical listing of functions in Appendix A.

Functions Enhanced in V.2.0.x

Some function enhancements were already available in the V.2.0.x releases:

IIF() Expression

O. Loa

(V.2.0.x) An IIF() expression can be used as a shortcut for a CASE expression that tests exactly two conditions. It returns the value of the first sub-expression if the given search condition evaluates to TRUE, otherwise it returns a value of the second sub-expression.

```
IIF (<search_condition>, <value1>, <value2>)
```

is implemented as a shortcut for

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

Example

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

Improvement in CAST() Behaviour

D. Yemanov

(V.2.0.x) The infamous “Datatype unknown” error (SF Bug #1371274) when attempting some castings has been eliminated. It is now possible to use CAST to advise the engine about the data type of a parameter.

Example

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

Expression Arguments for SUBSTRING()

O. Loa, D. Yemanov

(V.2.0.x) The built-in function SUBSTRING() can now take arbitrary expressions in its parameters.

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

Note

The length of the resulting column is the same as the length of the first argument. This means that, in the following

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

the new column has a length of 50, not 1. (Thank the SQL standards committee!)

DSQL Parsing of Table Names is Stricter

A. Brinkman

Alias handling and ambiguous field detecting have been improved. In summary:

1. When a table alias is provided for a table, either that alias, or no alias, must be used. It is no longer valid to supply only the table name.
2. Ambiguity checking now checks first for ambiguity at the current level of scope, making it valid in some conditions for columns to be used without qualifiers at a higher scope level.

Examples

1. When an alias is present it must be used; or no alias at all is allowed.
   a. This query was allowed in FB1.5 and earlier versions:

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

   but will now correctly report an error that the field "RDB$RELATIONS.RDB$RELATION_NAME" could not be found.

   Use this (preferred):

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

   or this statement:

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

b. The statement below will now correctly use the FieldID from the subquery and from the updating table:

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

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

c. This example did not run correctly in Firebird 1.5 and earlier:

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

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

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

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

b. Ambiguity checking in subqueries: the query below would run in Firebird 1.5 without reporting an ambiguity, but will report it in Firebird 2:

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

EXECUTE BLOCK Statement

V. Khorsun

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

Syntax pattern

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

For the client, the call `isc_dsql_sql_info` with the parameter `isc_info_sql_stmt_type` returns

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

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

Example

The user SQL is

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

The preprocessed SQL is

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

```
    SUSPEND;
  END
```

Derived Tables

A. Brinkman

Implemented support for derived tables in DSQL (subqueries in FROM clause) as defined by SQL200X. A derived table is a set, derived from a dynamic SELECT statement. Derived tables can be nested, if required, to build complex queries and they can be involved in joins as though they were normal tables or views.

Syntax Pattern

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

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

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

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

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

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

Examples

a) Simple derived table:

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

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

```
SELECT
  DT.FIELDS,
  Count(*)
FROM
  (SELECT
     R.RDB$RELATION_NAME,
```

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

c) UNION and ORDER BY example:

```
SELECT
  DT.*
FROM
  (SELECT
     R.RDB$RELATION_NAME,
     R.RDB$RELATION_ID
   FROM
     RDB$RELATIONS R
   UNION ALL
   SELECT
     R.RDB$OWNER_NAME,
     R.RDB$RELATION_ID
   FROM
     RDB$RELATIONS R
   ORDER BY
     2) AS DT
WHERE
  DT.RDB$RELATION_ID <= 4
```

Points to Note

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

ROLLBACK RETAIN Syntax

D. Yemanov

The ROLLBACK RETAIN statement is now supported in DSQL.

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

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

Syntax pattern: follows that of COMMIT RETAIN.

ROWS Syntax

D. Yemanov

ROWS syntax is used to limit the number of rows retrieved from a select expression. For an uppermost-level select statement, it would specify the number of rows to be returned to the host program. A more understandable alternative to the FIRST/SKIP clauses, the ROWS syntax accords with the latest SQL standard and brings some extra benefits. It can be used in unions, any kind of subquery and in UPDATE or DELETE statements.

It is available in both DSQL and PSQL.

Syntax Pattern

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

Examples

1.

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

2.

```
SELECT COL1, COL2,
  ( SELECT COL3 FROM T3 ORDER BY COL4 DESC ROWS 1 )
FROM T4
```

3.

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

Points to Note

1. When <expr2> is omitted, then ROWS <expr1> is semantically equivalent to FIRST <expr1>. When both <expr1> and <expr2> are used, then ROWS <expr1> TO <expr2> means the same as FIRST (<expr2> - <expr1> + 1) SKIP (<expr1> - 1)

2. There is nothing that is semantically equivalent to a SKIP clause used without a FIRST clause.

Enhancements to UNION Handling

The rules for UNION queries have been improved as follows:

UNION DISTINCT Keyword Implementation

D. Yemanov

UNION DISTINCT is now allowed as a synonym for simple UNION, in accordance with the SQL-99 specification. It is a minor change: DISTINCT is the default mode, according to the standard. Formerly, Firebird did not support the explicit inclusion of the optional keyword DISTINCT.

Syntax Pattern

```
UNION [{DISTINCT | ALL}]
```

Improved Type Coercion in UNIONs

A. Brinkman

Automatic type coercion logic between subsets of a union is now more intelligent. Resolution of the data type of the result of an aggregation over values of compatible data types, such as case expressions and columns at the same position in a union query expression, now uses smarter rules.

Syntax Rules

Let DTS be the set of data types over which we must determine the final result data type.

1. All of the data types in DTS shall be comparable.
2. Case:
   a. If any of the data types in DTS is character string, then:
      i. If any of the data types in DTS is variable-length character string, then the result data type is variable-length character string with maximum length in characters equal to the largest maximum amongst the data types in DTS.
      ii. Otherwise, the result data type is fixed-length character string with length in characters equal to the maximum of the lengths in characters of the data types in DTS.
      iii. The characterset/collation is used from the first character string data type in DTS.
   b. If all of the data types in DTS are exact numeric, then the result data type is exact numeric with scale equal to the maximum of the scales of the data types in DTS and the maximum precision of all data types in DTS.

      > **Note**
      >
      > NOTE :: Checking for precision overflows is done at run-time only. The developer should take measures to avoid the aggregation resolving to a precision overflow.

   c. If any data type in DTS is approximate numeric, then each data type in DTS shall be numeric else an error is thrown.
   d. If some data type in DTS is a date/time data type, then every data type in DTS shall be a date/time data type having the same date/time type.

e. If any data type in DTS is BLOB, then each data type in DTS shall be BLOB and all with the same sub-type.

UNIONs Allowed in ANY/ALL/IN Subqueries

D. Yemanov

The subquery element of an ANY, ALL or IN search may now be a UNION query.

Enhancements to NULL Logic

The following features involving NULL in DSQL have been implemented:

New [NOT] DISTINCT Test Treats Two NULL Operands as Equal

O. Loa, D. Yemanov

A new equivalence predicate behaves exactly like the equality/inequality predicates, but, instead of testing for equality, it tests whether one operand is distinct from the other.

Thus, IS NOT DISTINCT treats (NULL equals NULL) as if it were true, since one NULL (or expression resolving to NULL) is not distinct from another. It is available in both DSQL and PSQL.

Syntax Pattern

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

Examples

1.

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

2.

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

> **Note**
>
> **Points to note**
>
> 1. Because the DISTINCT predicate considers that two NULL values are not distinct, it never evaluates to the truth value UNKNOWN. Like the IS [NOT] NULL predicate, it can only be True or False.
> 2. The NOT DISTINCT predicate can be optimized using an index, if one is available.

NULL Comparison Rule Relaxed

D. Yemanov

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

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

> **Note**
>
> All such expressions evaluate to NULL. The change does not alter nullability-aware semantics of the engine, it simply relaxes the syntax restrictions a little.

NULLs Ordering Changed to Comply with Standard

N. Samofatov

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

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

Examples

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

SQL> select a from gnull order by a asc;

           A
============
      <null>
           1

SQL> select a from gnull order by a desc;

           A
```

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

SQL> select a from gnull order by a asc nulls first;

           A
============
      <null>
           1

SQL> select a from gnull order by a asc nulls last;

           A
============
           1
      <null>


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

           A
============
           1
      <null>

SQL> select a from gnull order by a desc nulls first;

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

Subqueries and INSERT Statements Can Now Accept UNION Sets

D. Yemanov

SELECT specifications used in subqueries and in INSERT INTO <insert-specification> SELECT.. statements can now specify a UNION set.

New Extensions to UPDATE and DELETE Syntaxes

O. Loa

ROWS specifications and PLAN and ORDER BY clauses can now be used in UPDATE and DELETE statements.

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

Syntax Pattern

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

```
[ROWS <value> [TO <value>]]
```

or

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

Extended Context Variables

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

Sub-second Values Enabled for Time and DateTime Variables

D. Yemanov

CURRENT_TIMESTAMP, 'NOW' Now Return Milliseconds

The context variable CURRENT_TIMESTAMP and the date/time literal 'NOW' will now return the sub-second time part in milliseconds.

Seconds Precision Enabled for CURRENT_TIME and CURRENT_TIMESTAMP

CURRENT_TIME and CURRENT_TIMESTAMP now optionally allow seconds precision

The feature is available in both DSQL and PSQL.

Syntax Pattern

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

Examples

```
1. SELECT CURRENT_TIME FROM RDB$DATABASE;
2. SELECT CURRENT_TIME(3) FROM RDB$DATABASE;
3. SELECT CURRENT_TIMESTAMP(3) FROM RDB$DATABASE;
```

Note

1. The maximum possible precision is 3 which means accuracy of 1/1000 second (one millisecond). This accuracy may be improved in the future versions.
2. If no seconds precision is specified, the following values are implicit:
   - 0 for CURRENT_TIME
   - 3 for CURRENT_TIMESTAMP

A Useful Trick with Date Literals

H. Borrie

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

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

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

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

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

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

```
select TIMESTAMP 'NOW'
```

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

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

New System Functions to Retrieve Context Variables

N. Samofatov

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

Syntax Pattern

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

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

Declaration

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

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

Usage

RDB$SET_CONTEXT and RDB$GET_CONTEXT set and retrieve the current value of a context variable. Groups of context variables with similar properties are identified by Namespace identifiers. The namespace determines the usage rules, such as whether the variables may be read and written to, and by whom.

> **Note**
>
> Namespace and variable names are case-sensitive.

- RDB$GET_CONTEXT retrieves current value of a variable. If the variable does not exist in namespace, the function returns NULL.
- RDB$SET_CONTEXT sets a value for specific variable, if it is writable. The function returns a value of 1 if the variable existed before the call and 0 otherwise.
- To delete a variable from a context, set its value to NULL.

Pre-defined Namespaces

A fixed number of pre-defined namespaces is available:

USER_SESSION

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

USER_TRANSACTION

Offers similar possibilities for individual transactions.

SYSTEM

Provides read-only access to the following variables:

- NETWORK_PROTOCOL :: The network protocol used by client to connect. Currently used values: “TCPv4”, “WNET”, “XNET” and NULL.
- CLIENT_ADDRESS :: The wire protocol address of the remote client, represented as a string. The value is an IP address in form "xxx.xxx.xxx.xxx" for TCPv4 protocol; the local process ID for XNET protocol; and NULL for any other protocol.
- DB_NAME :: Canonical name of the current database. It is either the alias name (if connection via file names is disallowed DatabaseAccess = NONE) or, otherwise, the fully expanded database file name.
- ISOLATION_LEVEL :: The isolation level of the current transaction. The returned value will be one of "READ COMMITTED", "SNAPSHOT", "CONSISTENCY".
- TRANSACTION_ID :: The numeric ID of the current transaction. The returned value is the same as would be returned by the CURRENT_TRANSACTION pseudo-variable.
- SESSION_ID :: The numeric ID of the current session. The returned value is the same as would be returned by the CURRENT_CONNECTION pseudo-variable.
- CURRENT_USER :: The current user. The returned value is the same as would be returned by the CURRENT_USER pseudo-variable or the predefined variable USER.
- CURRENT_ROLE :: Current role for the connection. Returns the same value as the CURRENT_ROLE pseudo-variable.

Notes

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

Example of Use

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

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

CREATE TRIGGER UI_JOURNAL FOR JOURNAL BEFORE INSERT OR UPDATE
  as
  begin
    new.jrn_lastuser = rdb$get_context('USER_TRANSACTION', 'User_ID');
    new.jrn_lastaddr = rdb$get_context('SYSTEM', 'CLIENT_ADDRESS');
    new.jrn_lasttransaction = rdb$get_context('USER_TRANSACTION', 'Trn_ID');
  end ^
  commit ^
  execute procedure set_context('skidder', 1) ^
```

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

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

Example

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

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

0 means not defined already; we have set it to 'ru'

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

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

1 means it was defined already; we have changed it to 'ca'

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

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

1 says it existed before; we have changed it to NULL, i.e. undefined it.

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

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

0, since nothing actually happened this time: it was already undefined .

Improvements in Handling User-specified Query Plans

D. Yemanov

1. Plan fragments are propagated to nested levels of joins, enabling manual optimization of complex outer joins

2. A user-supplied plan will be checked for correctness in outer joins

3. Short-circuit optimization for user-supplied plans has been added

4. A user-specified access path can be supplied for any SELECT-based statement or clause

Syntax rules

The following schema describing the syntax rules should be helpful when composing plans.

```
PLAN ( { <stream_retrieval> | <sorted_streams> | <joined_streams> } )

<stream_retrieval> ::= { <natural_scan> | <indexed_retrieval> |
    <navigational_scan> }

<natural_scan> ::= <stream_alias> NATURAL

<indexed_retrieval> ::= <stream_alias> INDEX ( <index_name>
    [, <index_name> ...] )

<navigational_scan> ::= <stream_alias> ORDER <index_name>
    [ INDEX ( <index_name> [, <index_name> ...] ) ]

<sorted_streams> ::= SORT ( <stream_retrieval> )

<joined_streams> ::= JOIN ( <stream_retrieval>, <stream_retrieval>
    [, <stream_retrieval> ...] )
    | [SORT] MERGE ( <sorted_streams>, <sorted_streams> )
```

Details

Natural scan means that all rows are fetched in their natural storage order. Thus, all pages must be read before search criteria are validated.

Indexed retrieval uses an index range scan to find row ids that match the given search criteria. The found matches are combined in a sparse bitmap which is sorted by page numbers, so every data page will be read only once. After that the table pages are read and required rows are fetched from them.

Navigational scan uses an index to return rows in the given order, if such an operation is appropriate.-

- The index b-tree is walked from the leftmost node to the rightmost one.
- If any search criterion is used on a column specified in an ORDER BY clause, the navigation is limited to some subtree path, depending on a predicate.
- If any search criterion is used on other columns which are indexed, then a range index scan is performed in advance and every fetched key has its row id validated against the resulting bitmap. Then a data page is read and the required row is fetched.

> **Note**
>
> Note that a navigational scan incurs random page I/O, as reads are not optimized.

A *sort operation* performs an external sort of the given stream retrieval.

A *join* can be performed either via the nested loops algorithm (JOIN plan) or via the sort merge algorithm (MERGE plan).-

- An *inner nested loop join* may contain as many streams as are required to be joined. All of them are equivalent.
- An *outer nested loops join* always operates with two streams, so you'll see nested JOIN clauses in the case of 3 or more outer streams joined.

A *sort merge* operates with two input streams which are sorted beforehand, then merged in a single run.

Examples

```
SELECT RDB$RELATION_NAME
FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME LIKE 'RDB$%'
PLAN (RDB$RELATIONS NATURAL)
ORDER BY RDB$RELATION_NAME

SELECT R.RDB$RELATION_NAME, RF.RDB$FIELD_NAME
FROM RDB$RELATIONS R
  JOIN RDB$RELATION_FIELDS RF
  ON R.RDB$RELATION_NAME = RF.RDB$RELATION_NAME
PLAN MERGE (SORT (R NATURAL), SORT (RF NATURAL))
```

Notes

1. A PLAN clause may be used in all select expressions, including subqueries, derived tables and view definitions. It can be also used in UPDATE and DELETE statements, because they're implicitly based on select expressions.
2. If a PLAN clause contains some invalid retrieval description, then either an error will be returned or this bad clause will be silently ignored, depending on severity of the issue.
3. ORDER <navigational_index> INDEX (<filter_indices>) kind of plan is reported by the engine and can be used in the user-supplied plans starting with FB 2.0.

Improvements in Sorting

A. Brinkman

Some useful improvements have been made to SQL sorting operations:

Order By or Group By <alias-name>

Column aliases are now allowed in both these clauses.

Examples:

1. ORDER BY

```
SELECT RDB$RELATION_ID AS ID
FROM RDB$RELATIONS
ORDER BY ID
```

2. GROUP BY

```
SELECT RDB$RELATION_NAME AS ID, COUNT(*)
FROM RDB$RELATION_FIELDS
GROUP BY ID
```

GROUP BY Arbitrary Expressions

A GROUP BY condition can now be any valid expression.

Example

```
...
  GROUP BY
  SUBSTRING(CAST((A * B) / 2 AS VARCHAR(15)) FROM 1 FOR 2)
```

Order * Sets by Implicit Degree Number

Order by degree (ordinal column position) now works on a select * list.

Example

```
SELECT *
  FROM RDB$RELATIONS
  ORDER BY 9
```

Parameters and Ordinal Sorts--a “Gotcha”

According to grammar rules, since v.1.5, ORDER BY <value_expression> is allowed and <value_expression> could be a variable or a parameter. It is tempting to assume that ORDER BY <degree_number> could thus be validly represented as a replaceable input parameter, or an expression containing a parameter.

However, while the DSQL parser does not reject the parameterised ORDER BY clause expression if it resolves to an integer, the optimizer requires an absolute, constant value in order to identify the *position in the output list* of the ordering column or derived field. If a parameter is accepted by the parser, the output will undergo a “dummy sort” and the returned set will be unsorted.

NEXT VALUE FOR Expression

D. Yemanov

Added SQL-99 compliant NEXT VALUE FOR <sequence_name> expression as a synonym for GEN_ID(<generator-name>, 1), complementing the introduction of CREATE SEQUENCE syntax as the SQL standard equivalent of CREATE GENERATOR.

Examples

1.

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

2.

```
INSERT INTO EMPLOYEE (ID, NAME)
  VALUES (NEXT VALUE FOR S_EMPLOYEE, 'John Smith');
```

Note

1. Currently, increment ("step") values not equal to 1 (one) can be used only by calling the GEN_ID function. Future versions are expected to provide full support for SQL-99 sequence generators, which allows the required increment values to be specified at the DDL level. Unless there is a vital need to use a step value that is not 1, use of a NEXT VALUE FOR value expression instead of the GEN_ID function is recommended.
2. GEN_ID(<name>, 0) allows you to retrieve the current sequence value, but it should never be used in insert/update statements, as it produces a high risk of uniqueness violations in a concurrent environment.

RETURNING Clause for INSERT Statements

D. Yemanov

The RETURNING clause syntax has been implemented for the INSERT statement, enabling the return of a result set from the INSERT statement. The set contains the column values actually stored. Most common usage would be for retrieving the value of the primary key generated inside a BEFORE-trigger.

Available in DSQL and PSQL.

Syntax Pattern

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

Example(s)

1.

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

2.

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

Note

1. The INTO part (i.e. the variable list) is allowed in PSQL only (to assign local variables) and rejected in DSQL.
2. In DSQL, values are being returned within the same protocol roundtrip as the INSERT itself is executed.
3. If the RETURNING clause is present, then the statement is described as isc_info_sql_stmt_exec_procedure by the API (instead of isc_info_sql_stmt_insert), so the existing connectivity drivers should support this feature automagically.
4. Any explicit record change (update or delete) performed by AFTER-triggers is ignored by the RETURNING clause.
5. Cursor based inserts (INSERT INTO ... SELECT ... RETURNING ...) are not supported.
6. This clause can return table column values or arbitrary expressions.

Articles

SELECT Statement & Expression Syntax

Dmitry Yemanov

About the semantics

- A select statement is used to return data to the caller (PSQL module or the client program)
- Select expressions retrieve parts of data that construct columns that can be in either the final result set or in any of the intermediate sets. Select expressions are also known as subqueries.

Syntax rules

```
<select statement> ::=
  <select expression> [FOR UPDATE] [WITH LOCK]

<select expression> ::=
  <query specification> [UNION [{ALL | DISTINCT}] <query specification>]

<query specification> ::=
  SELECT [FIRST <value>] [SKIP <value>] <select list>
  FROM <table expression list>
  WHERE <search condition>
  GROUP BY <group value list>
  HAVING <group condition>
  PLAN <plan item list>
  ORDER BY <sort value list>
  ROWS <value> [TO <value>]

<table expression> ::=
  <table name> | <joined table> | <derived table>

<joined table> ::=
  {<cross join> | <qualified join>}
```

```
<cross join> ::=
  <table expression> CROSS JOIN <table expression>

<qualified join> ::=
  <table expression> [{INNER | {LEFT | RIGHT | FULL} [OUTER]}] JOIN <table expression>
  ON <join condition>

<derived table> ::=
  '(' <select expression> ')'
```

Conclusions

- FOR UPDATE mode and row locking can only be performed for a final dataset, they cannot be applied to a subquery
- Unions are allowed inside any subquery
- Clauses FIRST, SKIP, PLAN, ORDER BY, ROWS are allowed for any subquery

Notes

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

  Clauses allowed at the end of UPDATE/DELETE statements are PLAN, ORDER BY and ROWS.

Data Type of an Aggregation Result

Arno Brinkman

When aggregations, CASE evaluations and UNIONs for output columns are performed over a mix of comparable data types, the engine has to choose one data type for the result. The developer often has to prepare a variable or buffer for such results and is mystified when a request returns a data type exception. The rules followed by the engine in determining the data type for an output column under these conditions are explained here.

1. Let DTS be the set of data types over which we must determine the final result data type.
2. All of the data types in DTS shall be comparable.
3. In the case that
   a. any of the data types in DTS is character string
      i. If all data types in DTS are fixed-length character strings, then the result is also a fixed-length character string; otherwise the result is a variable-length character string.

         The resulting string length, in characters, is equal to the maximum of the lengths, in characters, of the data types in DTS.

ii. The character set and collation used are taken from the data type of the first character string in DTS.

b. all of the data types in DTS are exact numeric

the result data type is exact numeric with scale equal to the maximum of the scales of the data types in DTS and precision equal to the maximum precision of all data types in DTS.

c. any data type in DTS is approximate numeric

each data type in DTS must be numeric, otherwise an error is thrown.

d. any data type in DTS is a date/time data type

every data type in DTS must be a date/time type having the *same date/time type*, otherwise an error is thrown.

e. any data type in DTS is BLOB

each data type in DTS must be BLOB and all with the same sub-type.

Chapter 7

Procedural SQL (PSQL)

A handful of improvements was added to the collection of PSQL extensions that came with Firebird 2. The highlights are new capabilities to use domains and collation sequences when declaring variables and arguments in procedures and triggers.

Quick Links

Domains in PSQL

Adriano dos Santos Fernandes

(V.2.1) It is now possible to use a domain when declaring the data types of arguments and variables in PSQL modules. Depending on your requirements, you can declare the argument or variable using

- the domain identifier alone, in lieu of the native data type identifier, to have the variable inherit all of the attributes of the domain; or
- the data type of the domain, without inheriting CHECK constraints and the DEFAULT value (if declared in the domain), by including the `TYPE OF` keyword in the declaration (see the syntax below)

Syntax

```
data_type ::=
    <builtin_data_type>
    | <domain_name>
    | TYPE OF <domain_name>
```

Examples

```
CREATE DOMAIN DOM AS INTEGER;

CREATE PROCEDURE SP (
  I1 TYPE OF DOM,
  I2 DOM)
```

```
RETURNS (
  O1 TYPE OF DOM,
  O2 DOM)
AS
  DECLARE VARIABLE V1 TYPE OF DOM;
  DECLARE VARIABLE V2 DOM;

BEGIN
  ...
END
```

> **Note**
>
> A new field RDB$VALID_BLR was added in RDB$PROCEDURES and RDB$TRIGGERS to indicate whether the procedure/trigger is valid after an ALTER DOMAIN operation. The value of RDB$VALID_BLR is shown in the ISQL commands SHOW PROCEDURE or SHOW TRIGGER.

COLLATE in Stored Procedures and Parameters

A. dos Santos Fernandes

(V.2.1) Collations can now be applied to PSQL variables, including stored procedure parameters.

WHERE CURRENT OF Now Allowed for Views

Feature request CORE-1213

(V.2.1) The cursor operator **WHERE CURRENT OF** can now step through a cursor set selected from a view set, just as it does in a cursor set output from a SELECT on a table. For example:

```
...
FOR SELECT ...
   FROM MY_VIEW INTO ... AS CURSOR VIEW_CURSOR DO
BEGIN
  ...
  DELETE FROM MY_VIEW
    WHERE CURRENT OF VIEW_CURSOR;
  ...
END
```

Context Variable ROW_COUNT Enhanced

D. Yemanov

ROW_COUNT has been enhanced so that it can now return the number of rows returned by a SELECT statement.

For example, it can be used to check whether a singleton SELECT INTO statement has performed an assignment:

```
  ..
```

```
BEGIN
  SELECT COL FROM TAB INTO :VAR;

  IF (ROW_COUNT = 0) THEN
    EXCEPTION NO_DATA_FOUND;
END
..
```

See also its usage in the examples below for explicit PSQL cursors.

Explicit Cursors

D. Yemanov

It is now possible to declare and use multiple cursors in PSQL. Explicit cursors are available in a DSQL EXECUTE BLOCK structure as well as in stored procedures and triggers.

Syntax pattern

```
DECLARE [VARIABLE] <cursor_name> CURSOR FOR ( <select_statement> );
OPEN <cursor_name>;
FETCH <cursor_name> INTO <var_name> [, <var_name> ...];
CLOSE <cursor_name>;
```

Examples

1.

```
DECLARE RNAME CHAR(31);
DECLARE C CURSOR FOR ( SELECT RDB$RELATION_NAME
                       FROM RDB$RELATIONS );
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
  BEGIN
    FETCH C INTO :RNAME;
    IF (ROW_COUNT = 0) THEN
      LEAVE;
    SUSPEND;
  END
  CLOSE C;
END
```

2.

```
DECLARE RNAME CHAR(31);
DECLARE FNAME CHAR(31);
DECLARE C CURSOR FOR ( SELECT RDB$FIELD_NAME
                       FROM RDB$RELATION_FIELDS
                       WHERE RDB$RELATION_NAME = :RNAME
                       ORDER BY RDB$FIELD_POSITION );
BEGIN
  FOR
    SELECT RDB$RELATION_NAME
```

```
    FROM RDB$RELATIONS
    INTO :RNAME
  DO
  BEGIN
    OPEN C;
    FETCH C INTO :FNAME;
    CLOSE C;
    SUSPEND;
  END
END
```

Note

- Cursor declaration is allowed only in the declaration section of a PSQL block/procedure/trigger, as with any regular local variable declaration.
- Cursor names are required to be unique in the given context. They must not conflict with the name of another cursor that is "announced", via the AS CURSOR clause, by a FOR SELECT cursor. However, a cursor can share its name with any other type of variable within the same context, since the operations available to each are different.
- Positioned updates and deletes with cursors using the WHERE CURRENT OF clause are allowed.
- Attempts to fetch from or close a FOR SELECT cursor are prohibited.
- Attempts to open a cursor that is already open, or to fetch from or close a cursor that is already closed, will fail.
- All cursors which were not explicitly closed will be closed automatically on exit from the current PSQL block/procedure/trigger.
- The ROW_COUNT system variable can be used after each FETCH statement to check whether any row was returned.

Defaults for Stored Procedure Arguments

V. Khorsun

Defaults can now be declared for stored procedure arguments.

The syntax is the same as a default value definition for a column or domain, except that you can use '=' in place of 'DEFAULT' keyword.

Arguments with default values must be last in the argument list; that is, you cannot declare an argument that has no default value after any arguments that have been declared with default values. The caller must supply the values for all of the arguments preceding any that are to use their defaults.

For example, it is illegal to do something like this: `supply arg1, arg2, miss arg3, set arg4...`

Substitution of default values occurs at run-time. If you define a procedure with defaults (say P1), call it from another procedure (say P2) and skip some final, defaulted arguments, then the default values for P1 will be substituted by the engine at time execution P1 starts. This means that, if you change the default values for P1, it is not necessary to recompile P2.

However, it is still necessary to disconnect all client connections, as discussed in the Borland InterBase 6 beta "Data Definition Guide" (DataDef.pdf), in the section "Altering and dropping procedures in use".

Examples

```
CONNECT ... ;
SET TERM ^;
CREATE PROCEDURE P1 (X INTEGER = 123)
RETURNS (Y INTEGER)
AS
BEGIN
  Y = X;
  SUSPEND;
END ^
COMMIT ^
SET TERM ;^

SELECT * FROM P1;

           Y
============

         123

EXECUTE PROCEDURE P1;

           Y
============
         123

SET TERM ^;
CREATE PROCEDURE P2
RETURNS (Y INTEGER)
AS
BEGIN
  FOR SELECT Y FROM P1 INTO :Y
  DO SUSPEND;
END ^
COMMIT ^
SET TERM ;^

SELECT * FROM P2;

           Y
============

         123

SET TERM ^;
ALTER PROCEDURE P1 (X INTEGER = CURRENT_TRANSACTION)
         RETURNS (Y INTEGER)
AS
BEGIN
  Y = X;
 SUSPEND;
END; ^
COMMIT ^
SET TERM ;^

SELECT * FROM P1;

           Y
============
```

```
        5875

SELECT * FROM P2;

           Y
============

         123

COMMIT;

CONNECT  ... ;

SELECT * FROM P2;

           Y
============

        5880
```

> **Note**
>
> The source and BLR for the argument defaults are stored in RDB$FIELDS.

LEAVE <label> Syntax Support

D. Yemanov

New `LEAVE <label>` syntax now allows PSQL loops to be marked with labels and terminated in Java style. The purpose is to stop execution of the current block and unwind back to the specified label. After that execution resumes at the statement following the terminated loop.

Syntax pattern

```
<label_name>: <loop_statement>
...
LEAVE [<label_name>]
```

where <loop_statement> is one of: WHILE, FOR SELECT, FOR EXECUTE STATEMENT.

Examples

1.

```
FOR
  SELECT COALESCE(RDB$SYSTEM_FLAG, 0), RDB$RELATION_NAME
    FROM RDB$RELATIONS
    ORDER BY 1
  INTO :RTYPE, :RNAME
  DO
  BEGIN
    IF (RTYPE = 0) THEN
      SUSPEND;
```

```
    ELSE
      LEAVE; -- exits current loop
  END
```

2.

```
CNT = 100;
L1:
WHILE (CNT >= 0) DO
BEGIN
  IF (CNT < 50) THEN
    LEAVE L1; -- exists WHILE loop
  CNT = CNT - l;
END
```

3.

```
STMT1 = 'SELECT RDB$RELATION_NAME FROM RDB$RELATIONS';
L1:
FOR
  EXECUTE STATEMENT :STMT1 INTO :RNAME
DO
BEGIN
  STMT2 = 'SELECT RDB$FIELD_NAME FROM RDB$RELATION_FIELDS
    WHERE RDB$RELATION_NAME = ';
  L2:
  FOR
    EXECUTE STATEMENT :STMT2 || :RNAME INTO :FNAME
  DO
  BEGIN
    IF (RNAME = 'RDB$DATABASE') THEN
      LEAVE L1; -- exits the outer loop
    ELSE IF (RNAME = 'RDB$RELATIONS') THEN
      LEAVE L2; -- exits the inner loop
    ELSE
      SUSPEND;
  END
END
```

> **Note**
>
> Note that LEAVE without an explicit label means interrupting the current (innermost) loop.

OLD Context Variables Now Read-only

D. Yemanov

The set of OLD context variables available in trigger modules is now read-only. An attempt to assign a value to OLD.something will be rejected.

> **Note**
>
> NEW context variables are now read-only in AFTER-triggers as well.

PSQL Stack Trace

V. Khorsun

The API client can now extract a simple stack trace Error Status Vector when an exception occurs during PSQL execution (stored procedures or triggers). A stack trace is represented by one string (2048 bytes max.) and consists of all the stored procedure and trigger names, starting from the point where the exception occurred, out to the outermost caller. If the actual trace is longer than 2Kb, it is truncated.

Additional items are appended to the status vector as follows:

```
isc_stack_trace, isc_arg_string, <string length>, <string>
```

`isc_stack_trace` is a new error code with value of 335544842L.

Examples

Metadata creation

```
CREATE TABLE ERR (
  ID INT NOT NULL PRIMARY KEY,
  NAME VARCHAR(16));

CREATE EXCEPTION EX '!';
SET TERM ^;

CREATE OR ALTER PROCEDURE ERR_1 AS
BEGIN
  EXCEPTION EX 'ID = 3';
END ^

CREATE OR ALTER TRIGGER ERR_BI FOR ERR
  BEFORE INSERT AS
BEGIN
  IF (NEW.ID = 2)
  THEN EXCEPTION EX 'ID = 2';

  IF (NEW.ID = 3)
  THEN EXECUTE PROCEDURE ERR_1;

  IF (NEW.ID = 4)
  THEN NEW.ID = 1 / 0;
END ^

CREATE OR ALTER PROCEDURE ERR_2 AS
BEGIN
  INSERT INTO ERR VALUES (3, '333');
END ^
```

1. User exception from a trigger:

```
SQL" INSERT INTO ERR VALUES (2, '2');
Statement failed, SQLCODE = -836
exception 3
```

```
-ID = 2
-At trigger 'ERR_BI'
```

2. User exception from a procedure called by a trigger:

```
SQL" INSERT INTO ERR VALUES (3, '3');
Statement failed, SQLCODE = -836
exception 3
-ID = 3
-At procedure 'ERR_1'
At trigger 'ERR_BI'
```

3. Run-time exception occurring in trigger (division by zero):

```
SQL" INSERT INTO ERR VALUES (4, '4');
Statement failed, SQLCODE = -802
arithmetic exception, numeric overflow, or string truncation
-At trigger 'ERR_BI'
```

4. User exception from procedure:

```
SQL" EXECUTE PROCEDURE ERR_1;
Statement failed, SQLCODE = -836
exception 3
-ID = 3
-At procedure 'ERR_1'
```

5. User exception from a procedure with a deeper call stack:

```
SQL" EXECUTE PROCEDURE ERR_2;
Statement failed, SQLCODE = -836
exception 3
-ID = 3
-At procedure 'ERR_1'
At trigger 'ERR_BI'
At procedure 'ERR_2'
```

Call a UDF as a Void Function (Procedure)

N. Samofatov

In PSQL, supported UDFs, e.g. RDB$SET_CONTEXT, can be called as though they were void functions (a.k.a “procedures” in Object Pascal). For example:

```
BEGIN
   ...
   RDB$SET_CONTEXT('USER_TRANSACTION', 'MY_VAR', '123');
   ...
END
```

Chapter 8

New Reserved Words and Changes

The following keywords have been added, or have changed status, since Firebird 1.5. Those marked with an asterisk (*) are not present in the SQL standard.

Newly Reserved Words

```
BIT_LENGTH (v.2.0)
BOTH (v.2.0)
CHAR_LENGTH (v.2.0)
CHARACTER_LENGTH (v.2.0)
CLOSE (v.2.0)
CONNECT (v.2.1) <<—
CROSS (v.2.0)
DISCONNECT (v.2.1) <<—
FETCH (v.2.0)
GLOBAL  (v.2.1) <<—
INSENSITIVE  (v.2.1) <<—
LEADING (v.2.0)
LOWER (v.2.0)
OCTET_LENGTH (v.2.0)
OPEN (v.2.0)
ROWS (v.2.0)
START (v.2.1) <<—
TRAILING (v.2.0)
TRIM (v.2.0)
```

Changed from Non-reserved to Reserved

```
USING (v.2.0)
```

Keywords Added as Non-reserved

```
ABS  (v.2.1) <<—
ACCENT * (v.2.1) <<—
ACOS * (v.2.1) <<—
ALWAYS * (v.2.1) <<—
ASCII_CHAR * (v.2.1) <<—
ASCII_VAL * (v.2.1) <<—
ASIN * (v.2.1) <<—
ATAN * (v.2.1) <<—
```

```
ATAN2 * (v.2.1) <<—
BACKUP * (v.2.0)
BIN_AND * (v.2.1) <<—
BIN_OR * (v.2.1) <<—
BIN_SHL * (v.2.1) <<—
BIN_SHR * (v.2.1) <<—
BIN_XOR * (v.2.1) <<—
BLOCK * (v.2.0)
CEIL (v.2.1) <<—
COLLATION  (v.2.0)
COMMENT * (v.2.0)
COS * (v.2.1) <<—
COSH * (v.2.1) <<—
COT * (v.2.1) <<—
DATEADD * (v.2.1) <<—
DATEDIFF * (v.2.1) <<—
DECODE * (v.2.1) <<—
DIFFERENCE *  (v.2.0)
EXP (v.2.1) <<—
FLOOR (v.2.1) <<—
GEN_UUID (v.2.1) <<—
GENERATED (v.2.1) <<—
HASH * (v.2.1) <<—
IIF * (v.2.0)
LIST * (v.2.1) <<—
LN (v.2.1) <<—
LOG * (v.2.1) <<—
LOG10 * (v.2.1) <<—
LPAD * (v.2.1) <<—
MATCHED (v.2.1) <<—
MATCHING * (v.2.1) <<—
MAXVALUE * (v.2.1) <<—
MILLISECOND * (v.2.1) <<—
MINVALUE * (v.2.1) <<—
MOD (v.2.1) <<—
NEXT (v.2.0)
OVERLAY (v.2.1) <<—
PAD (v.2.1) <<—
PI * (v.2.1) <<—
PLACING (v.2.1) <<—
POWER (v.2.1) <<—
PRESERVE (v.2.1) <<—
RAND * (v.2.1) <<—
REPLACE * (v.2.1) <<—
RESTART (v.2.0)
RETURNING * (v.2.0)
REVERSE * (v.2.1) <<—
ROUND * (v.2.1) <<—
RPAD * (v.2.1) <<—
SCALAR_ARRAY * (v.2.0)
SEQUENCE (v.2.0)
SIGN * (v.2.1) <<—
SIN * (v.2.1) <<—
SINH * (v.2.1) <<—
SPACE (v.2.1) <<—
SQRT (v.2.1) <<—
TAN * (v.2.1) <<—
TANH * (v.2.1) <<—
TEMPORARY (v.2.1) <<—
TRUNC * (v.2.1) <<—
WEEK * (v.2.1) <<—
```

Keywords No Longer Reserved

```
ACTION (v.2.0)
CASCADE (v.2.0)
FREE_IT * (v.2.0)
RESTRICT (v.2.0)
ROLE (v.2.0)
TYPE (v.2.0)
WEEKDAY * (v.2.0)
YEARDAY * (v.2.0)
```

No Longer Reserved as Keywords

```
BASENAME * (v.2.0)
GROUP_COMMIT_WAIT * (v.2.0)
NUM_LOG_BUFS * (v.2.0)
CACHE * (v.2.0)
LOGFILE * (v.2.0)
RAW_PARTITIONS * (v.2.0)
CHECK_POINT_LEN * (v.2.0)
LOG_BUF_SIZE * (v.2.0)
```

Chapter 9

Indexing & Optimizations

Optimizations in V.2.1

Optimization improvements in v.2.1 include:

(v.2.1) *Economising on Indexed Reads for MIN() and MAX()*
Indexed MIN/MAX aggregates would produce three indexed reads instead of the expected single read. So, with an ASC index on the non-nullable COL, the query

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

should be completely equivalent, to

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

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

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

Improved PLAN Clause

D. Yemanov

(V.2.0.x) A PLAN clause optionally allows you to provide your own instructions to the engine and have it ignore the plan supplied by the optimizer. Firebird 2 enhancements allow you to specify more possible paths for the engine. For example:

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

For more details, please refer to the topic Query Plans Improvements in the DML chapter.

Optimizer Improvements

This section represents a collection of changes done in Firebird 2 to optimize many aspects of performance.

For All Databases

The first group of changes affect all databases, including those not yet upgraded to ODS 11.x.

Some General Improvements

O. Loa, D. Yemanov

- Much faster algorithms to process the dirty pages tree

  Firebird 2 offers a more efficient processing of the list of modified pages, a.k.a. the dirty pages tree. It affects all kinds of batch data modifications performed in a single transaction and eliminates the known issues with performance getting slower when using a buffer cache of >10K pages.

  This change also improves the overall performance of data modifications.

- Increased maximum page cache size to 128K pages (2GB for 16K page size)

Faster Evaluation of IN() and OR

O. Loa

Constant IN predicate or multiple OR booleans are now evaluated faster.

Sparse bitmap operations were optimized to handle multiple OR booleans or an IN (<constant list>) predicate more efficiently, improving performance of these operations.

Improved UNIQUE Retrieval

A. Brinkman

The optimizer will now use a more realistic cost value for unique retrieval.

More Optimization of NOT Conditions

D. Yemanov

NOT conditions are simplified and optimized via an index when possible.

Example

```
(NOT NOT A = 0) -> (A = 0)
(NOT A > 0) -> (A <= 0)
```

Distribute HAVING Conjunctions to the WHERE Clause

If a HAVING clause or any outer-level select refers to a field being grouped by, this conjunct is distributed deeper in the execution path than the grouping, thus allowing an index scan to be used. In other words, it allows the HAVING clause not only be treated as the WHERE clause in this case, but also be optimized the same way.

Examples

```
select rdb$relation_id, count(*)
from rdb$relations
group by rdb$relation_id
having rdb$relation_id > 10

select * from (
  select rdb$relation_id, count(*)
  from rdb$relations
  group by rdb$relation_id
  ) as grp (id, cnt)
where grp.id > 10
```

In both cases, an index scan is performed instead of a full scan.

Distribute UNION Conjunctions to the Inner Streams

Distribute UNION conjunctions to the inner streams when possible.

Improved Handling of CROSS JOIN and Merge/SORT

Improved cross join and merge/sort handling

Better Choice of Join Order for Mixed Inner/Outer Joins

reasonable join order for intermixed inner and outer joins

Equality Comparison on Expressions

MERGE PLAN may now be generated for joins using equality comparsion on expressions

For ODS 11 Databases only

This group of optimizations affects databases that were created or restored under Firebird 2 or higher.

Segment-level Selectivities are Used

See Selectivity Maintenance per Segment.

Better Support for IS NULL and STARTING WITH

Previously, IS NULL and STARTING WITH predicates were optimized separately from others, thus causing non-optimal plans in complex ANDed/ORed boolean expressions. From v2.0 and ODS11, these predicates are optimized in a regular way and hence benefit from all possible optimization strategies.

Matching of Both OR and AND Nodes to Indexes

Complex boolean expressions consisting of many AND/OR predicates are now entirely mapped to available indices if at all possible. Previously, such complex expressions could be optimized badly.

Better JOIN Orders

Cost estimations have been improved in order to improve JOIN orders.

Indexed Order Enabled for Outer Joins

It is now possible for indexed order to be utilised for outer joins, i.e. navigational walk.

Enhancements to Indexing

252-byte index length limit is gone

A. Brinkman

New and reworked index code is very fast and tolerant of large numbers of duplicates. The old aggregate key length limit of 252 bytes is removed. Now the limit depends on page size: the maximum size of the key in bytes is 1/4 of the page size (512 on 2048, 1024 on 4096, etc.)

A 40-bit record number is included on “non leaf-level pages” and duplicates (key entries) are sorted by this number.

Expression Indexes

O. Loa, D. Yemanov, A. Karyakin

Arbitrary expressions applied to values in a row in dynamic DDL can now be indexed, allowing indexed access paths to be available for search predicates that are based on expressions.

Syntax Pattern

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

Examples

1.

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

2.

```
CREATE INDEX IDX2 ON T2
  COMPUTED BY ( EXTRACT(YEAR FROM COL2) || EXTRACT(MONTH FROM COL2) );
COMMIT;
/*  */
SELECT * FROM T2
  ORDER BY EXTRACT(YEAR FROM COL2) || EXTRACT(MONTH FROM COL2)
-- PLAN (T2 ORDER IDX2)
```

Note

1. The expression used in the predicate must match *exactly* the expression used in the index declaration, in order to allow the engine to choose an indexed access path. The given index will not be available for any retrieval or sorting operation if the expressions do not match.

2. Expression indices have exactly the same features and limitations as regular indices, except that, by definition, they cannot be composite (multi-segment).

Changes to Null keys handling

V. Khorsun, A. Brinkman

- Null keys are now bypassed for uniqueness checks. (V. Khorsun)

  If a new key is inserted into a unique index, the engine skips all NULL keys before starting to check for key duplication. It means a performance benefit as, from v.1.5 on, NULLs have not been considered as duplicates.

- NULLs are ignored during the index scan, when it makes sense to ignore them. (A. Brinkman).

  Prevously, NULL keys were always scanned for all predicates. Starting with v.2.0, NULL keys are usually skipped before the scan begins, thus allowing faster index scans.

Note

The predicates IS NULL and IS NOT DISTINCT FROM still require scanning of NULL keys and they disable the aforementioned optimization.

Improved Index Compression

A. Brinkman

A full reworking of the index compression algorithm has made a manifold improvement in the performance of many queries.

Selectivity Maintenance per Segment

D. Yemanov, A. Brinkman

Index selectivities are now stored on a per-segment basis. This means that, for a compound index on columns (A, B, C), three selectivity values will be calculated, reflecting a full index match as well as all partial matches. That is to say, the selectivity of the multi-segment index involves those of segment A alone (as it would be if it were a single-segment index), segments A and B combined (as it would be if it were a double-segment index) and the full three-segment match (A, B, C), i.e., all the ways a compound index can be used.

This opens more opportunities to the optimizer for clever access path decisions in cases involving partial index matches.

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

Chapter 10

International Language Support (INTL)

Adriano dos Santos Fernandes

This chapter describes the new international language support interface that was introduced with Firebird 2. Since then, a number of additions and improvements have been added, including the ability to implement UNICODE collations from external libraries generically. New DDL syntax has been introduced to assist with this task, in the form of the CREATE COLLATION statement.

New INTL Interface for Non-ASCII Character Sets

A. dos Santos Fernandes

Originally described by N. Samofatov, Firebird 2's new interface for international character sets features many enhancements that have been implemented by me.

Architecture

Firebird allows character sets and collations to be declared in any character field or variable declaration. The default character set can also be specified at database create time, to cause every CHAR/VARCHAR declaration that does not specifically include a CHARACTER SET clause to use this default.

At attachment time you normally specify the character set that the *client* is to use to read strings. If no "client" (or "connection") character set is specified, character set NONE is assumed.

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

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

Enhancements

Enhancements that the new system brings include:

Well-formedness checks

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

Uppercasing

In Firebird 1.5.x, only the ASCII-equivalent characters are uppercased in any character set's default (binary) collation order, which is the one that is used if no collation is specified.

For example,

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

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

In Firebird 2 the result is:

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

Maximum String Length

In v.1.5.x the engine does not verify the logical length of multi-byte character set (MBCS) strings. Hence, a UNICODE_FSS field takes three times as many characters as the declared field size, three being the maximum length of one UNICODE_FSS character.

This has been retained for compatibility for legacy character sets. However, new character sets (UTF8, for example) do not inherit this limitation.

sqlsubtype and Attachment Character Set

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

Enhancements for BLOBs

Several character set-related enhancements have been added for text BLOBs.

COLLATE clauses for BLOBs

A DML COLLATE clause is now allowed with BLOBs.

Example

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

Full equality comparisons between BLOBs

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

Character set conversion for BLOBs

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

INTL Plug-ins

Character sets and collations are installed using a *manifest file*.

The manifest file should be put in the $rootdir/intl with a .conf extension. It is used to locate character sets and collations in the libraries. If a character set/collation is declared more than once, it is not loaded and the error is reported in the log.

The file `/intl/fbintl.conf` is an example of a manifest file. The following snippet is an excerpt from `/intl/fbintl.conf`:

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

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

<charset WIN1250>
```

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

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

New Character Sets/Collations

Two character sets introduced in Firebird 2 will be of particular interest if you have struggled with the shortcomings of UNICODE_FSS in past versions.

UTF8 character set

The UNICODE_FSS character set has a number of problems: it's an old version of UTF8 that accepts malformed strings and does not enforce correct maximum string length. In FB 1.5.X UTF8 is an alias to UNICODE_FSS.

Now, UTF8 is a new character set, without the inherent problems of UNICODE_FSS.

UNICODE collations (for UTF8)

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

Sort order sample:

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

C
======
A
B
a
b
á

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

C
```

```
======
a
A
á
b
B
```

Developments in V.2.1

The 2.1 release sees further capabilities implemented for

1. using ICU charsets through fbintl
2. UNICODE collation (charset_UNICODE) being available for all *fbintl* charsets
3. using collation attributes
4. CREATE/DROP COLLATION statements
5. SHOW COLLATION and collation extraction in ISQL
6. Verifying that text blobs are well-formed
7. Transliterating text blobs automatically

ICU Character Sets

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

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

Registering an ICU Character Set Module

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

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

Registering a Character Set on the Server

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

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

Registering a Character Set in a Database

To register the module in a database, you have two options:

- Use the CREATE COLLATION statement, OR—
- Run the procedure `sp_register_character_set`, the source for which can be found in `misc/intl.sql` beneath your Firebird 2.1 root

The CREATE COLLATION Statement

Syntax for CREATE COLLATION

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

> **Note**
>
> Specific attributes should be separated by semicolon and are case sensitive.

Examples

```
/* 1 */
CREATE COLLATION UNICODE_ENUS_CI
    FOR UTF8
    FROM UNICODE
    CASE INSENSITIVE
    'LOCALE=en_US';
/* 2 */
CREATE COLLATION NEW_COLLATION
    FOR WIN1252
    PAD SPACE;

/* NEW_COLLATION should be declared in .conf file
   in $root/intl directory */
```

Using the Stored Procedure

A Sample

Here is the sample declaration in `fbintl.conf`:

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

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

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

The UNICODE Collations

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

Naming Conventions

The naming convention you should use is `charset_collation`. For example,

```
create collation win1252_unicode
  for win1252;

create collation win1252_unicode_ci
  for win1252
   from win1252_unicode
   case insensitive;
```

> **Note**
>
> The character set name should be as in fbintl.conf (i.e. ISO8859_1 instead of ISO88591, for example).

Specific Attributes for Collations

> **Note**
>
> Some attributes may not work with some collations, even though they do not report an error.

DISABLE-COMPRESSIONS
Disable compressions (aka contractions) changing the order of a group of characters.

Valid for collations of narrow character sets.

Format: DISABLE-COMPRESSIONS={0 | 1}

Example

DISABLE-COMPRESSIONS=1

DISABLE-EXPANSIONS
Disable expansions changing the order of a character to sort as a group of characters.

Valid for collations of narrow character sets.

Format: DISABLE-EXPANSIONS={0 | 1}

Example

DISABLE-EXPANSIONS=1

ICU-VERSION
Specify what version of ICU library will be used. Valid values are the ones defined in the config file (intl/fbintl.conf) in entry intl_module/icu_versions.

Valid for UNICODE and UNICODE_CI.

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

Example

ICU-VERSION=3.0

LOCALE
Specify the collation locale.

Valid for UNICODE and UNICODE_CI. Requires complete version of ICU libraries.

Format: LOCALE=xx_XX

Example

LOCALE=en_US

MULTI-LEVEL
Uses more than one level for ordering purposes.

Valid for collations of narrow character sets.

Format: MULTI-LEVEL={0 | 1}

Example

MULTI-LEVEL=1

SPECIALS-FIRST
Order special characters (spaces, symbols, etc) before alphanumeric characters.

Valid for collations of narrow character sets.

Format: SPECIALS-FIRST={0 | 1}

Example

SPECIALS-FIRST=1

Collation Changes in V.2.1

Spanish

ES_ES (as well as the new ES_ES_CI_AI) collation automatically uses attributes DISABLE-COMPRESSIONS=1;SPECIALS-FIRST=1.

> **Note**
>
> The attributes are stored at database creation time, so the changes do not apply to databases with ODS < 11.1.

The ES_ES_CI_AI collation was standardised to current usage.

UTF-8
Case-insensitive collation for UTF-8. See feature request CORE-972

Metadata Text Conversion

Firebird versions 2.0.x had two problems related to character sets and metadata extraction:

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

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

   > **Note**
   >
   > Even in the current version (2.1 Beta 1) the problem can still occur if CREATE or ALTER operations are performed with the connection character set as NONE or UNICODE_FSS and you are using non-UNICODE_FSS data.

2. In reads from text BLOBs, transliteration from the BLOB character set to the client character set was not being performed.

Repairing Your Metadata Text

If your metadata text was created with non-ASCII encoding, you need to repair your database in order to read the metadata correctly after upgrading it to v.2.1.

> **Important**
>
> The procedure involves multiple passes through the database, using scripts. It is strongly recommended that you disconnect and reconnect before each pass.
>
> The database should already have been converted to ODS11.1 by way of a gbak backup and restore.
>
> Before doing anything, make a copy of the database.
>
> In the examples that follow, the string *$fbroot$* represents the path to your Firebird installation root directory, e.g. */opt/firebird*.

Create the procedures in the database

```
[1] isql /path/to/your/database.fdb
[2] SQL> input '$fbroot$/misc/upgrade/metadata/metadata_charset_create.sql';
```

Check your database

```
[1] isql /path/to/your/database.fdb
[2] SQL> select * from rdb$check_metadata;
```

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

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

Fixing the metadata

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

```
[1] isql /path/to/your/database.fdb
[2] SQL> input '$fbroot$/misc/upgrade/metatdata/metadata_charset_create.sql';
[3] SQL> select * from rdb$fix_metadata('WIN1252');  -- replace WIN1252 by your charset
[4] SQL> commit;
```

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

> **Important**
>
> It should be run once!

After this, you can remove the upgrade procedures.

Remove the upgrade procedures

```
[1] isql /path/to/your/database.fdb
[2] SQL> input '$fbroot$/misc/upgrade/metadata/metadata_charset_drop.sql';
```

Supported Character Sets

See Appendix B at the end of these notes, for a full listing of the supported character sets.

Chapter 11

Administrative Features

Firebird is gradually adding new features to assist in the administration of databases. Firebird 2.1 sees the introduction of a new set of system tables through which administrators can monitor transactions and statements that are active in a database. These facilities employ a new v.2.1 DDL feature, Global Temporary Tables to provide snapshots.

Monitoring Tables

Dmitry Yemanov

Firebird 2.1 introduces the ability to monitor server-side activity happening inside a particular database. The engine offers a set of so-called “virtual” tables that provides the user with a snapshot of the current activity within the given database.

The word “virtual” means that the table data is not materialised until explicitly asked for. However, the metadata of the virtual table is stable and can be retrieved from the schema.

> **Note**
>
> Virtual monitoring tables exist only in ODS 11.1 (and higher) databases, so a migration via backup/restore is required in order to use this feature.

The Concept

The key term of the monitoring feature is an *activity snapshot*. It represents the current state of the database, comprising a variety of information about the database itself, active attachments and users, transactions, prepared and running statements, and more.

A snapshot is created the first time any of the monitoring tables is being selected from in the given transaction and it is preserved until the transaction ends, in order that multiple-table queries (e.g., master-detail ones) will always return a consistent view of the data.

In other words, the monitoring tables always behave like a snapshot table stability (“consistency”) transaction, even if the host transaction has been started with a lower isolation level.

To refresh the snapshot, the current transaction should be finished and the monitoring tables should be queried in a new transaction context.

Scope and Security

- Access to the monitoring tables is available in both DSQL and PSQL.
- Complete database monitoring is available to SYSDBA and the database owner.
- Regular users are restricted to the information about their own attachments only—other attachments are invisible to them.

Metadata

MON$DATABASE (connected database)

- MON$DATABASE_NAME (database pathname or alias)
- MON$PAGE_SIZE (page size)
- MON$ODS_MAJOR (major ODS version)
- MON$ODS_MINOR (minor ODS version)
- MON$OLDEST_TRANSACTION (OIT number)
- MON$OLDEST_ACTIVE (OAT number)
- MON$OLDEST_SNAPSHOT (OST number)
- MON$NEXT_TRANSACTION (next transaction number)
- MON$PAGE_BUFFERS (number of pages allocated in the cache)
- MON$SQL_DIALECT (SQL dialect of the database)
- MON$SHUTDOWN_MODE (current shutdown mode)
  0: online
  1: multi-user shutdown
  2: single-user shutdown
  3: full shutdown
- MON$SWEEP_INTERVAL (sweep interval)
- MON$READ_ONLY (read-only flag)
- MON$FORCED_WRITES (sync writes flag)
- MON$RESERVE_SPACE (reserve space flag)
- MON$CREATION_DATE (creation date/time)
- MON$PAGES (number of pages allocated on disk)
- MON$BACKUP_STATE (current physical backup state)
  0: normal
  1: stalled
  2: merge
- MON$STAT_ID (statistics ID)

MON$ATTACHMENTS (connected attachments)

- MON$ATTACHMENT_ID (attachment ID)
- MON$SERVER_PID (server process ID)
- MON$STATE (attachment state)
  0: idle
  1: active
- MON$ATTACHMENT_NAME (connection string)
- MON$USER (user name)
- MON$ROLE (role name)
- MON$REMOTE_PROTOCOL (remote protocol name)
- MON$REMOTE_ADDRESS (remote address)
- MON$REMOTE_PID (remote client process ID)
- MON$REMOTE_PROCESS (remote client process pathname)
- MON$CHARACTER_SET_ID (attachment character set)
- MON$TIMESTAMP (connection date/time)
- MON$GARBAGE_COLLECTION (garbage collection flag)
- MON$STAT_ID (statistics ID)

- columns MON$REMOTE_PID and MON$REMOTE_PROCESS contains non-NULL values only if the client library is version 2.1 or higher
- column MON$REMOTE_PROCESS can contain a non-pathname value if an application has specified a custom process name via DPB
- column MON$GARBAGE_COLLECTION indicates whether GC is allowed for this attachment (as specified via the DPB in *isc_attach_database*).

MON$TRANSACTIONS (started transactions)

```
 - MON$TRANSACTION_ID (transaction ID)
 - MON$ATTACHMENT_ID (attachment ID)
 - MON$STATE (transaction state)
     0: idle (state after prepare, until execution begins)
     1: active (state during execution and fetch.  Idle state
       returns after cursor is closed)
 - MON$TIMESTAMP (transaction start date/time)
 - MON$TOP_TRANSACTION (top transaction)
 - MON$OLDEST_TRANSACTION (local OIT number)
 - MON$OLDEST_ACTIVE (local OAT number)
 - MON$ISOLATION_MODE (isolation mode)
     0: consistency
     1: concurrency
     2: read committed record version
     3: read committed no record version
 - MON$LOCK_TIMEOUT (lock timeout)
     -1: infinite wait
     0: no wait
     N: timeout N
 - MON$READ_ONLY (read-only flag)
 - MON$AUTO_COMMIT (auto-commit flag)
 - MON$AUTO_UNDO (auto-undo flag)
 - MON$STAT_ID (statistics ID)
```

- MON$TOP_TRANSACTION is the upper limit used by the sweeper transaction when advancing the global OIT. All transactions above this threshold are considered active. It is normally equivalent to the MON$TRANSACTION_ID but COMMIT RETAINING or ROLLBACK RETAINING will cause MON $TOP_TRANSACTION to remain unchanged (“stuck”) when the transaction ID is incremented.
- MON$AUTO_UNDO indicates the auto-undo status set for the transaction, i.e., whether a transaction-level savepoint was created. The existence of the transaction-level savepoint allows changes to be undone if ROLLBACK is called and the transaction is then just committed. If this savepoint does not exist, or it does exist but the number of changes is very large, then an actual ROLLBACK is executed and the the transaction is marked in the TIP as “dead”.

MON$STATEMENTS (prepared statements)

```
 - MON$STATEMENT_ID (statement ID)
 - MON$ATTACHMENT_ID (attachment ID)
 - MON$TRANSACTION_ID (transaction ID)
```

- MON$STATE (statement state)
  0: idle
  1: active
- MON$TIMESTAMP (statement start date/time)
- MON$SQL_TEXT (statement text, if appropriate)
- MON$STAT_ID (statistics ID)

- column MON$SQL_TEXT contains NULL for GDML statements
- columns MON$TRANSACTION_ID and MON$TIMESTAMP contain valid values for active statements only
- The execution plan and the values of parameters are not available

MON$CALL_STACK (call stack of active PSQL requests)

- MON$CALL_ID (call ID)
- MON$STATEMENT_ID (top-level DSQL statement ID)
- MON$CALLER_ID (caller request ID)
- MON$OBJECT_NAME (PSQL object name)
- MON$OBJECT_TYPE (PSQL object type)
- MON$TIMESTAMP (request start date/time)
- MON$SOURCE_LINE (SQL source line number)
- MON$SOURCE_COLUMN (SQL source column number)
- MON$STAT_ID (statistics ID)

- column MON$STATEMENT_ID groups call stacks by the top-level DSQL statement that initiated the call chain. This ID represents an active statement record in the table MON$STATEMENTS.
- columns MON$SOURCE_LINE and MON$SOURCE_COLUMN contain line/column information related to the PSQL statement currently being executed

MON$IO_STATS (I/O statistics)

- MON$STAT_ID (statistics ID)
- MON$STAT_GROUP (statistics group)
  0: database
  1: attachment
  2: transaction
  3: statement
  4: call
- MON$PAGE_READS (number of page reads)
- MON$PAGE_WRITES (number of page writes)
- MON$PAGE_FETCHES (number of page fetches)
- MON$PAGE_MARKS (number of pages with changes pending)

MON$RECORD_STATS (record-level statistics)

- MON$STAT_ID (statistics ID)
- MON$STAT_GROUP (statistics group)
  0: database

1: attachment
2: transaction
3: statement
4: call

- MON$RECORD_SEQ_READS (number of records read sequentially)
- MON$RECORD_IDX_READS (number of records read via an index)
- MON$RECORD_INSERTS (number of inserted records)
- MON$RECORD_UPDATES (number of updated records)
- MON$RECORD_DELETES (number of deleted records)
- MON$RECORD_BACKOUTS (number of records where a new primary record version or a change to an existing primary record version is backed out due to rollback or savepoint undo)
- MON$RECORD_PURGES (number of records where record version chain is being purged of versions no longer needed by OAT or younger transactions)
- MON$RECORD_EXPUNGES (number of records where record version chain is being deleted due to deletions by transactions older than OAT)

> **Note**
>
> Textual descriptions of all “state” and “mode” values can be found in the system table RDB$TYPES.

Usage

Creation of a snapshot is usually quite a fast operation, but some delay could be expected under high load (especially in the Classic Server).

A valid database connection is required in order to retrieve the monitoring data. The monitoring tables return information about the attached database only. If multiple databases are being accessed on the server, each of them has to be connected to and monitored separately.

The system variables `CURRENT_CONNECTION` and `CURRENT_TRANSACTION` could be used to select data about the caller's current connection and transaction respectively. These variables correspond to the ID columns of the appropriate monitoring tables.

Examples

1. Retrieve IDs of all CS processes loading CPU at the moment

```
SELECT MON$SERVER_PID
  FROM MON$ATTACHMENTS
  WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
  AND MON$STATE = 1
```

2. Retrieve information about client applications

```
SELECT MON$USER, MON$REMOTE_ADDRESS,
    MON$REMOTE_PID,
```

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

3. Get isolation level of the current transaction

```
SELECT MON$ISOLATION_MODE
  FROM MON$TRANSACTIONS
  WHERE MON$TRANSACTION_ID = CURRENT_TRANSACTION
```

4. Get statements that are currently active

```
SELECT ATT.MON$USER,
       ATT.MON$REMOTE_ADDRESS,
       STMT.MON$SQL_TEXT,
       STMT.MON$TIMESTAMP
  FROM MON$ATTACHMENTS ATT
    JOIN MON$STATEMENTS STMT
      ON ATT.MON$ATTACHMENT_ID = STMT.MON$ATTACHMENT_ID
  WHERE ATT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION
    AND STMT.MON$STATE = 1
```

5. Retrieve call stacks for all connections

```
WITH RECURSIVE HEAD AS
  (
     SELECT CALL.MON$STATEMENT_ID,
            CALL.MON$CALL_ID,
            CALL.MON$OBJECT_NAME,
            CALL.MON$OBJECT_TYPE
     FROM MON$CALL_STACK CALL
       WHERE CALL.MON$CALLER_ID IS NULL
     UNION ALL
       SELECT CALL.MON$STATEMENT_ID,
              CALL.MON$CALL_ID,
              CALL.MON$OBJECT_NAME,
              CALL.MON$OBJECT_TYPE
       FROM MON$CALL_STACK CALL
         JOIN HEAD
           ON CALL.MON$CALLER_ID = HEAD.MON$CALL_ID
  )
  SELECT MON$ATTACHMENT_ID,
         MON$OBJECT_NAME,
         MON$OBJECT_TYPE
  FROM HEAD
    JOIN MON$STATEMENTS STMT
      ON STMT.MON$STATEMENT_ID = HEAD.MON$STATEMENT_ID
    WHERE STMT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

Cancel a Running Query

Runaway and long-running queries can now be cancelled *from a separate connection*.

There is no API function call directed at this feature. It will be up to the SysAdmin (SYSDBA or Owner) to make use of the data available in the monitoring tables and devise an appropriate mechanism for reining in the rogue statements.

Example

As a very rough example, the following statement will kill all statements currently running in the database, other than any that belong to the *separate* connection that the SysAdmin is using himself:

```
delete from mon$statements
  where mon$attachment_id <> current_connection
```

More Context Information

More context information about the server and database ('SYSTEM') is available via SELECT calls to the RDB $GET_CONTEXT function, including the engine version.

Example

```
SELECT RDB$GET_CONTEXT('SYSTEM', 'ENGINE_VERSION')
  FROM RDB$DATABASE
```

For detailed information about using these context calls, refer to the v.2.0.1 release notes.

Chapter 12

Security

In this chapter are details of the changes to Firebird security that came with the release of Firebird 2 and its successors. Further changes and enhancements introduced in V.2.1 are highlighted.

Summary of Changes

Improving security has had a lot of focus in Firebird 2.0 development. The following is a summary of the major changes.

New security database

The new security database is renamed as `security2.fdb`. Inside, the user authentication table, where user names and passwords are stored, is now called `RDB$USERS`. There is no longer a table named “users” but a new *view* over RDB$USERS that is named “USERS”. Through this view, users can change their passwords.

For details of the new database, see New Security Database in the section about authentication later in this chapter.

For instructions on updating previous security databases, refer to the section Dealing with the New Security Database at the end of this chapter.

Using Windows Security to Authenticate Users

(V.2.1) From Firebird 2.1 onward, Windows “Trusted User” security can be applied for authenticating Firebird users on a Windows host. The Trusted User's security context is passed to the Firebird server instead of the Firebird user name and password and, if it succeeds, it is used to determine the Firebird security user name.

For details see the section below, Windows Trusted User Security.

Better password encryption

A. Peshkov

Password encryption/decryption now uses a more secure password hash calculation algorithm.

Users can modify their own passwords

A. Peshkov

The SYSDBA remains the keeper of the security database. However, users can now modify their own passwords.

Non-server access to security database is rejected

A. Peshkov

gsec now uses the Services API. The server will refuse any access to security2.fdb except through the Services Manager.

Active protection from brute-force attack

A. Peshkov

Attempts to get access to the server using brute-force techniques on accounts and passwords are now detected and locked out.

- Login with password is required from any remote client
- Clients making too many wrong login attempts are blocked from further attempts for a period

Support for brute-force attack protection has been included in both the attachment functions of the Firebird API and the Services API. For more details, see Protection from Brute-force Hacking

Vulnerabilities have been closed

A. Peshkov, C. Valderrama

Several known vulnerabilities in the API have been closed.

> **Caution**
>
> It must be noted that the restoration of the **server redirection ("multi-hop") capability** to Firebird 2 potentially throws up a new vulnerability. For that reason, it is controlled by a parameter (Redirection) in `firebird.conf`, which you should not enable unless you really understand its implications.
>
> These days, the ability to redirect requests to other servers is dangerous. Suppose you have one carefully protected firebird server, access to which is possible from the Internet. In a situation where this server has unrestricted access to your internal LAN, it will work as a gateway for incoming requests like `firebird.your.domain.com:internal_server:/private/database.fdb` .
>
> Knowing the name or IP address of some internal server on your LAN is enough for an intruder: he does note even need login access to the external server. Such a gateway easily overrides a firewall that is protecting your LAN from outside attack.

Details of the Security Changes in Firebird 2

Security focus was directed at some recognised weaknesses in Firebird's security from malicious attacks:

- the lack of brute-force resistant passwords encryption in the security database
- the ability for any remote user with a valid account to open the security database and read hashes from it (especially interesting in combination with the first point)
- the inability for users to change their own passwords

- the lack of protection against remote brute-forcing of passwords on the server directly

Authentication

Firebird authentication checks a server-wide security database in order to decide whether a database or server connection request is authorised. The security database stores the user names and passwords of all authorised login identities.

Firebird 1.5 Authentication

In Firebird 1.5 the DES algorithm is used twice to hash the password: first by the client, then by the server, before comparing it with the hash stored in security database. However, this sequence becomes completely broken when the SYSDBA changes a password. The client performs the hash calculation twice and stores the resulting hash directly in the security database. Therefore, hash management is completely client-dependent (or, actually, client-defined).

Firebird 2: Server-side Hashing

To be able to use stronger hashes, another approach was called for. The hash to be stored on the server should always be calculated on the server side. Such a schema already exists in Firebird -- in the Services API. This led to the decision to use the Services API for any client activity related to user management. Now, *gsec* and the isc_user_add(modify, delete) API functions all use services to access the security database. (Embedded access to Classic server on POSIX is the exception --see below).

It became quite easy to make any changes to the way passwords are hashed - it is always performed by the server. It is no longer *gsec*'s problem to calculate the hash for the security database: it simply asks services to do the work!

It is worth noting that the new *gsec* works successfully with older Firebird versions, as long as the server's architecture supports services.

The SHA-1 Hashing Algorithm

This method leads to the situation where

1. a hash valid for user A is invalid for user B

2. when a user changes his password -- even to exactly the same string as before -- the data stored in RDB $USERS.RDB$PASSWD is new.

Although this situation does not increase resistance to a brute-force attempt to crack the password, it does make "visual" analysis of a stolen password database much harder.

The New Security Database

The structure of security database was changed. In general, now it contains a patch by Ivan Prenosil, with some minor differences, enabling any user to change his/her own password, .

- In firebird 1.5 the table USERS has to be readable by PUBLIC, an engine requirement without which the password validation process would fail. Ivan's patch solution used a view, with the condition "WHERE USER

= "". That worked due to another bug in the engine that left the SQL variable USER empty, not 'authenticator', as it might seem from engine's code.

Once that bug was fixed, it was certainly possible to add the condition "USER = 'authenticator'". For the short term, that was OK, because the username is always converted to upper case.

- A better solution was found, that avoids making user authentication depend on an SQL trick. The result is that the non-SYSDBA user can see only his own login in any user-management tool (*gsec*, or any graphical interface that use the Services API). SYSDBA continues to have full access to manage users' accounts.

New security database structure

The Firebird 2 security database is named `security2.fdb`. For user authentication it has a new table named `RDB$USERS` that stores the new hashed passwords. A view over this table replaces the old USERS table and enables users to change their own passwords.

The DDL for the new structures can be found in Appendix C.

gsec in Firebird 2

Special measures were thus taken to make remote connection to the security database completely impossible. Don't be surprised if some old program fails on attempting direct access: this is by design. Users information may now be accessed only through the Services API and the equivalent internal access to services now implemented in the `isc_user_*` API functions.

Protection from Brute-force Hacking

Current high-speed CPUs and fast WAN connections make it possible to try to brute-force Firebird server users' passwords. This is especially dangerous for Superserver which, since Firebird 1.5, performs user authentication very fast. Classic is slower, since it has to create new process for each connection, attach to the security database within that connection and compile a request to the table RDB$USERS before validating login and password. Superserver caches the connection and request, thus enabling a much faster user validation.

Given the 8-byte maximum length of the traditional Firebird password, the brute-force hacker had a reasonable chance to break into the Firebird installation.

The v.2.0 Superserver has active protection to make a brute-force attack more difficult. After a few failed attempts to log in, the user and IP address are locked for a few seconds, denying any attempt to log in with that particular user name OR from that particular IP address for a brief period.

No setup or configuration is required for this feature. It is active automatically as soon as the Firebird 2.0 SuperServer starts up.

Using Windows Security to Authenticate Users

Alex Peshkov

(V.2.1) From Firebird 2.1 onward, Windows “Trusted User” security can be applied for authenticating Firebird users on a Windows host. The Trusted User's security context is passed to the Firebird server and, if it succeeds, it is used to determine the Firebird security user name.

Simply omitting the user and password parameters from the DPB/SPB will automatically cause Windows Trusted User authentication to be applied, in almost all cases. See the Environment section, below, for exceptions.

Illustration

Suppose you have logged in to the Windows server SRV as user 'John'. If you connect to server SRV with *isql*, without specifying a Firebird user name and password:

```
isql srv:employee
```

and do:

```
SQL> select CURRENT_USER from rdb$database;
```

you will get something like:

```
USER
==================================================
SRV\John
```

SQL Privileges

Windows users can be granted rights to access database objects and roles in the same way as regular Firebird users, emulating the capability that has been always been available users of Unix and Linux hosted Firebird databases.

Administrators

If a member of the built-in Domain Admins group connects to Firebird using trusted authentication, he/she will be connected as SYSDBA.

Configuration Parameter “Authentication”

The new parameter `Authentication` has been added to firebird.conf for configuring the authentication method on Windows. Possible values are.-

Authentication = Native
: Provides full compatibility with previous Firebird versions, avoiding trusted authentication.

Authentication = Trusted
: The Security database is ignored and only Windows authentication is used. In some respects, on Windows this is more secure than Native, in the sense that it is no less and no more secure than the security of the host operating system.

Authentication = Mixed
: This is the default setting.

  To retain the legacy behaviour, when the `ISC_USER` and `ISC_PASSWORD` variables are set in the environment, they are picked and used instead of trusted authentication.

> **Note**
>
> Trusted authentication can be coerced to override the environment variables if they are set—refer to the notes below.

Forcing Trusted Authentication

For the situation where trusted authentication is needed and there is a likelihood that `ISC_USER` and `ISC_PASSWORD` are set, there is a new DPB parameter that you can add to the DPB—`isc_dpb_trusted_auth`.

Most of the Firebird command-line utilities support parameter by means of the switch `-tru[sted]` (the abbreviated form is available, according to the usual rules for abbreviating switches).

> **Note**
>
> The *qli* and *nbackup* utilities do not follow the pattern: they use single-letter switches that are somewhat arcane. The switch of interest for *qli* is -K). For *nbackup*, watch this space. The facility to force trusted authentication is yet to be implemented for it.

Example

```
C:\Pr~\bin>isql srv:db            -- log in using trusted authentication
C:\Pr~\bin>set ISC_USER=user1
C:\Pr~\bin>set ISC_PASSWORD=12345
C:\Pr~\bin>isql srv:db            -- log in as 'user1' from environment
C:\Pr~\bin>isql -trust srv:db     -- log in using trusted authentication
```

> **Important**
>
> Windows rules for full domain user names allow names longer than the maximum 31 characters allowed by Firebird for user names. **The 31-character limit is enforced** and, from V.2.1, logins passing longer names are disabled. This will remain the situation until the mapping of OS objects to database objects is implemented in a later Firebird version.

Classic Server on POSIX

For reasons both technical and historical, a Classic server on POSIX with embedded clients is especially vulnerable to security exposure. Users having embedded access to databases MUST be given at least read access to the security database.

This is the main reason that made implementing enhanced password hashes an absolute requirement. A malicious user with user-level access to Firebird could easily steal a copy of the security database, take it home and quietly brute-force the old DES hashes! Afterwards, he could change data in critical databases stored on that server. Firebird 2 is much less vulnerable to this kind of compromise.

But the embedded POSIX server had one more problem with security: its implementation of the Services API calls the command-line *gsec*, as normal users do. Therefore, an embedded user-maintenance utility must have full access to security database.

The main reason to restrict direct access to the security database was to protect it from access by old versions of client software. Fortuitously, it also minimizes the exposure of the embedded Classic on POSIX at the same time, since it is quite unlikely that the combination of an old client and the new server would be present on the production box.

For Any Platform

> **Caution**
>
> The level of Firebird security is still not satisfactory in one serious respect, so please read this section carefully before opening port 3050 to the Internet.

An important security problem with Firebird still remains unresolved: the transmission of poorly encrypted passwords "in clear" across the network. It is not possible to resolve this problem without breaking old clients.

To put it another way, a user who has set his/her password using a new secure method would be unable to use an older client to attach to the server. Taking this into account with plans to upgrade some aspects of the API in the next version, the decision was made not to change the password transmission method in Firebird 2.0.

The immediate problem can be solved easily by using any IP-tunneling software (such as ZeBeDee) to move data to and from a Firebird server, for both 1.5 and 2.0. It remains the recommended way to access your remote Firebird server across the Internet.

Other Security Improvements

isc_service_query() wrongly revealed the full database file spec

Feature request CORE-1091

(V.2.1) When the server is configured "DatabaseAccess = None", `isc_service_query()` would return the full database file path and name. It has been corrected to return the database alias—one more argument in favour of making the use of database aliases standard practice!

Any user could view the server log through the Services API

Feature request CORE-1148

This was a minor security vulnerability. Regular users are now blocked from retrieving the server log using the Services API. Requests are explicitly checked to ensure that the authenticated user is SYSDBA.

Dealing with the New Security Database

A. Peshkov

If you try to put a pre-Firebird 2 security database -- security.fdb or a renamed isc4.gdb -- into Firebird's new home directory and then try to connect to the server, you will get the message "Cannot attach to password

database". It is not a bug: it is by design. A security database from an earlier Firebird version cannot be used directly in Firebird 2.0 or higher.

The newly structured security database is named security2.fdb.

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

> **Note**
>
> A copy of the script appears in Appendix C.

Doing the Security Database Upgrade

To do the upgrade, follow these steps:

1. Put your old security database in some place known to you, but not in Firebird's new home directory. Keep a copy available at all times!
2. Start Firebird 2, using its new, native security2.fdb.
3. Convert your old security database to ODS11 (i.e. backup and restore it using Firebird 2.0). Without this step, running the security_database.sql script will fail!
4. Connect the restored security database as SYSDBA and run the script.
5. Stop the Firebird service.
6. Copy the upgraded database to the Firebird 2 home directory as security2.fdb.
7. Restart Firebird.

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

Nullability of RDB$PASSWD

In pre-2.0 versions of Firebird it was possible to have a user with NULL password. From v.2.0 onward, the RDB $PASSWD field in the security database is constrained as NOT NULL.

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

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

to be

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

Caution with LegacyHash

As long as you configure `LegacyHash = 1` in `firebird.conf`, Firebird's security does not work completely. To set this right, it is necessary to do as follows:

1. Change the SYSDBA password
2. Have the users change their passwords (in 2.0 each user can change his or her own password).
3. Set LegacyHash back to default value of 0, or comment it out.
4. Stop and restart Firebird for the configuration change to take effect.

Chapter 13

Command-line Utilities

General Enhancements

Utilities Support for Database Triggers

(V. 2.1) A new parameter was added to *gbak*, *nbackup* and *isql* to suppress Database Triggers from running. It is available only to the database owner and SYSDBA:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Firebird Services

New Command-line Utility fbsvcmgr

Alex Peshkov

(V.2.1) The new utility *fbsvcmgr* provides a command-line interface to the Services API, enabling access to any service that is implemented in Firebird.

Although there are numerous database administration tools around that surface the Services API through graphical interfaces, the new tool addresses the problem for admins needing to access remote Unix servers in broad networks through a text-only connection. Previously, meeting such a requirement needed a programmer.

Using fbsvcmgr

fbsvcmgr does not emulate the switches implemented in the traditional "g*" utilities. Rather, it is just a front-end through which the Services API functions and parameters can pass. Users therefore need to be familiar with the Services API as it stands currently. The API header file—ibase.h, in the ../include directory of your Firebird installation— should be regarded as the primary source of information about what is available, backed up by the InterBase 6.0 beta API Guide.

Parameters

Specify the Services Manager

The first required parameter for a command line call is the Services Manager you want to connect to:

- For a local connection use the simple symbol `service_mgr`

- To attach to a remote host, use the format `hostname:service_mgr`

Specify subsequent service parameter blocks (SPBs)
Subsequent SPBs, with values if required, follow. Any SPB can be optionally prefixed with a single '-' symbol. For the long command lines that are typical for *fbsvcmgr*, use of the '-' improves the readability of the command line. Compare, for example, the following (each a single command line despite the line breaks printed here):

```
# fbsvcmgr service_mgr user sysdba password masterke
        action_db_stats dbname employee sts_hdr_pages
```

and

```
# fbsvcmgr service_mgr -user sysdba -password masterke
        -action_db_stats -dbname employee -sts_hdr_pages
```

SPB Syntax

The SPB syntax that *fbsvcmgr* understands closely matches with what you would encounter in the `ibase.h` include file or the InterBase 6.0 API documentation, except that a slightly abbreviated form is used to reduce typing and shorten the command lines a little. Here's how it works.

All SPB parameters have one of two forms: (1) `isc_spb_VALUE` or (2) `isc_VALUE1_svc_VALUE2`. For *fbsvcmgr* you just need to pick out the `VALUE, VALUE1 or VALUE2` part[s] when you supply your parameter.

Accordingly, for (1) you would type simply `VALUE`, while for (2) you would type VALUE1_VALUE2. For example:

```
isc_spb_dbname => dbname
isc_action_svc_backup => action_backup
isc_spb_sec_username => sec_username
isc_info_svc_get_env_lock => info_get_env_lock
```

and so on.

> **Note**
>
> An exception is `isc_spb_user_name`: it can be specified as either `user_name` or simply `user`.

It is not realistic to attempt to describe all of the SPB parameters in release notes. In the InterBase 6.0 beta documentation it takes about 40 pages! The next section highlights some known differences between the operation of *fbsvcmgr* and what you might otherwise infer from the old beta documentation.

fbsvcmgr Syntax Specifics

“Do's and Don'ts”
With *fbsvcmgr* you can perform a single action—and get its results if applicable—or you can use it to retrieve multiple information items from the Services Manager. You cannot do both in a single command.

For example,

```
# fbsvcmgr service_mgr -user sysdba -password masterke
    -action_display_user
```

will list all current users on the local firebird server:

```
SYSDBA                          Sql Server Administrator      0    0
QA_USER1                                                      0    0
QA_USER2                                                      0    0
QA_USER3                                                      0    0
QA_USER4                                                      0    0
QA_USER5                                                      0    0
GUEST                                                         0    0
SHUT1                                                         0    0
SHUT2                                                         0    0
QATEST                                                        0    0
```

...and...

```
# fbsvcmgr service_mgr -user sysdba -password masterke
    -info_server_version -info_implementation
```

will report both the server version and its implementation:

```
Server version: LI-T2.1.0.15740 Firebird 2.1 Alpha 1
Server implementation: Firebird/linux AMD64
```

But an attempt to mix all of this in single command line:

```
# fbsvcmgr service_mgr -user sysdba -password masterke
    -action_display_user -info_server_version -info_implementation
```

raises an error:

```
Unknown switch “-info_server_version”
```

Undocumented Items

The function `isc_spb_rpr_list_limbo_trans` was omitted from the IB6 beta documentation. It is supported in *fbsvcmgr*.

Support for New Services API Items in v.2.1

Two new items that were added to the Services API in Firebird 2.1 are supported by *fbsvcmgr*:

- `isc_spb_trusted_auth` (type it as `trusted_auth`) applies only to Windows. It forces Firebird to use Windows trusted authentication.
-

`isc_spb_dbname` gives the ability to set a database name parameter (type as `dbname`) in all service actions related to accessing the security database from a remote client, equivalent to supplying the `-database` switch to the *gsec* utility.

> **Note**
>
> For *gsec* the `-database` switch is mostly used to specify a remote server you want to administer. In *fbsvcmgr*, the name of the server is already given in the first parameter (via the `service_mgr` symbol) so the `[isc_spb_]dbname` parameter is mostly unnecessary.

Documentation Bugs

The format described for some parameters in the InterBase 6 beta documentation are buggy. When in trouble, treat `ibase.h` as the primary source for the correct form.

Unsupported functions

- Everything to do with licensing was removed from the original InterBase 6 open source code and is therefore not supported either in Firebird or by *fbsvcmgr*.
- The old Config file view/modification functions have been unsupported since Firebird 1.5 and are not implemented by *fbsvcmgr*.

Backup Service Misbehaviour Fixed

A. Peshkov

Feature request CORE-1232

(V.2.1) Some misbehaviours that could occur when the Services Manager was doing backup/restore operations and some parameter items were missing or in the wrong sequence. The problem still affects lower versions, including v.2.0.x, so care should be taken to specify all required switches and supply the database name and backup file spec in the correct order when using the **-se[vice_mgr]** switch.

Disable Non-SYSDBA Access to Privileged Services

A. Peshkov

Feature request CORE-787

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

Backup Tools

Firebird 2 brings plenty of enhancements to backing up databases: a new utility for running on-line incremental backups and some improvements to *gbak* to avoid some of the traps that sometimes befall end-users.

New On-line Incremental Backup

N. Samofatov

Fast, on-line, page-level incremental backup facilities have been implemented.

The backup engine comprises two parts:

- NBak, the engine support module
- NBackup, the tool that does the actual backups

Nbak

The functional responsibilities of NBAK are:

1. to redirect writes to difference files when asked (`ALTER DATABASE BEGIN BACKUP` statement)
2. to produce a GUID for the database snapshot and write it into the database header before the `ALTER DATABASE BEGIN BACKUP` statement returns
3. to merge differences into the database when asked (`ALTER DATABASE END BACKUP` statement)
4. to mark pages written by the engine with the current SCN [page scan] counter value for the database
5. to increment SCN on each change of backup state

The backup state cycle is:

nbak_state_normal -> nbak_state_stalled -> nbak_state_merge -> nbak_state_normal

- In *normal* state writes go directly to the main database files.
- In *stalled* state writes go to the difference file only and the main files are read-only.
- In *merge* state new pages are not allocated from difference files. Writes go to the main database files. Reads of mapped pages compare both page versions and return the version which is fresher, because we don't know if it is merged or not.

> **Note**
>
> This merge state logic has one quirky part. Both Microsoft and Linux define the contents of file growth as “undefined” i.e., garbage, and both zero-initialize them.
>
> This is why we don't read mapped pages beyond the original end of the main database file and keep them current in difference file until the end of a merge. This is almost half of NBak fetch and write logic, tested by using modified PIO on existing files containing garbage.

NBackup

The functional responsibilities of NBackup are

1. to provide a convenient way to issue ALTER DATABASE BEGIN/END BACKUP
2. to fix up the database after filesystem copy (physically change `nbak_state_diff` to `nbak_state_normal` in the database header)
3. to create and restore incremental backups.

Incremental backups are multi-level. That means if you do a Level 2 backup every day and a Level 3 backup every hour, each Level 3 backup contains all pages changed from the beginning of the day till the hour when the Level 3 backup is made.

Backing Up

Creating incremental backups has the following algorithm:

1. Issue `ALTER DATABASE BEGIN BACKUP` to redirect writes to the difference file
2. Look up the SCN and GUID of the most recent backup at the previous level
3. Stream database pages having SCN larger than was found at step 2 to the backup file.
4. Write the GUID of the previous-level backup to the header, to enable the consistency of the backup chain to be checked during restore.
5. Issue `ALTER DATABASE END BACKUP`
6. Add a record of this backup operation to `RDB$BACKUP_HISTORY`. Record current level, SCN, snapshot GUID and some miscellaneous stuff for user consumption.

Restoring

Restore is simple: we reconstruct the physical database image for the chain of backup files, checking that the backup_guid of each file matches prev_guid of the next one, then fix it up (change its state in header to nbak_state_normal).

Usage

```
nbackup <options>
```

Valid Options

```
-L <database>   Lock database for filesystem copy
-N <database>       Unlock previously locked database
-F <database>               Fixup database after filesystem copy
-B <level> <database> [<filename>]  Create incremental backup
-R <database> [<file0> [<file1>...]]   Restore incremental backup
-U <user>             User name
-P <password>         Password
```

Note

1. <database> may specify a database alias
2. incremental backup of multi-file databases is not supported yet
3. "stdout" may be used as a value of <filename> for the -B option

User Manual

P. Vinkenoog

A user manual for NBak/NBackup has been prepared. It can be downloaded from the documentation area at the Firebird website: www.firebirdsql.org/pdfmanual/ - the file name is `Firebird-nbackup.pdf`.

gbak Backup/Porting/Restore Utility

A number of enhancements have been added to *gbak*.

Changed Behaviours, New Switches

V. Khorsun

The new gbak switch

```
-RECREATE_DATABASE [OVERWRITE]
```

is a separate switch designed to make harder for the unsuspecting to overwrite a database accidentally, as could occur easily with the shortened form of the old switch:

```
-R[EPLACE_DATABASE]
```

In summary:

- gbak -R (or gbak -r) now applies to the new -R[ECREATE_DATABASE] switch and will never overwrite an existing database if the O[VERWRITE] argument is absent
- The short form of the old gbak -R[EPLACE_DATABASE] is now -REP[LACE_DATABASE]. This switch does not accept the O[VERWRITE] argument.
- The -REP[LACE_DATABASE] switch should be considered as deprecated, i.e. it will become unavailable in some future Firebird release.

This change means that, if you have any legacy batch or cron scripts that rely on “gbak -r” or “gbak -R” without modification, then the operation will except if the database exists.

If you want to retain the ability of your script to overwrite your database unconditionally, you will need to modify the command to use either the new switch with the OVERWRITE argument or the new short form for the old -REPLACE_DATABASE switch.

gbak Made More Version-friendly

C. Valderrama

(V.2.1) In its latest evolution, *gbak* can be used to restore a database on any version of Firebird.

Hide User Name & Password in Shell

A. Peshkov

Feature request CORE-867

(V.2.1) *gbak* now changes param0 to prevent the user name and password from being displayed in `ps axf`.

gbak -V and the “Counter” Parameter

During Firebird 1 development, an optional numeric <*counter*> argument was added to the -V[erbose] switch of gbak for both backup and restore. It was intended to allow you to specify a number and get a running count of rows processed as the row counter passed each interval of that number of rows. It caused undesirable side-effects and was removed before Firebird 1.0 was ever released. So, although it never happened, it was documented as “implemented” in the release notes and other places.

ISQL Query Utility

Work on ISQL has involved a lot of bug-fixing and the introduction of a few new, useful features.

One trick to note is that CHAR and VARCHAR types defined in character set OCTETS (alias BINARY) now display in hex format. Currently, this feature cannot be toggled off.

New Switches

The following command-line switches were added:

-b[ail] "Bail out"

D. Ivanov, C. Valderrama

Command line switch -b to instruct isql to bail out on error when used in non-interactive mode, returning an error code to the operating system.

When using scripts as input in the command line, it may be totally unappropriate to let isql continue executing a batch of commands after an error has happened. Therefore, the "-b[ail]" option will cause script execution to stop at the first error it detects. No further statements in the input script will be executed and isql will return an error code to the operating system.

- Most cases have been covered, but if you find some error that is not recognized by isql, you should inform the project, as this is a feature in progress.
- Currently there is no differentiation by error code---any non-zero return code should be interpreted as failure. Depending on other options (like -o, -m and -m2) , isql will show the error message on screen or will send it to a file.

Some Features

- Even if isql is executing nested scripts, it will cease all execution and will return to the operating system when it detects an error. Nested scripts happen when a script A is used as isql input but in turn A contains

an INPUT command to load script B an so on. Isql doesn't check for direct or indirect recursion, thus if the programmer makes a mistake and script A loads itself or loads script B that in turn loads script A again, isql will run until it exhaust memory or an error is returned from the database, at whose point -bail if activated will stop all activity.

- DML errors will be caught when being prepared or executed, depending on the type of error.
- In many cases, isql will return the line number of a DML statement that fails during execution of a script. (More about error line numbers ...)
- DDL errors will be caught when being prepared or executed by default, since isql uses AUTODDL ON by default. However, if AUTO DLL is OFF, the server only complains when the script does an explicit COMMIT and this may involve several SQL statements.
- The feature can be enabled/disabled interactively or from a script by means of the command

```
SET BAIL [ON | OFF]
```

  As is the case with other SET commands, simply using SET BAIL will toggle the state between activated and deactivated. Using SET will display the state of the switch among many others.

- Even if BAIL is activated, it doesn't mean it will change isql behavior. An additional requirement should be met: the session should be non-interactive. A non-interactive session happens when the user calls isql in batch mode, giving it a script as input.

  Example

```
isql -b -i my_fb.sql -o results.log -m -m2
```

> **Tip**
>
> However, if the user loads isql interactively and later executes a script with the input command, this is considered an interactive session even though isql knows it is executing a script.

  Example

```
isql
Use CONNECT or CREATE DATABASE to specify a database
SQL> set bail;
SQL> input my_fb.sql;
SQL> ^Z
```

  Whatever contents the script has, it will be executed completely, errors and all, even if the BAIL option is enabled.

-m2 to Output Stats and Plans

C. Valderrama

This is a command-line option -M2 to send the statistics and plans to the same output file as the other output (via the -o[utput] switch).

When the user specifies that the output should be sent to a file, two possibilities have existed for years: either

- at the command line, the switch -o followed by a file name is used
- the command OUTput followed by a file name is used, either in a batch session or in the interactive isql shell. (In either case, simply passing the command OUTput is enough to have the output returned to the console). However, although error messages are shown in the console, they are not output to the file.

The -m command line switch was added, to meld (mix) the error messages with the normal output to wherever the output was being redirected.

This left still another case: statistics about operations (SET STATs command) and SQL plans as the server returns them. SET PLAN and SET PLANONLY commands have been treated as diagnostic messages and, as such, were always sent to the console.

What the -m2 command line switch does is to ensure that stats and plans information go to the same file the output has been redirected to.

> **Note**
>
> Neither -m nor -m2 has an interactive counterpart through a SET command. They are for use only as command-line isql options.

-r2 to Pass a Case-Sensitive Role Name

C. Valderrama

The sole objective of this parameter is to specify a case-sensitive role name.

- The default switch for this parameter is -r. Roles provided in the command line are uppercased
- With -r2, the role is passed to the engine exactly as typed in the command line.

New Commands and Enhancements

The following commands have been added or enhanced.

Ctrl-C to cancel query output

M. Kubecek
A. dos Santos Fernandes

Feature request CORE-704

(V. 2.1) Output from a SELECT in an interactive isql session can now be stopped using Ctrl-C. Note, this merely stops fetching rows from the buffer, it does not cancel the query.

Extension of isql SHOW SYSTEM command

A. dos Santos Fernandes

Feature request CORE-978

(V. 2.1) The SHOW <object_type> command is meant to show user objects of that type. The SHOW SYSTEM commmand is meant to show system objects but, until now, it only showed system tables. Now it lists the predefined system UDFs incorporated into FB 2.

It may be enhanced to list system views if we create some of them in the future.

SHOW COLLATIONS command

A. dos Santos Fernandes

(V. 2.1) Lists all the character set/collation pairs declared in the database.

SET HEAD[ing] toggle

C. Valderrama

Some people consider it useful to be able to do a SELECT inside isql and have the output sent to a file, for additional processing later, especially if the number of columns makes isql display impracticable. However, isql by default prints column headers and. in this scenario, they are a nuisance.

Therefore, printing the column headers -- previously a fixed feature -- can now be enabled/disabled interactively or from a script by means of the

```
SET HEADing [ON | OFF]
```

command in the isql shell. As is the case with other SET commands, simply using SET HEAD will toggle the state between activated and deactivated.

> **Note**
>
> There is no command line option to toggle headings off.

Using SET will display the state of SET HEAD, along with other switches that can be toggled on/off in the isql shell.

SET SQLDA_DISPLAY ON/OFF

A. dos Santos Fernandes

This SQLDA_DISPLAY command shows the input SQLDA parameters of INSERTs, UPDATEs and DELETEs. It was previously available only in DEBUG builds and has now been promoted to the public builds. It shows the information for raw SQLVARs. Each SQLVAR represents a field in the XSQLDA, the main structure used in the FB API to talk to clients transferring data into and out of the server.

> **Note**
>
> The state of this option is not included in the output when you type `SET;` in isql to see the current settings of most options.

SET TRANSACTION Enhanced

C. Valderrama

The SET TRANSACTION statement has been enhanced so that, now, all TPB options are supported:

- NO AUTO UNDO
- IGNORE LIMBO
- LOCK TIMEOUT <number>

Example

```
SET TRANSACTION WAIT SNAPSHOT NO AUTO UNDO LOCK TIMEOUT 10
```

See also the document *doc/sql.extensions/README.set_transaction.txt*.

SHOW DATABASE now Returns ODS Version Number

C. Valderrama

ODS (On-Disk Structure) version is now returned in the SHOW DATABASE command (C. Valderrama)

Ability to show the line number where an error happened in a script

C. Valderrama

In previous versions, the only reasonable way to know where a script had caused an error was using the switch -e for echoing commands, -o to send the output to a file and -m to merge the error output to the same file. This way, you could observe the commands *isql* executed and the errors if they exist. The script continued executing to the end. The server only gives a line number related to the single command (statement) that it's executing, for some DSQL failures. For other errors, you only know the statement caused problems.

With the addition of -b for bail as described in (1), the user is given the power to tell isql to stop executing scripts when an error happens, but you still need to echo the commands to the output file to discover which statement caused the failure.

Now, the ability to signal the script-related line number of a failure enables the user to go to the script directly and find the offending statement. When the server provides line and column information, you will be told the exact line of DML in the script that caused the problem. When the server only indicates a failure, you will be told the starting line of the statement that caused the failure, related to the whole script.

This feature works even if there are nested scripts, namely, if script SA includes script SB and SB causes a failure, the line number is related to SB. When SB is read completely, isql continues executing SA and then isql continues counting lines related to SA, since each file gets a separate line counter. A script SA includes SB when SA uses the INPUT command to load SB.

Lines are counted according to what the underlying IO layer considers separate lines. For ports using EDITLINE, a line is what readline() provides in a single call. The line length limit of 32767 bytes remains unchanged.

Enhanced Command-line Help

M. Kubecek

When unknown parameters are used, isql now shows all of the command-line parameters and their explanations instead of just a simple list of allowed switches.

```
opt/firebird/bin] isql -?
```

```
Unknown switch: ?
usage:    isql [options] [<database>]
        -a(all)                 extract metadata incl. legacy non-SQL tables
        -b(ail)                 bail on errors (set bail on)
        -c(ache) <num>          number of cache buffers
        -ch(arset) <charset>    connection charset (set names)
        -d(atabase) <database>  database name to put in script creation
        -e(cho)                 echo commands (set echo on)
        -ex(tract)              extract metadata
        -i(nput) <file>         input file (set input)
        -m(erge)                merge standard error
        -m2                     merge diagnostic
        -n(oautocommit)         no autocommit DDL (set autoddl off)
        -now(arnings)           do not show warnings
        -o(utput) <file>        output file (set output)
        -pag(elength) <size>    page length
        -p(assword) <password>  connection password
        -q(uiet)                do not show the message "Use CONNECT..."
        -r(ole) <role>          role name
        -r2 <role>              role (uses quoted identifier)
        -sqldialect <dialect>   SQL dialect (set sql dialect)
        -t(erminator) <term>    command terminator (set term)
        -u(ser) <user>          user name
        -x                      extract metadata
        -z                      show program and server version
```

gsec Authentication Manager

Changes to the *gsec* utility include:

gsec return code

C. Valderrama

gsec now returns an error code when used as a non-interactive utility. Zero indicates success; any other code indicates failure.

gfix Server Utility

Changes to the gfix utility include:

New Shutdown States (Modes)

N. Samofatov, D. Yemanov

The options for `gfix -shut[down]` have been extended to include two extra states or modes to govern the shutdown.

New Syntax Pattern

```
gfix <command> [<state>] [<options>]
```

```
<command> ::= {-shut | -online}
<state> ::= {normal | multi | single | full}
<options> ::= {-force <timeout> | -tran | -attach}
```

- “normal” state = online database

- “multi” state = multi-user shutdown mode (the legacy one, unlimited attachments of SYSDBA/owner are allowed)

- “single” state = single-user shutdown (only one attachment is allowed, used by the restore process)

- “full” state = full/exclusive shutdown (no attachments are allowed)

> **Note**
>
> “Multi” is the default state for -shut, “normal” is the default state for -online.

The modes can be switched sequentially:

normal <-> multi <-> single <-> full

Examples

```
gfix -shut single -force 0
gfix -shut full -force 0
gfix -online single
gfix -online
```

You cannot use `-shut` to bring a database one level “more online” and you cannot use `-online` to make a database more protected (an error will be thrown).

These are prohibited:

```
gfix -shut single -force 0
gfix -shut multi -force 0

gfix -online
gfix -online full

gfix -shut -force 0
gfix -online single
```

Builds and Installs

Parameter for Instance name added to instsvc.exe

D. Yemanov

Feature request CORE-673

(V.2.1) `instsvc.exe` now supports multi-instance installations.

Revised Win32 Installer Docs

P. Reeves

(V.2.1) The documentation for command-line setup on Windows has been revised. See `doc/install_windows_manually.txt`.

Help on command line switches

Feature request CORE-548

(V.2.1) On-line help is now available on the switches for command-line setup on Windows.

Gentoo/FreeBSD detection during install

A. Peshkov

Feature request CORE-1047

Gentoo or FreeBSD is now detected during configuration, making it more likely that the binary install will work “out of the box” on these platforms.

Chapter 14

External Functions (UDFs)

Ability to Signal SQL NULL via a Null Pointer

C. Valderrama

Previous to Firebird 2, UDF authors only could guess that their UDFs might return a null, but they had no way to ascertain it. This led to several problems with UDFs. It would often be assumed that a null string would be passed as an empty string, a null numeric would be equivalent to zero and a null date would mean the base date used by the engine.

For a numeric value, the author could not always assume null if the UDF was compiled for an environment where it was known that null was not normally recognized.

Several UDFs, including the ib_udf library distributed with Firebird, assumed that an empty string was more likely to signal a null parameter than a string of length zero. The trick may work with CHAR type, since the minimum declared CHAR length is one and would contain a blank character normally: hence, binary zero in the first position would have the effect of signalling NULL.

However, but it is not applicable to VARCHAR or CSTRING, where a length of zero is valid.

The other solution was to rely on raw descriptors, but this imposes a lot more things to check than they would want to tackle. The biggest problem is that the engine won't obey the declared type for a parameter; it will simply send whatever data it has for that parameter, so the UDF is left to decide whether to reject the result or to try to convert the parameter to the expected data type.

Since UDFs have no formal mechanism to signal errors, the returned value would have to be used as an indicator.

The basic problem was to keep the simplicity of the typical declarations (no descriptors) while at the same time being able to signal null.

The engine normally passed UDF parameters by reference. In practical terms, that means passing a pointer to the data to tell the UDF that we have SQL NULL. However, we could not impose the risk of crashing an unknown number of different, existing public and private UDFs that do not expect NULL. The syntax had to be enhanced to enable NULL handling to be requested explicitly.

The solution, therefore, is to restrict a request for SQL NULL signaling to UDFs that are known to be capable of dealing with the new scenario. To avoid adding more keywords, the NULL keyword is appended to the UDF parameter type and no other change is required.

Example

```
declare external function sample
  int null
  returns int by value...;
```

If you are already using functions from `ib_udf` and want to take advantage of null signaling (and null recognition) in some functions, you should connect to your desired database, run the script `../misc/upgrade/ib_udf_upgrade.sql` that is in the Firebird directory, and commit afterwards.

> **Caution**
>
> It is recommended to do this when no other users are connected to the database.

The code in the listed functions in that script has been modified to recognize null only when NULL is signaled by the engine. Therefore, starting with FB v2, `rtrim()`, `ltrim()` and several other string functions no longer assume that an empty string means a NULL string.

The functions won't crash if you don't upgrade: they will simply be unable to detect NULL.

If you have never used ib_udf in your database and want to do so, you should connect to the database, run the script `../udf/ib_udf2.sql`, preferably when no other users are connected, and commit afterwards.

> **Note**
>
> - Note the "2" at the end of the name.
> - The original script for FB v1.5 is still available in the same directory.

UDF library diagnostic messages improved

A. Peshkov

Diagnostics regarding a missing/unusable UDF module have previously made it hard to tell whether a module was missing or access to it was being denied due to the UDFAccess setting in firebird.conf. Now we have separate, understandable messages for each case.

UDFs Added and Changed

UDFs added or enhanced in Firebird 2.0's supplied libraries are:

IB_UDF_rand() vs IB_UDF_srand()

F. Schlottmann-Goedde

In previous versions, the external function rand() sets the random number generator's starting point based on the current time and then generates the pseudo-random value.

```
srand((unsigned) time(NULL));
return ((float) rand() / (float) RAND_MAX);
```

The problem with this algorithm is that it will return the same value for two calls done within a second.

To work around this issue, rand() was changed in Firebird 2.0 so that the starting point is not set explicitly. This ensures that different values will always be returned.

In order to keep the legacy behaviour available in case somebody needs it, *srand()* has been introduced. It does exactly the same as the old rand() did.

IB_UDF_lower

The function `IB_UDF_lower()` in the IB_UDF library might conflict with the new internal function `lower()`, if you try to declare it in a database using the ib_udf.sql script from a previous Firebird version.

```
/* ib_udf.sql declaration that now causes conflict */
DECLARE EXTERNAL FUNCTION lower
    CSTRING(255)
    RETURNS CSTRING(255) FREE_IT
    ENTRY_POINT 'IB_UDF_lower' MODULE_NAME 'ib_udf';
```

The problem will be resolved in the latest version of the new ib_udf2.sql script, where the old UDF is declared using a quoted identifier.

```
/* New declaration in ib_udf2.sql */
DECLARE EXTERNAL FUNCTION "LOWER"
  CSTRING(255) NULL
  RETURNS CSTRING(255) FREE_IT
  ENTRY_POINT 'IB_UDF_lower' MODULE_NAME 'ib_udf';
```

> **Tip**
>
> It is preferable to use the internal function LOWER() than to call the UDF.

General UDF Changes

Build Changes

C. Valderrama Contributors

The FBUDF library no longer depends on `[lib]fbclient` to be built.

Chapter 15

New Configuration Parameters and Changes

Authentication

A. Peshkov

(V.2.1) On Windows server platforms, from V.2.1 forward, *Authentication* is used for configuring the server authentication mode if you need it to be other than the default *mixed*.

- *trusted* makes use of Windows “trusted authentication” which, under the right conditions, 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.

RelaxedAliasChecking

V. Khorsun

(V.2.1) *RelaxedAliasChecking* is a new configuration parameter added to permit a slight relaxation of the Firebird 2.0.x restrictions on mixing relation aliases and table names in a query. For example, with RelaxedAliasChecking set to true (=1) in firebird.conf, the following query will succeed in Firebird 2.1, whereas it would fail in v.2.0.x, or in v.2.1 with the parameter set to its default of 0:

```
SELECT ATABLE.FIELD1, B.FIELD2
  FROM ATABLE A JOIN BTABLE B
  ON A.ID = BTABLE.ID
```

Caution

Understand that this is a *temporary facility* whose purpose is to provide some headspace for migrating systems using legacy code that exploited the tolerance of InterBase and older Firebird server versions to non-standard SQL usage.

- Don't enable this parameter if you have no “offending” code in your applications or PSQL modules. It is not intended as an invitation to write sloppy code!
- Regard it as a time-bomb. It will be permanently removed from a future release.

MaxFileSystemCache

V. Khorsun

(V.2.1) Sets a threshold determining whether Firebird will allow the page cache to be duplicated to the filesystem cache or not. If this parameter is set to any (integer) value greater than zero, its effect depends on the current default size of the page cache: if the default page cache (in pages) is less than the value of MaxFileSystemCache (in pages) then filesystem caching is enabled, otherwise it is disabled.

> **Note**
>
> This applies both when the page cache buffer size is set implicitly by the DefaultDBCachePages setting or explicitly as a database header attribute.

Thus,

- To disable filesystem caching always, set MaxFileSystemCache to zero
- To enable filesystem caching always, set MaxFileSystemCache an integer value that is sufficiently large to exceed the size of the database page cache. Remember that the effect of this value will be affected by subsequent changes to the page cache size.

> **Important**
>
> The default setting for MaxFileSystemCache is 65536 pages, i.e. filesystem caching is enabled.

DatabaseGrowthIncrement

V. Khorsun

(V.2.1) For better control of disk space preallocation, the new parameter *DatabaseGrowthIncrement* has been added to `firebird.conf`. It represents the upper limit for the size, *in bytes*, of the chunk of disk that will be requested for preallocation as pages for writes from the cache. Default: 134,217,728 bytes (128 MB).

For background information, please refer to the topic Enlarge Disk Allocation Chunks in the chapter “Global Improvements in Firebird 2.1”.

When the engine needs to initialize more disk space, it allocates a block that is 1/16th of the space already allocated, but not less than 128 KB and not greater than the DatabaseGrowthIncrement value. The DatabaseGrowthIncrement value can be raised to increase the maximum size of newly-allocated blocks to more than the default 128 MB. Set it to zero to disable preallocation.

> **Note**
>
> - The lower limit of the block size is purposely hard-coded at 128 KB and cannot be reconfigured.
> - Space is not preallocated for database shadow files.
> - Preallocation is disabled for a database that has the “No reserve” option set.

ExternalFileAccess

A. Peshkov

Modified in Firebird 2, to allow the first path cited in ExternalFilesAccess to be used as the default when a new external file is created.

LegacyHash

A. Peshkov

This parameter enables you to configure Firebird 2 to reject an old DES hash always in an upgraded security database. If you don't use the security database upgrade procedure, this parameter does not affect Firebird operation. A DES hash cannot arrive in the new security2.fdb.

Refer to the Security DB Upgrade Security section for instructions on upgrading your existing Firebird 1.5 security.fdb (or a renamed isc4.gdb) to the new security database layout.

The default value is 1 (true).

Redirection

A. Peshkov

Parameter for controlling redirection of remote requests. It controls the multi-hop capability that was broken in InterBase 6 and is restored in Firebird 2.

About Multi-hop

When you attach to some database using multiple hosts in the connection string, only the last host in this list is the one that opens the database. The other hosts act as intermediate gateways on port gds_db. Previously, when working, this feature was available unconditionally. Now, it can be configured.

Remote redirection is turned *off* by default.

> **Caution**
>
> If you are considering enabling multi-hop capability, please study the Warning text in the chapter on Security and in the documentation for this parameter in the firebird.conf file.

GCPolicy

V. Khorsun

Garbage collection policy. It is now possible to choose the policy for garbage collection on SuperServer. The possible settings are cooperative, background and combined, as explained in the notes for `GPolicy` in `firebird.conf`.

Not applicable to Classic, which supports only cooperative garbage collection.

OldColumnNaming

P. Reeves

The parameter OldColumnNaming has been ported forward from Firebird 1.5.3. This parameter allows users to revert to pre-V1.5 column naming behaviour in SELECT expressions. The installation default is 0 (disabled). If it is enabled, the engine will not attempt to supply run-time identifiers, e.g. CONCATENATION for derived fields where the developer has neglected to provide identifiers.

> **Important**
>
> This setting affects all databases on the server and will potentially produce exceptions or unpredicted results where mixed applications are implemented.

UsePriorityScheduler

A. Peshkov

Setting this parameter to zero now disables switching of thread priorities completely. It affects only the Win32 SuperServer.

TCPNoNagle has changed

K. Kuznetzov

The default value for TcpNoNagle is now TCP_NODELAY.

Removed or Deprecated Parameters

CreateInternalWindow

D. Yemanov

The option CreateInternalWindow is no longer required to run multiple server instances and it has been removed.

DeadThreadsCollection is no longer used

A. Peshkov

The DeadThreadsCollection parameter is no longer used at all. Dead threads are now efficiently released "on the fly", making configuration unnecessary. Firebird 2.0 silently ignores this parameter.

Chapter 16

Firebird 2 Project Teams

Table 16.1. Firebird Development Teams

Developer	Country	Major Tasks
Dmitry Yemanov	Russian Federation	Full-time database engineer/implementor, core team leader
Alex Peshkov	Russian Federation	Security features coordinator; buildmaster; porting authority
Claudio Valderrama	Chile	Code scrutineer; bug-finder and fixer; ISQL enhancements; UDF fixer, designer and implementor
Vladislav Khorsun	Ukraine	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/Canada	Designed and implemented new inline NBackup; code-fixer and DB engineer during part of V.2.0 development
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
Sean Leyne	Canada	Bugtracker organizer
Dimitrios Ioannides	Greece	New Jira bugtracker implementor
Ann Harrison	U.S.A.	Frequent technical advisor
Jim Starkey	U.S.A.	Frequent architectural advisor; occasional bug-fixer
Roman Rokytskyy	Germany	Jaybird implementor and co-coordinator
Ryan Baldwin	U.K.	Jaybird Type 2 driver developer
Evgeny Putilin	Russian Federation	Java stored procedures implementation
Carlos Guzman Alvarez	Spain	

Developer	Country	Major Tasks
		Developer and coordinator of .NET providers for Firebird until 2007
Jiri Cincura	Czech Republic	Developer and coordinator of .NET providers from January 2008
Vladimir Tsvigun	Ukraine	Developer and coordinator of ODBC/JDBC driver for Firebird
David Rushby (d.)	U.S.A.	Developer and coordinator of KInterbase Python interface for Firebird until his accidental death in July, 2007
Konstantin Kuznetsov	Russian Federation	Solaris Intel builds
Paul Vinkenoog	The Netherlands	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
Olivier Mascia	Belgium	IBPP C++ interface developer; re-implementor of Win32 installation services
Oleg Loa	Russian Federation	Contributor
Grzegorz Prokopski	Hungary	Debian builds
Erik Kunze	Germany	SINIX-Z port; raw device enablement
Helen Borrie	Australia	Release notes editor; Chief of Thought Police

Appendix A: New Built-in Functions

(Firebird 2.1)

Function	Format	Description
ABS	ABS(<number>)	Returns the absolute value of a number.
`select abs(amount) from transactions;`		
ACOS	ACOS(<number>)	Returns the arc cosine of a number. Argument to ACOS must be in the range -1 to 1. Returns a value in the range 0 to PI.
`select acos(x) from y;`		
ASCII_CHAR	ASCII_CHAR(<number>)	Returns the ASCII character with the specified code. The argument to ASCII_CHAR must be in the range 0 to 255. The result is returned in character set NONE.
`select ascii_char(x) from y;`		
ASCII_VAL	ASCII_VAL(<string>)	Returns the ASCII code of the first character of the specified string. 1. Returns 0 if the string is empty 2. Throws an error if the first character is multi-byte
`select ascii_val(x) from y;`		
ASIN	ASIN(<number>)	Returns the arc sine of a number. The argument to ASIN must be in the range -1 to 1. It returns a result in the range -PI/2 to PI/2.
`select asin(x) from y;`		
ATAN	ATAN(<number>)	Returns the arc tangent of a number. Returns a value in the range -PI/2 to PI/2.
`select atan(x) from y;`		
ATAN2	ATAN2(<number>, <number>)	Returns the arc tangent of the first number / the second number. Returns a value in the range -PI to PI.

Function	Format	Description
`select atan2(x, y) from z;`		
BIN_AND	BIN_AND(<number> [, <number> ...])	Returns the result of a binary AND operation performed on all arguments.
`select bin_and(flags, 1) from x;`		
BIN_OR	BIN_OR(<number> [, <number> ...])	Returns the result of a binary OR operation performed on all arguments.
`select bin_or(flags1, flags2) from x;`		
BIN_SHL	BIN_SHL(<number>, <number>)	Returns the result of a binary shift left operation performed on the arguments (first << second).
`select bin_shl(flags1, 1) from x;`		
BIN_SHR	BIN_SHR(<number>, <number>)	Returns the result of a binary shift right operation performed on the arguments (first >> second).
`select bin_shr(flags1, 1) from x;`		
BIN_XOR	BIN_XOR(<number> [, <number> ...])	Returns the result of a binary XOR operation performed on all arguments.
`select bin_xor(flags1, flags2) from x;`		
BIT_LENGTH	BIT_LENGTH(<string> | <string_expr>)	Returns the length of a string in bits.

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

CEIL | CEILING	{ CEIL | CEILING }(<number>)	Returns a value representing the smallest integer that is greater than or equal to the input argument.
`1) select ceil(val) from x;` `2) select ceil(2.1), ceil(-2.1) from rdb$database;  -- returns 3, -2`		

Function	Format	Description
CHAR_LENGTH | *CHARACTER_LENGTH*	CHAR_LENGTH(<string> | <string_expr>)	Returns the number of characters in a string or expression result.

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

COS	COS(<number>)	Returns the cosine of a number. The angle is specified in radians and returns a value in the range -1 to 1.

```
select cos(x) from y;
```

COSH	COSH(<number>)	Returns the hyperbolic cosine of a number.

```
select cosh(x) from y;
```

COT	COT(<number>)	Returns 1 / tan(argument).

```
select cot(x) from y;
```

DATEADD	See below	Returns a date/time/timestamp value increased (or decreased, when negative) by the specified amount of time.

Format:

```
DATEADD( <number> <timestamp_part> TO <date_time> )
DATEADD( <timestamp_part>, <number>, <date_time> )
timestamp_part ::= { YEAR | MONTH | DAY | HOUR |
                     MINUTE | SECOND | MILLISECOND }
```

1. YEAR, MONTH and DAY cannot be used with time values.
2. HOUR, MINUTE, SECOND and MILLISECOND cannot be used with date values.
3. All timestamp_part values can be used with timestamp values.

Example

```
select dateadd(day, -1, current_date) as yesterday
  from rdb$database;

  /* or (expanded syntax) */
```

Function	Format	Description
`select dateadd(-1 day to current_date) as yesterday` `  from rdb$database;`		
DATEDIFF	See below	Returns an exact numeric value representing the interval of time from the first date/time/timestamp value to the second one.

Format:

```
DATEDIFF( <timestamp_part> FROM <date_time> TO <date_time> )
DATEDIFF( <timestamp_part>, <date_time>, <date_time> )
timestamp_part ::= { YEAR | MONTH | DAY |
                     HOUR | MINUTE | SECOND | MILLISECOND }
```

1. Returns a positive value if the second value is greater than the first one, negative when the first one is greater, or zero when they are equal.
2. Comparison of date with time values is invalid.
3. YEAR, MONTH, and DAY cannot be used with time values.
4. HOUR, MINUTE, SECOND and MILLISECOND cannot be used with date values.
5. All timestamp_part values can be used with timestamp values.

Example

```
select datediff(
  DAY, (cast('TOMORROW' as date) -10), current_date)
    as datediffresult
from rdb$database;
```

DECODE	See below	DECODE is a shortcut for a CASE ... WHEN ... ELSE expression.

Format:

```
DECODE( <expression>, <search>, <result>
  [ , <search>, <result> ... ] [, <default> ]
```

Example

```
  select decode(state, 0, 'deleted', 1, 'active', 'unknown') from things;
```

EXP	EXP(<number>)	Returns the exponential e to the argument.

```
  select exp(x) from y;
```

Function	Format	Description
FLOOR	FLOOR(<number>)	Returns a value representing the largest integer that is less than or equal to the input argument.

```
1) select floor(val) from x;
2) select floor(2.1), floor(-2.1)
      from rdb$database;  -- returns 2, -3
```

GEN_UUID	GEN_UUID() -- no arguments	Returns a universal unique number.

```
insert into records (id) value (gen_uuid());
```

HASH	HASH(<string>)	Returns a HASH of a string.

```
select hash(x) from y;
```

LEFT	LEFT(<string>, <number>)	Returns the substring of a specified length that appears at the start of a left-to-right string.

```
select left(name, char_length(name) - 10)
      from people
      where name like '% FERNANDES';
```

1. The first position in a string is 1, not 0.
2. If the <number> argument evaluates to a non-integer, banker's rounding is applied.

LN	LN(<number>)	Returns the natural logarithm of a number.

```
select ln(x) from y;
```

LOG	LOG(<number>, <number>)	LOG(x, y) returns the logarithm base x of y.

```
select log(x, 10) from y;
```

LOG10	LOG10(<number>)	Returns the logarithm base ten of a number.

```
select log10(x) from y;
```

LOWER	LOWER(<string>)	**(v.2.0.x)** Returns the input argument converted to all lower-case characters.

```
isql -q -ch dos850

SQL> create database 'test.fdb';
```

Function	Format	Description
SQL> create table t (c char(1) character set dos850); SQL> insert into t values ('A'); SQL> insert into t values ('E'); SQL> insert into t values ('Á');; SQL> insert into t values ('É'); SQL> select c, lower(c) from t; C LOWER ====== ====== A a E e Á á É é		
LPAD	LPAD(<string>, <number> [, <string>])	LPAD(string1, length, string2) prepends string2 to the beginning of string1 until the length of the result string becomes equal to length.
1. If the second string is omitted the default value is one space. 2. If the result string would exceed the length, the second string is truncated. Example select lpad(x, 10) from y;		
MAXVALUE	MAXVALUE(<value> [, <value> ...])	Returns the maximum value of a list of values.
select maxvalue(v1, v2, 10) from x;		
MINVALUE	MINVALUE(<value> [, <value> ...)	Returns the minimun value of a list of values.
select minvalue(v1, v2, 10) from x;		
MOD	MOD(<number>, <number>)	Modulo: MOD(X, Y) returns the remainder part of the division of X by Y.
select mod(x, 10) from y;		
OCTET_LENGTH	OCTET_LENGTH(<string> | <string_expr>)	Returns the length of a string or expression result in bytes.
select rdb$relation_name, octet_length(rdb$relation_name), octet_length(trim(rdb$relation_name)) from rdb$relations;		

Function	Format	Description
OVERLAY	See below	Returns string1 replacing the substring FROM start FOR length by string2.

Format:

```
OVERLAY( <string1> PLACING <string2> FROM <start> [ FOR <length> ] )
```

The OVERLAY function is equivalent to:

```
SUBSTRING(<string1>, 1 FOR <start> - 1) || <string2> ||
SUBSTRING(<string1>, <start> + <length>)
```

1. The first position in a string is 1, not 0.
2. If the <start> and/or <length > argument evaluates to a non-integer, banker's rounding is applied.

If <length> is not specified, CHAR_LENGTH(<string2>) is implied.

PI	PI() -- no arguments	Returns the PI constant (3.1459...).

```
val = PI();
```

POSITION	See below	Returns the start position of the first string inside the second string, relative to the beginning of the outer string. In the second form, an offset position may be supplied so that the function returns a result relative to that position in the outer string.

```
POSITION(  <string> IN <string>  )
POSITION( <string>, <string> [, <offset-position>] )
```

```
select rdb$relation_name
  from rdb$relations
  where position('RDB$' IN rdb$relation_name) = 1;
```

POWER	POWER(<number>, <number>)	POWER(X, Y) returns X to the power of Y.

```
select power(x, 10) from y;
```

RAND	RAND() -- no argument	Returns a random number between 0 and 1.

Function	Format	Description
`select * from x order by rand();`		
REPLACE	REPLACE(<stringtosearch>, <findstring>, <replstring>)	Replaces all occurrences of <findstring> in <stringtosearch> with <replstring>.
`select replace(x, ' ', ',') from y;`		
REVERSE	REVERSE(<value>)	Returns a string in reverse order. Useful function for creating an expression index that indexes strings from right to left.

`create index people_email on people`
`  computed by (reverse(email));`
`select * from people`
`  where reverse(email) starting with reverse('.br');`		

RIGHT	RIGHT(<string>, <number>)	Returns the substring, of the specified length, from the right-hand end of a string.

`select right(rdb$relation_name, char_length(rdb$relation_name) - 4)`
`  from rdb$relations`
`  where rdb$relation_name like 'RDB$%';`		

ROUND	ROUND(<number>, [<number>])	Returns a number rounded to the specified scale.
Example `select round(salary * 1.1, 0) from people;` If the scale (second parameter) is negative or is omitted, the integer part of the value is rounded. E.g., ROUND(123.456, -1) returns 120.000.		
RPAD	RPAD(<string1>, <length> [, <string2>])	Appends <string2> to the end of <string1> until the length of the result string becomes equal to <length>.
Example `select rpad(x, 10) from y;` 1. If the second string is omitted the default value is one space. 2. If the result string would exceed the length, the final application of <string2> will be truncated.		

Function	Format	Description
SIGN	SIGN(<number>)	Returns 1, 0, or -1 depending on whether the input value is positive, zero or negative, respectively.
`select sign(x) from y;`		
SIN	SIN(<number>)	Returns the sine of an input number that is expressed in radians.
`select sin(x) from y;`		
SINH	SINH(<number>)	Returns the hyperbolic sine of a number.
`select sinh(x) from y;`		
SQRT	SQRT(<number>)	Returns the square root of a number.
`select sqrt(x) from y;`		
TAN	TAN(<number>)	Returns the tangent of an input number that is expressed in radians.
`select tan(x) from y;`		
TANH	TANH(<number>)	Returns the hyperbolic tangent of a number.
`select tanh(x) from y;`		
TRIM	See below	**(V.2.0.x)** Trims characters (default: blanks) from the left and/or right of a string.

```
TRIM <left paren> [ [ <trim specification> ] [ <trim character> ]
  FROM ] <value expression> <right paren>

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

   <trim character> ::= <value expression>
```

Rules

1. If <trim specification> is not specified, BOTH is assumed.
2. If <trim character> is not specified, ' ' is assumed.
3. If <trim specification> and/or <trim character> is specified, FROM should be specified.

Function	Format	Description

4. If <trim specification> and <trim character> is not specified, FROM should not be specified.
 	
Example A)
 	
`select`
`  rdb$relation_name,`
`  trim(leading 'RDB$' from rdb$relation_name)`
`from rdb$relations`
`  where rdb$relation_name starting with 'RDB$';`
 	
Example B)
 	
`select`
`  trim(rdb$relation_name) || ' is a system table'`
`from rdb$relations`
`  where rdb$system_flag = 1;`		

TRUNC	TRUNC(<number> [, <number>])	Returns the integral part (up to the specified scale) of a number.

`1) select trunc(x) from y;`
`2) select trunc(-2.8), trunc(2.8)`
`  from rdb$database;  -- returns -2, 2`
`3) select trunc(987.65, 1), trunc(987.65, -1)`
`  from rdb$database;  -- returns 987.60, 980.00`		

Appendix B: International Character Sets

A. dos Santos Fernandes & Others

New Character Sets and Collations Implemented

The following new character set and/or collation implementations have been added in Firebird 2 releases:

Character Set	Collation	Description	Implemented By
ISO8859_1	ES_ES_CI_AI	Spanish language case- and accent-insensitive collation for ISO8859_1 character set.	A. dos Santos Fernandes
"	PT_BR	Brazil Portuguese collation for ISO8859_1 character set.	A. dos Santos Fernandes, P. H. Albanez
ISO8859_2	ISO_PLK	Polish collation for ISO8859_2 character set.	J. Glowacki, A. dos Santos Fernandes
KOI8-R	KOI8-R	Russian language character set and dictionary collation.	O. Loa, A. Karyakin
KOI8-U	KOI8-U	Ukrainian language character set and dictionary collation.	O. Loa, A. Karyakin
WIN1250	BS_BA	Bosnian language collation for WIN1250 character set.	F. Hasovic
"	WIN_CZ_AI	Czech language case-insensitive collation for WIN1250 character set	I. Prenosil, A. dos Santos Fernandes
"	WIN_CZ_CI_AI	Czech language case- and accent-insensitive collation for WIN1250 character set	I. Prenosil, A. dos Santos Fernandes
WIN1252	WIN_PTBR	Brazil Portuguese collation for WIN1252 character set.	A. dos Santos Fernandes, P. H. Albanez
WIN1257	WIN1257_LV	Latvian dictionary collation.	O. Loa, A. Karyakin
"	WIN1257_LT	Lithuanian dictionary collation.	O. Loa, A. Karyakin
"	WIN1257_EE	Estonian dictionary collation.	O. Loa, A. Karyakin
WIN1258	(n/a)	Vietnamese language subset of charset WIN1258.	Nguyen The Phuong, A. dos Santos Fernandes
UTF8	UCS_BASIC	Unicode 4.0 support with UTF8 character set and UCS_BASIC collation.	A. dos Santos Fernandes

Character Set	Collation	Description	Implemented By
"	UNICODE	Unicode 4.0 support with UTF8 character set and UNICODE collation.	A. dos Santos Fernandes
(Unspecified)	FR_FR_CI_AI	(V.2.1) French language case-insensitive and accent-insensitive collation.	A. dos Santos Fernandes
CP943C	(n/a)	(V.2.1) Japanese character set.	A. dos Santos Fernandes

Narrow Character Sets

```
CYRL,
DOS437, DOS737, DOS775, DOS850, DOS852, DOS857, DOS858, DOS860,
DOS861, DOS862, DOS863, DOS864, DOS865, DOS866, DOS869,
ISO8859_1, ISO8859_13, ISO8859_2, ISO8859_3, ISO8859_4,
ISO8859_5, ISO8859_6, ISO8859_7, ISO8859_8, ISO8859_9,
KOI8R, KOI8U,
NEXT,
TIS620,
WIN1250, WIN1251, WIN1252, WIN1253, WIN1254, WIN1255, WIN1256,
WIN1257 and WIN1258.
```

ICU Character Sets

```
UTF-8 ibm-1208 ibm-1209 ibm-5304 ibm-5305 windows-65001 cp1208
UTF-16 ISO-10646-UCS-2 unicode csUnicode ucs-2
UTF-16BE x-utf-16be ibm-1200 ibm-1201 ibm-5297 ibm-13488
  ibm-17584 windows-1201 cp1200 cp1201 UTF16_BigEndian
UTF-16LE x-utf-16le ibm-1202 ibm-13490 ibm-17586
  UTF16_LittleEndian windows-1200
UTF-32 ISO-10646-UCS-4 csUCS4 ucs-4
UTF-32BE UTF32_BigEndian ibm-1232 ibm-1233
UTF-32LE UTF32_LittleEndian ibm-1234
UTF16_PlatformEndian
UTF16_OppositeEndian
UTF32_PlatformEndian
UTF32_OppositeEndian
UTF-7 windows-65000
IMAP-mailbox-name
SCSU
BOCU-1 csBOCU-1
CESU-8
ISO-8859-1 ibm-819 IBM819 cp819 latin1 8859_1 csISOLatin1
  iso-ir-100 ISO_8859-1:1987 l1 819
US-ASCII ASCII ANSI_X3.4-1968 ANSI_X3.4-1986 ISO_646.irv:1991
  iso_646.irv:1983 ISO646-US us csASCII iso-ir-6 cp367 ascii7
  646 windows-20127
ISO_2022,locale=ja,version=0 ISO-2022-JP csISO2022JP
ISO_2022,locale=ja,version=1 ISO-2022-JP-1 JIS JIS_Encoding
ISO_2022,locale=ja,version=2 ISO-2022-JP-2 csISO2022JP2
```

```
ISO_2022,locale=ja,version=3 JIS7 csJISEncoding
ISO_2022,locale=ja,version=4 JIS8
ISO_2022,locale=ko,version=0 ISO-2022-KR csISO2022KR
ISO_2022,locale=ko,version=1 ibm-25546
ISO_2022,locale=zh,version=0 ISO-2022-CN
ISO_2022,locale=zh,version=1 ISO-2022-CN-EXT
HZ HZ-GB-2312
ISCII,version=0 x-iscii-de windows-57002 iscii-dev
ISCII,version=1 x-iscii-be windows-57003 iscii-bng windows-57006
  x-iscii-as
ISCII,version=2 x-iscii-pa windows-57011 iscii-gur
ISCII,version=3 x-iscii-gu windows-57010 iscii-guj
ISCII,version=4 x-iscii-or windows-57007 iscii-ori
ISCII,version=5 x-iscii-ta windows-57004 iscii-tml
ISCII,version=6 x-iscii-te windows-57005 iscii-tlg
ISCII,version=7 x-iscii-ka windows-57008 iscii-knd
ISCII,version=8 x-iscii-ma windows-57009 iscii-mlm
gb18030 ibm-1392 windows-54936
LMBCS-1 lmbcs
LMBCS-2
LMBCS-3
LMBCS-4
LMBCS-5
LMBCS-6
LMBCS-8
LMBCS-11
LMBCS-16
LMBCS-17
LMBCS-18
LMBCS-19
ibm-367_P100-1995 ibm-367 IBM367
ibm-912_P100-1995 ibm-912 iso-8859-2 ISO_8859-2:1987 latin2
  csISOLatin2 iso-ir-101 l2 8859_2 cp912 912 windows-28592
ibm-913_P100-2000 ibm-913 iso-8859-3 ISO_8859-3:1988 latin3
  csISOLatin3 iso-ir-109 l3 8859_3 cp913 913 windows-28593
ibm-914_P100-1995 ibm-914 iso-8859-4 latin4 csISOLatin4
  iso-ir-110 ISO_8859-4:1988 l4 8859_4 cp914 914 windows-28594
ibm-915_P100-1995 ibm-915 iso-8859-5 cyrillic csISOLatinCyrillic
  iso-ir-144 ISO_8859-5:1988 8859_5 cp915 915 windows-28595
ibm-1089_P100-1995 ibm-1089 iso-8859-6 arabic csISOLatinArabic
  iso-ir-127 ISO_8859-6:1987 ECMA-114 ASMO-708 8859_6 cp1089
  1089 windows-28596 ISO-8859-6-I ISO-8859-6-E
ibm-813_P100-1995 ibm-813 iso-8859-7 greek greek8 ELOT_928
  ECMA-118 csISOLatinGreek iso-ir-126 ISO_8859-7:1987 8859_7
  cp813 813 windows-28597
ibm-916_P100-1995 ibm-916 iso-8859-8 hebrew csISOLatinHebrew
  iso-ir-138 ISO_8859-8:1988 ISO-8859-8-I ISO-8859-8-E 8859_8
  cp916 916 windows-28598
ibm-920_P100-1995 ibm-920 iso-8859-9 latin5 csISOLatin5
  iso-ir-148 ISO_8859-9:1989 l5 8859_9 cp920 920 windows-28599
  ECMA-128
ibm-921_P100-1995 ibm-921 iso-8859-13 8859_13 cp921 921
ibm-923_P100-1998 ibm-923 iso-8859-15 Latin-9 l9 8859_15 latin0
  csisolatin0 csisolatin9 iso8859_15_fdis cp923 923 windows-28605
ibm-942_P12A-1999 ibm-942 ibm-932 cp932 shift_jis78 sjis78
  ibm-942_VSUB_VPUA ibm-932_VSUB_VPUA
ibm-943_P15A-2003 ibm-943 Shift_JIS MS_Kanji csShiftJIS
  windows-31j csWindows31J x-sjis x-ms-cp932 cp932 windows-932
  cp943c IBM-943C ms932 pck sjis ibm-943_VSUB_VPUA
ibm-943_P130-1999 ibm-943 Shift_JIS cp943 943 ibm-943_VASCII_VSUB_VPUA
ibm-33722_P12A-1999 ibm-33722 ibm-5050 EUC-JP
```

```
  Extended_UNIX_Code_Packed_Format_for_Japanese
  csEUCPkdFmtJapanese X-EUC-JP eucjis windows-51932
  ibm-33722_VPUA IBM-eucJP
ibm-33722_P120-1999 ibm-33722 ibm-5050 cp33722 33722
  ibm-33722_VASCII_VPUA
ibm-954_P101-2000 ibm-954 EUC-JP
ibm-1373_P100-2002 ibm-1373 windows-950
windows-950-2000 Big5 csBig5 windows-950 x-big5
ibm-950_P110-1999 ibm-950 cp950 950
macos-2566-10.2 Big5-HKSCS big5hk HKSCS-BIG5
ibm-1375_P100-2003 ibm-1375 Big5-HKSCS
ibm-1386_P100-2002 ibm-1386 cp1386 windows-936 ibm-1386_VSUB_VPUA
windows-936-2000 GBK CP936 MS936 windows-936
ibm-1383_P110-1999 ibm-1383 GB2312 csGB2312 EUC-CN ibm-eucCN
  hp15CN cp1383 1383 ibm-1383_VPUA
ibm-5478_P100-1995 ibm-5478 GB_2312-80 chinese iso-ir-58
  csISO58GB231280 gb2312-1980 GB2312.1980-0
ibm-964_P110-1999 ibm-964 EUC-TW ibm-eucTW cns11643 cp964 964
  ibm-964_VPUA
ibm-949_P110-1999 ibm-949 cp949 949 ibm-949_VASCII_VSUB_VPUA
ibm-949_P11A-1999 ibm-949 cp949c ibm-949_VSUB_VPUA
ibm-970_P110-1995 ibm-970 EUC-KR KS_C_5601-1987 windows-51949
  csEUCKR ibm-eucKR KSC_5601 5601 ibm-970_VPUA
ibm-971_P100-1995 ibm-971 ibm-971_VPUA
ibm-1363_P11B-1998 ibm-1363 KS_C_5601-1987 KS_C_5601-1989 KSC_5601
  csKSC56011987 korean iso-ir-149 5601 cp1363 ksc windows-949
  ibm-1363_VSUB_VPUA
ibm-1363_P110-1997 ibm-1363 ibm-1363_VASCII_VSUB_VPUA
windows-949-2000 windows-949 KS_C_5601-1987 KS_C_5601-1989
  KSC_5601 csKSC56011987 korean iso-ir-149 ms949
ibm-1162_P100-1999 ibm-1162
ibm-874_P100-1995 ibm-874 ibm-9066 cp874 TIS-620 tis620.2533
  eucTH cp9066
windows-874-2000 TIS-620 windows-874 MS874
ibm-437_P100-1995 ibm-437 IBM437 cp437 437 csPC8CodePage437
  windows-437
ibm-850_P100-1995 ibm-850 IBM850 cp850 850 csPC850Multilingual
  windows-850
ibm-851_P100-1995 ibm-851 IBM851 cp851 851 csPC851
ibm-852_P100-1995 ibm-852 IBM852 cp852 852 csPCp852 windows-852
ibm-855_P100-1995 ibm-855 IBM855 cp855 855 csIBM855 csPCp855
ibm-856_P100-1995 ibm-856 cp856 856
ibm-857_P100-1995 ibm-857 IBM857 cp857 857 csIBM857 windows-857
ibm-858_P100-1997 ibm-858 IBM00858 CCSID00858 CP00858
  PC-Multilingual-850+euro cp858
ibm-860_P100-1995 ibm-860 IBM860 cp860 860 csIBM860
ibm-861_P100-1995 ibm-861 IBM861 cp861 861 cp-is csIBM861
  windows-861
ibm-862_P100-1995 ibm-862 IBM862 cp862 862 csPC862LatinHebrew
  DOS-862 windows-862
ibm-863_P100-1995 ibm-863 IBM863 cp863 863 csIBM863
ibm-864_X110-1999 ibm-864 IBM864 cp864 csIBM864
ibm-865_P100-1995 ibm-865 IBM865 cp865 865 csIBM865
ibm-866_P100-1995 ibm-866 IBM866 cp866 866 csIBM866 windows-866
ibm-867_P100-1998 ibm-867 cp867
ibm-868_P100-1995 ibm-868 IBM868 CP868 868 csIBM868 cp-ar
ibm-869_P100-1995 ibm-869 IBM869 cp869 869 cp-gr csIBM869
  windows-869
ibm-878_P100-1996 ibm-878 KOI8-R koi8 csKOI8R cp878
ibm-901_P100-1999 ibm-901
ibm-902_P100-1999 ibm-902
```

```
ibm-922_P100-1999 ibm-922 cp922 922
ibm-4909_P100-1999 ibm-4909
ibm-5346_P100-1998 ibm-5346 windows-1250 cp1250
ibm-5347_P100-1998 ibm-5347 windows-1251 cp1251
ibm-5348_P100-1997 ibm-5348 windows-1252 cp1252
ibm-5349_P100-1998 ibm-5349 windows-1253 cp1253
ibm-5350_P100-1998 ibm-5350 windows-1254 cp1254
ibm-9447_P100-2002 ibm-9447 windows-1255 cp1255
windows-1256-2000 windows-1256 cp1256
ibm-9449_P100-2002 ibm-9449 windows-1257 cp1257
ibm-5354_P100-1998 ibm-5354 windows-1258 cp1258
ibm-1250_P100-1995 ibm-1250 windows-1250
ibm-1251_P100-1995 ibm-1251 windows-1251
ibm-1252_P100-2000 ibm-1252 windows-1252
ibm-1253_P100-1995 ibm-1253 windows-1253
ibm-1254_P100-1995 ibm-1254 windows-1254
ibm-1255_P100-1995 ibm-1255
ibm-5351_P100-1998 ibm-5351 windows-1255
ibm-1256_P110-1997 ibm-1256
ibm-5352_P100-1998 ibm-5352 windows-1256
ibm-1257_P100-1995 ibm-1257
ibm-5353_P100-1998 ibm-5353 windows-1257
ibm-1258_P100-1997 ibm-1258 windows-1258
macos-0_2-10.2 macintosh mac csMacintosh windows-10000
macos-6-10.2 x-mac-greek windows-10006 macgr
macos-7_3-10.2 x-mac-cyrillic windows-10007 maccy
macos-29-10.2 x-mac-centraleurroman windows-10029 x-mac-ce macce
macos-35-10.2 x-mac-turkish windows-10081 mactr
ibm-1051_P100-1995 ibm-1051 hp-roman8 roman8 r8 csHPRoman8
ibm-1276_P100-1995 ibm-1276 Adobe-Standard-Encoding
  csAdobeStandardEncoding
ibm-1277_P100-1995 ibm-1277 Adobe-Latin1-Encoding
ibm-1006_P100-1995 ibm-1006 cp1006 1006
ibm-1098_P100-1995 ibm-1098 cp1098 1098
ibm-1124_P100-1996 ibm-1124 cp1124 1124
ibm-1125_P100-1997 ibm-1125 cp1125
ibm-1129_P100-1997 ibm-1129
ibm-1131_P100-1997 ibm-1131 cp1131
ibm-1133_P100-1997 ibm-1133
ibm-1381_P110-1999 ibm-1381 cp1381 1381
ibm-37_P100-1995 ibm-37 IBM037 ibm-037 ebcdic-cp-us ebcdic-cp-ca
  ebcdic-cp-wt ebcdic-cp-nl csIBM037 cp037 037 cpibm37 cp37
ibm-273_P100-1995 ibm-273 IBM273 CP273 csIBM273 ebcdic-de cpibm273
  273
ibm-277_P100-1995 ibm-277 IBM277 cp277 EBCDIC-CP-DK EBCDIC-CP-NO
  csIBM277 ebcdic-dk cpibm277 277
ibm-278_P100-1995 ibm-278 IBM278 cp278 ebcdic-cp-fi ebcdic-cp-se
  csIBM278 ebcdic-sv cpibm278 278
ibm-280_P100-1995 ibm-280 IBM280 CP280 ebcdic-cp-it csIBM280
  cpibm280 280
ibm-284_P100-1995 ibm-284 IBM284 CP284 ebcdic-cp-es csIBM284
  cpibm284 284
ibm-285_P100-1995 ibm-285 IBM285 CP285 ebcdic-cp-gb csIBM285
  ebcdic-gb cpibm285 285
ibm-290_P100-1995 ibm-290 IBM290 cp290 EBCDIC-JP-kana csIBM290
ibm-297_P100-1995 ibm-297 IBM297 cp297 ebcdic-cp-fr csIBM297
  cpibm297 297
ibm-420_X120-1999 ibm-420 IBM420 cp420 ebcdic-cp-ar1 csIBM420 420
ibm-424_P100-1995 ibm-424 IBM424 cp424 ebcdic-cp-he csIBM424 424
ibm-500_P100-1995 ibm-500 IBM500 CP500 ebcdic-cp-be csIBM500
  ebcdic-cp-ch cpibm500 500
```

```
ibm-803_P100-1999 ibm-803 cp803
ibm-838_P100-1995 ibm-838 IBM-Thai csIBMThai cp838 838 ibm-9030
ibm-870_P100-1995 ibm-870 IBM870 CP870 ebcdic-cp-roece
  ebcdic-cp-yu csIBM870
ibm-871_P100-1995 ibm-871 IBM871 ebcdic-cp-is csIBM871 CP871
  ebcdic-is cpibm871 871
ibm-875_P100-1995 ibm-875 IBM875 cp875 875
ibm-918_P100-1995 ibm-918 IBM918 CP918 ebcdic-cp-ar2 csIBM918
ibm-930_P120-1999 ibm-930 ibm-5026 cp930 cpibm930 930
ibm-933_P110-1995 ibm-933 cp933 cpibm933 933
ibm-935_P110-1999 ibm-935 cp935 cpibm935 935
ibm-937_P110-1999 ibm-937 cp937 cpibm937 937
ibm-939_P120-1999 ibm-939 ibm-931 ibm-5035 cp939 939
ibm-1025_P100-1995 ibm-1025 cp1025 1025
ibm-1026_P100-1995 ibm-1026 IBM1026 CP1026 csIBM1026 1026
ibm-1047_P100-1995 ibm-1047 IBM1047 cpibm1047
ibm-1097_P100-1995 ibm-1097 cp1097 1097
ibm-1112_P100-1995 ibm-1112 cp1112 1112
ibm-1122_P100-1999 ibm-1122 cp1122 1122
ibm-1123_P100-1995 ibm-1123 cp1123 1123 cpibm1123
ibm-1130_P100-1997 ibm-1130
ibm-1132_P100-1998 ibm-1132
ibm-1140_P100-1997 ibm-1140 IBM01140 CCSID01140 CP01140 cp1140
  cpibm1140 ebcdic-us-37+euro
ibm-1141_P100-1997 ibm-1141 IBM01141 CCSID01141 CP01141 cp1141
  cpibm1141 ebcdic-de-273+euro
ibm-1142_P100-1997 ibm-1142 IBM01142 CCSID01142 CP01142 cp1142
  cpibm1142 ebcdic-dk-277+euro ebcdic-no-277+euro
ibm-1143_P100-1997 ibm-1143 IBM01143 CCSID01143 CP01143 cp1143
  cpibm1143 ebcdic-fi-278+euro ebcdic-se-278+euro
ibm-1144_P100-1997 ibm-1144 IBM01144 CCSID01144 CP01144 cp1144
  cpibm1144 ebcdic-it-280+euro
ibm-1145_P100-1997 ibm-1145 IBM01145 CCSID01145 CP01145 cp1145
  cpibm1145 ebcdic-es-284+euro
ibm-1146_P100-1997 ibm-1146 IBM01146 CCSID01146 CP01146 cp1146
  cpibm1146 ebcdic-gb-285+euro
ibm-1147_P100-1997 ibm-1147 IBM01147 CCSID01147 CP01147 cp1147
  cpibm1147 ebcdic-fr-297+euro
ibm-1148_P100-1997 ibm-1148 IBM01148 CCSID01148 CP01148 cp1148
  cpibm1148 ebcdic-international-500+euro
ibm-1149_P100-1997 ibm-1149 IBM01149 CCSID01149 CP01149 cp1149
  cpibm1149 ebcdic-is-871+euro
ibm-1153_P100-1999 ibm-1153 cpibm1153
ibm-1154_P100-1999 ibm-1154 cpibm1154
ibm-1155_P100-1999 ibm-1155 cpibm1155
ibm-1156_P100-1999 ibm-1156 cpibm1156
ibm-1157_P100-1999 ibm-1157 cpibm1157
ibm-1158_P100-1999 ibm-1158 cpibm1158
ibm-1160_P100-1999 ibm-1160 cpibm1160
ibm-1164_P100-1999 ibm-1164 cpibm1164
ibm-1364_P110-1997 ibm-1364 cp1364
ibm-1371_P100-1999 ibm-1371 cpibm1371
ibm-1388_P103-2001 ibm-1388 ibm-9580
ibm-1390_P110-2003 ibm-1390 cpibm1390
ibm-1399_P110-2003 ibm-1399
ibm-16684_P110-2003 ibm-16684
ibm-4899_P100-1998 ibm-4899 cpibm4899
ibm-4971_P100-1999 ibm-4971 cpibm4971
ibm-12712_P100-1998 ibm-12712 cpibm12712 ebcdic-he
ibm-16804_X110-1999 ibm-16804 cpibm16804 ebcdic-ar
ibm-1137_P100-1999 ibm-1137
```

```
ibm-5123_P100-1999 ibm-5123
ibm-8482_P100-1999 ibm-8482
ibm-37_P100-1995,swaplfnl ibm-37-s390 ibm037-s390
ibm-1047_P100-1995,swaplfnl ibm-1047-s390
ibm-1140_P100-1997,swaplfnl ibm-1140-s390
ibm-1142_P100-1997,swaplfnl ibm-1142-s390
ibm-1143_P100-1997,swaplfnl ibm-1143-s390
ibm-1144_P100-1997,swaplfnl ibm-1144-s390
ibm-1145_P100-1997,swaplfnl ibm-1145-s390
ibm-1146_P100-1997,swaplfnl ibm-1146-s390
ibm-1147_P100-1997,swaplfnl ibm-1147-s390
ibm-1148_P100-1997,swaplfnl ibm-1148-s390
ibm-1149_P100-1997,swaplfnl ibm-1149-s390
ibm-1153_P100-1999,swaplfnl ibm-1153-s390
ibm-12712_P100-1998,swaplfnl ibm-12712-s390
ibm-16804_X110-1999,swaplfnl ibm-16804-s390
ebcdic-xml-us
```

Appendix C: Security Database Upgrade for Firebird 2

A. Peshkov

Security Upgrade Script

```
/* Script security_database.sql
 *
 *  The contents of this file are subject to the Initial
 *  Developer's Public License Version 1.0 (the "License");
 *  you may not use this file except in compliance with the
 *  License. You may obtain a copy of the License at
 *  http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_idpl.
 *
 *  Software distributed under the License is distributed AS IS,
 *  WITHOUT WARRANTY OF ANY KIND, either express or implied.
 *  See the License for the specific language governing rights
 *  and limitations under the License.
 *
 *  The Original Code was created by Alex Peshkov on 16-Nov-2004
 *  for the Firebird Open Source RDBMS project.
 *
 *  Copyright (c) 2004 Alex Peshkov
 *  and all contributors signed below.
 *
 *  All Rights Reserved.
 *  Contributor(s): ______________________________________.
 *
 */

-- 1. temporary table to alter domains correctly.
CREATE TABLE UTMP (
  USER_NAME VARCHAR(128) CHARACTER SET ASCII,
  SYS_USER_NAME VARCHAR(128) CHARACTER SET ASCII,
  GROUP_NAME VARCHAR(128) CHARACTER SET ASCII,
  UID   INTEGER,
  GID   INTEGER,
  PASSWD   VARCHAR(64) CHARACTER SET BINARY,
  PRIVILEGE  INTEGER,
  COMMENT  BLOB SUB_TYPE TEXT SEGMENT SIZE 80
    CHARACTER SET UNICODE_FSS,
  FIRST_NAME  VARCHAR(32) CHARACTER SET UNICODE_FSS
    DEFAULT _UNICODE_FSS '',
  MIDDLE_NAME VARCHAR(32) CHARACTER SET UNICODE_FSS
    DEFAULT _UNICODE_FSS '',
  LAST_NAME VARCHAR(32) CHARACTER SET UNICODE_FSS
    DEFAULT _UNICODE_FSS ''
  );
COMMIT;

-- 2. save users data
INSERT INTO UTMP(USER_NAME, SYS_USER_NAME, GROUP_NAME,
  UID, GID, PRIVILEGE, COMMENT, FIRST_NAME, MIDDLE_NAME,
  LAST_NAME, PASSWD)
SELECT USER_NAME, SYS_USER_NAME, GROUP_NAME,
```

```
  UID, GID, PRIVILEGE, COMMENT, FIRST_NAME, MIDDLE_NAME,
  LAST_NAME, PASSWD
 FROM USERS;
COMMIT;

-- 3. drop old tables and domains
DROP TABLE USERS;
DROP TABLE HOST_INFO;
COMMIT;

DROP DOMAIN COMMENT;
DROP DOMAIN NAME_PART;
DROP DOMAIN GID;
DROP DOMAIN HOST_KEY;
DROP DOMAIN HOST_NAME;
DROP DOMAIN PASSWD;
DROP DOMAIN UID;
DROP DOMAIN USER_NAME;
DROP DOMAIN PRIVILEGE;
COMMIT;

-- 4. create new objects in database
CREATE DOMAIN RDB$COMMENT AS BLOB SUB_TYPE TEXT SEGMENT SIZE 80
  CHARACTER SET UNICODE_FSS;
CREATE DOMAIN RDB$NAME_PART AS VARCHAR(32)
  CHARACTER SET UNICODE_FSS DEFAULT _UNICODE_FSS '';
CREATE DOMAIN RDB$GID AS INTEGER;
CREATE DOMAIN RDB$PASSWD AS VARCHAR(64) CHARACTER SET BINARY;
CREATE DOMAIN RDB$UID AS INTEGER;
CREATE DOMAIN RDB$USER_NAME AS VARCHAR(128)
CHARACTER SET UNICODE_FSS;
CREATE DOMAIN RDB$USER_PRIVILEGE AS INTEGER;
COMMIT;


CREATE TABLE RDB$USERS (
  RDB$USER_NAME       RDB$USER_NAME NOT NULL PRIMARY KEY,
  /* local system user name
     for setuid for file permissions */
  RDB$SYS_USER_NAME   RDB$USER_NAME,
  RDB$GROUP_NAME      RDB$USER_NAME,
  RDB$UID             RDB$UID,
  RDB$GID             RDB$GID,
  RDB$PASSWD          RDB$PASSWD,  /* SEE NOTE BELOW */

  /* Privilege level of user -
     mark a user as having DBA privilege */
  RDB$PRIVILEGE       RDB$USER_PRIVILEGE,

  RDB$COMMENT         RDB$COMMENT,
  RDB$FIRST_NAME      RDB$NAME_PART,
  RDB$MIDDLE_NAME     RDB$NAME_PART,
  RDB$LAST_NAME       RDB$NAME_PART);
COMMIT;

CREATE VIEW USERS (USER_NAME, SYS_USER_NAME, GROUP_NAME,
  UID, GID, PASSWD, PRIVILEGE, COMMENT, FIRST_NAME,
  MIDDLE_NAME, LAST_NAME, FULL_NAME) AS

  SELECT RDB$USER_NAME, RDB$SYS_USER_NAME, RDB$GROUP_NAME,
    RDB$UID, RDB$GID, RDB$PASSWD, RDB$PRIVILEGE, RDB$COMMENT,
```

```
    RDB$FIRST_NAME, RDB$MIDDLE_NAME, RDB$LAST_NAME,
    RDB$first_name || _UNICODE_FSS ' ' || RDB$middle_name
      || _UNICODE_FSS ' ' || RDB$last_name
    FROM RDB$USERS
    WHERE CURRENT_USER = 'SYSDBA'
       OR CURRENT_USER = RDB$USERS.RDB$USER_NAME;
COMMIT;

GRANT ALL ON RDB$USERS to VIEW USERS;
GRANT SELECT ON USERS to PUBLIC;
GRANT UPDATE(PASSWD, GROUP_NAME, UID, GID, FIRST_NAME,
  MIDDLE_NAME, LAST_NAME)
 ON USERS TO PUBLIC;
COMMIT;

-- 5. move data from temporary table and drop it
INSERT INTO RDB$USERS(RDB$USER_NAME, RDB$SYS_USER_NAME,
  RDB$GROUP_NAME, RDB$UID, RDB$GID, RDB$PRIVILEGE, RDB$COMMENT,
  RDB$FIRST_NAME, RDB$MIDDLE_NAME, RDB$LAST_NAME, RDB$PASSWD)
SELECT USER_NAME, SYS_USER_NAME, GROUP_NAME, UID, GID,
  PRIVILEGE, COMMENT, FIRST_NAME, MIDDLE_NAME, LAST_NAME,
  PASSWD
    FROM UTMP;
COMMIT;

DROP TABLE UTMP;
COMMIT;
```

Note

This field should be constrained as NOT NULL. For information about this, see Nullability of RDB$PASSWD in the Security chapter.