Firebird 2.0 Release Notes

Helen Borrie (Collator/Editor)

28 June 2006 - Document v. 0200_82 - for Firebird 2.0 Release Candidate 3

Firebird 2.0 Release Notes

28 June 2006 - Document v. 0200_82 - for Firebird 2.0 Release Candidate 3

Helen Borrie (Collator/Editor)

Table of Contents

1. General Notes 1
    - Back Up! 1
    - Compatibility Issues 1
    - Incomplete Implementations 1
    - Bug Reporting and Support 2
2. New in Firebird 2.0 3
    - Derived Tables 3
    - PSQL Now Supports Named Cursors 3
    - Reimplemented Protocols on Windows 3
    - Local Protocol--XNET 3
    - Change to WNET (NetBEUI) Protocol 4
    - Reworking of Garbage Collection 4
    - Storing Databases on Raw Devices (POSIX Only) 5
    - Porting of the Services API to Classic is Complete 5
    - Lock Timeout for WAIT Transactions 5
    - New Implementation of String Search Operators 6
    - Reworking of Updatable Views 6
    - Additional Database Shutdown Modes Introduced 6
    - UDFs Improved re NULL Handling 7
    - Signalling SQL NULL 7
    - Run-time Checking for Concatenation Overflow 7
    - Changes to Synchronisation Logic 7
    - Support for 64-bit Platforms 8
    - Record Enumeration Limits Increased 8
    - Debugging Improvements 8
    - Improved Reporting from Bugchecks 8
    - Updated Internal Structure Reporting 8
    - New Debug Logging Facilities 8
    - Diagnostic Enhancement 8
    - Improved Connection Handling on POSIX Superserver 8
    - PSQL Invariant Tracking Reworked 8
    - ROLLBACK RETAIN Syntax Support 9
    - No More Registry Search on Win32 Servers 9
    - More Optimizer Improvements 9
    - ODS Changes 9
3. Changes to the Firebird API and ODS 11
    - API (Application Programming Interface) 11
    - Cleanup of ibase.h 11
    - Lock Timeout for WAIT Transactions 11
    - isc_dsql_info() Now Includes Relation Aliases 11
    - Enhancement to isc_blob_lookup_desc() 11
    - API Identifies Client Version 12
    - Additions to the isc_database_info() Structure 12
    - Additions to the isc_transaction_info() Structure 12
    - Improved Services API 13
    - ODS (On-Disk Structure) Changes 14
4. Data Definition Language (DDL) 16
    - New and Enhanced Syntaxes 16
    - CREATE SEQUENCE 16
    - REVOKE ADMIN OPTION FROM 17
    - SET/DROP DEFAULT Clauses for ALTER TABLE 17
    - New Syntaxes for Changing Exceptions 17

ALTER EXTERNAL FUNCTION 18
COMMENT Statement Implemented 18
Extensions to CREATE VIEW Specification 18
RECREATE TRIGGER Statement Implemented 19
Usage Enhancements 19
5. Data Manipulation Language (DML) 21
New and Extended DSQL Syntaxes 21
EXECUTE BLOCK Statement 21
Derived Tables 22
ROLLBACK RETAIN Syntax 24
ROWS Syntax 24
Enhancements to UNION Handling 25
IIF Expression Syntax Added 26
CAST() Behaviour Improved 26
Built-in Function SUBSTRING() Enhanced 27
Enhancements to NULL Logic 27
CROSS JOIN is Now Supported 29
Subqueries and INSERT Statements Can Now Accept UNION Sets 30
New Extensions to UPDATE and DELETE Syntaxes 30
New Context Variables 30
Improvements in Handling User-specified Query Plans 34
Improvements in Sorting 36
NEXT VALUE FOR Expression Syntax 37
RETURNING Clause for Insert Statements 38
DSQL parsing of table aliases is stricter 38
SELECT Statement & Expression Syntax 40
6. New Reserved Words and Changes 42
Newly Reserved Words 42
Changed from Non-reserved to Reserved 42
Keywords Added as Non-reserved 42
Keywords No Longer Reserved 43
No Longer Reserved as Keywords 43
7. Stored Procedure Language (PSQL) 44
PSQL Enhancements 44
Explicit Cursors 44
Defaults for Stored Procedure Arguments 45
LEAVE <label> Syntax Support 47
OLD Context Variables Now Read-only 48
PSQL Stack Trace 49
Call a UDF as a Void Function (Procedure) 50
8. Enhancements to Indexing 51
252-byte index length limit is gone 51
Expression Indexes 51
Changes to Null keys handling 52
Improved Index Compression 52
Selectivity Maintenance per Segment 52
Firebird Index Structure from ODS11 Onward 53
New flag for the new index structure 54
Duplicate nodes 54
Jump nodes 55
NULL state 56
9. Optimizations 58
Improved PLAN Clause 58
Optimizer Improvements 58
For All Databases 58
For ODS 11 Databases only 60
10. New Features for Text Data 61

New String Functions 61
LOWER() 61
TRIM() 61
New String Size Functions 62
New INTL Interface for Non-ASCII Character Sets 63
Architecture 63
Enhancements 63
New Character Sets and Collations Implemented 67
Character Set Bug Fixes 68
11. Security in Firebird 2 70
Summary of Changes 70
New security database 70
Better password encryption 70
Users can modify their own passwords 70
Non-server access to security database is rejected 70
Active protection from brute-force attack 70
Vulnerabilities have been closed 71
Details of the Security Changes in Firebird 2.0 71
Authentication 71
GSEC in Firebird 2 73
Protection from Brute-force Hacking 73
Classic Server on POSIX 73
Dealing with the New Security Database 74
Doing the Security Database Upgrade 74
12. Command-line Utilities 76
Backup Tools 76
New On-line Incremental Backup 76
GBak Backup/Porting/Restore Utility 78
ISQL Query Utility 79
New Switches 79
New Commands 81
ISQL Bugs Fixed 83
GSec Authentication Manager 84
GSEC return code 84
GFix Server Utility 85
New Shutdown States (Modes) 85
13. External Functions (UDFs) 87
Ability to Signal SQL NULL via a Null Pointer 87
UDF library diagnostic messages improved 88
UDFs Added and Changed 88
IB_UDF_srand() 88
IB_UDF_lower 88
General UDF Changes 89
Build Changes 89
14. New Configuration Parameters and Changes 90
ExternalFileAccess 90
LegacyHash 90
Redirection 90
About Multi-hop 90
GCPolicy 91
New parameter OldColumnNaming 91
UsePriorityScheduler 91
TCPNoNagle has changed 91
Removed or Deprecated Parameters 91
CreateInternalWindow 91
DeadThreadsCollection is no longer used 92
15. Known Compatibility Issues 93

The FIREBIRD Variable 93
Security in Firebird 2 (All Platforms) 93
SQL Migration Issues 94
DDL 94
DML 95
PSQL 97
Configuration Parameters 98
Command-line Tools 98
Change to GBAK -R Semantics 99
Performance 99
Firebird API 99
Windows-Specific Issues 100
Windows Local Connection Protocol with XNet 100
Client Impersonation No Longer Works 100
Interactive Option Added to instsvc.exe 100
16. INSTALLATION NOTES 102
Windows 32-bit Installs 102
READ THIS FIRST! 102
Other Pre-installation Issues 104
Using the Win32 Firebird Installer 106
Installing Superserver from a zip kit 108
Other Win32 Issues 109
Updated Notes for Windows Embedded 110
POSIX Platforms 112
READ THIS FIRST 112
Installing on Linux 114
Testing your Linux installation 115
Utility Scripts 117
Linux Server Tips 117
Uninstalling on Linux 117
Solaris 118
MacOS X 118
FreeBSD 118
Debian 118
17. Bugs Fixed 119
General Engine Bugs 119
Services Manager 126
GFix Bugs 126
DSQL Bugs 127
PSQL Bugs 129
Crash Conditions 130
Remote Interface Bugs 133
Indexing & Optimization 134
Vulnerabilities 135
ISQL Bugs 136
International Character Set Bugs 137
SQL Privileges 137
UDF Bugs 138
GBak 139
GPre 140
GSTAT 141
fb_lock_print 141
Linux Installs 141
Code Clean-up 141
Bugs Introduced in RC2 142
18. Appendix to Firebird 2 Release Notes 144
Security Upgrade Script 144

List of Figures

8.1. Existing structure (ODS10 and lower) 53
8.2. New ODS11 structure 54
8.3. Example data ((x) = size in x bytes) 55
8.4. Examples 56

Chapter 1

General Notes

The v.2.0 release cycle of Firebird brings with it a large collection of long-awaited enhancements under the hood that significantly improve performance, security and support for international languages. Several annoying limitations, along with a huge pile of old bugs inherited from the legacy code, have gone. Many of the command- line tools have been refurbished and this release introduces the all-new incremental backup tools NBak and NBackup.

The old “252 bytes or less” limit on index size is gone for good, replaced by much-extended limits that depend on page size. Calculation of index statistics has been revamped to improve the choices the optimizer has available, especially for complex outer join distributions and DISTINCT queries.

Many new additions have been made to the SQL language, including support for derived tables (SELECT ... FROM (SELECT ... FROM), PSQL blocks in dynamic SQL through the new EXECUTE BLOCK syntax and some handy new extensions in PSQL itself.

Before you read on to find out about all of these new features, please take a moment to note some points about approaching this new release.

Back Up!

The on-disk structure (ODS) of the databases created under Firebird has changed. Although Firebird 2.0 will connect to databases having older ODS versions, most of the new features will not be available to them.

Make transportable gbak backups of your existing databases--including your old security.fdb or (even older) isc4.gdb security databases--*before* you uninstall the older Firebird server and set out to install Firebird 2.0. Before you proceed, restore these backups in a temporary location, using the old gbak, and verify that the backups are good.

Compatibility Issues

Naturally, with so much bug-removal and closing of holes, there are sure to be things that worked before and now no longer work. A collection of Known Compatibility Issues is provided to assist you to work out what needs to be done in your existing system to make it compatible with Firebird 2.0.

Give special attention to the changes required in the area of user authentication.

Incomplete Implementations

In a couple of areas, planned implementations could not be completed for the v.2.0 release and will be deferred to later sub-releases:

64-bit Support

For Linux, 64-bit builds for both Superserver and Classic--so far tested on AMD64 only--are ready and available for installing.
For Windows, development and testing of a 64-bit version is hampered by lack of both reliable Microsoft 64-bit C/C++ compilers and reliable 64-bit support in the operating systems. Plans to provide this support for Firebird have not changed and we hope to have some builds for testing some time before the middle of the year.

Installer Support for Multiple Server Installations

Although the capability to run multiple Firebird servers simultaneously on a single host has been present since Firebird 1.5, we still do not provide the ability to set them up through our installer programs and scripts.

Bug Reporting and Support

As a release candidate, Firebird 2.0 is still in beta. No further features will be added before the release proper but our current purpose is find any late-breaking bugs and "gotchas". Please make a point of reading the instructions for bug reporting in the article How to Report Bugs Effectively, at the Firebird Project website.

Follow these guidelines as you test this software:

1. Write detailed bug reports, supplying the exact server model and 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. Don't post reports to the main Bug Tracker, which is for stable releases ONLY.

2. If you want to start a discussion thread about a bug or an implementation, do so by subscribing and posting to the Testers' list or directly to the firebird-devel list.

3. If you are a novice with Firebird and need "newbie" advice, we recommend that you don't start your experience here. Download the latest stable v.1.5 release kit for self-teaching and use the Firebird 1.5 Quick Start Guide and the firebird-support list to help you get started.

4. Don't use the regular bug-tracker or the firebird-support list to report bugs in the beta or to ask for expanded details about how a new feature works.

5. Consider joining up with your regional (language) group of formal field-testers. Details and contacts are in the QA section of the Firebird Developers' Corner.

H A P P Y T E S T I N G !

--The Firebird Project

Chapter 2

New in Firebird 2.0

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 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 interprocess 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. Horsun

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 interesting transaction (OIT), which appears as the *Oldest Transaction* 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 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 OIT 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.

Storing Databases on Raw Devices (POSIX Only)

E. Kunze, N. Samofatov

You can now store databases on raw devices under a POSIX OS and refer to the devices using database aliases. More information to come.

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 DDL Migration Issues in the Compatibility chapter of these notes.

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.

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.

ODS Changes

Various Contributors

The new On-Disk Structure (ODS) is ODS11.

For more information, see the chapter ODS Changes.

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_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. Horsun

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. Horsun

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:

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.

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. Horsun

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

Data Definition Language (DDL)

New and Enhanced Syntaxes

The following statement syntaxes and structures have been added to Firebird 2:

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.

REVOKE ADMIN OPTION FROM

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).

New 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 an 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 Implemented

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.

Declare BLOB Subtypes by Known Descriptive Identifiers

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 5

Data Manipulation Language (DML)

New and Extended DSQL Syntaxes

In this section are details of DML language statements or constructs that have been added to the DSQL language set in Firebird 2.0.

EXECUTE BLOCK Statement

V. Horsun

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 upper-most-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.

IIF Expression Syntax Added

O. Loa

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

is implemented as a shortcut for

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

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

Example

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

CAST() Behaviour Improved

D. Yemanov

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
```

Built-in Function SUBSTRING() Enhanced

O. Loa, D. Yemanov

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!)

Enhancements to NULL Logic

The following features involving NULL in DSQL have been implemented:

(NULL=NULL) Can Return True for DISTINCT Test

O. Loa, D. Yemanov

A new equivalence predicate behaves exactly like the equality/inequality predicates, but tests whether one value is distinct from the other. Thus, it treats (NULL = NULL) as TRUE. 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.

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
```

CROSS JOIN is Now Supported

D. Yemanov

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
```

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>]]
```

New Context Variables

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

New Context Variable ROW_COUNT

D. Yemanov

ROW_COUNT can now return the number of rows returned by a SELECT statement. More information to come.

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

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:

- 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 AFTER INSERT OR UPDATE
as
begin
  new.jrn_lastuser = rdb$get_context('USER_TRANSACTION', 'User_ID');
  new.jrn_lastaddr = rdb$get_context('SYSTEM', 'CLIENT_ADDRESS');
  new.jrn_lasttransaction = rdb$get_context('USER_TRANSACTION', 'Trn_ID');
end ^
commit ^
execute procedure set_context('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 SELECT * Sets by Degree Number

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

Example

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

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 Syntax

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.

DSQL parsing of table aliases 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
```

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 express
    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.

Chapter 6

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
BOTH
CHAR_LENGTH
CHARACTER_LENGTH
CLOSE
CROSS
FETCH
LEADING
LOWER
OCTET_LENGTH
OPEN
ROWS
TRAILING
TRIM
```

Changed from Non-reserved to Reserved

```
USING
```

Keywords Added as Non-reserved

```
BACKUP *
BLOCK *
COLLATION
COMMENT *
DIFFERENCE *
IIF *
NEXT
SCALAR_ARRAY *
SEQUENCE
RESTART
RETURNING *
```

Keywords No Longer Reserved

```
ACTION
RESTRICT
WEEKDAY *
CASCADE
ROLE
YEARDAY *
FREE_IT *
TYPE
```

No Longer Reserved as Keywords

```
BASENAME *
GROUP_COMMIT_WAIT *
NUM_LOG_BUFS *
CACHE *
LOGFILE *
RAW_PARTITIONS *
CHECK_POINT_LEN *
LOG_BUF_SIZE *
```

Chapter 7

Stored Procedure Language (PSQL)

PSQL Enhancements

The following enhancements have been made to the PSQL language extensions for stored procedures and triggers:

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. Horsun

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 (most inner) 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. Horsun

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). More information to come.

Chapter 8

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. Actual numbers and more information to come.

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. Horsun, A. Brinkman

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

  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 compatibil-

ity and still represents the total index selectivity, that is used for a full index match.

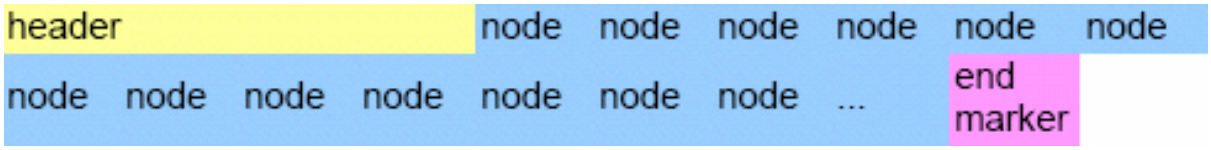
Firebird Index Structure from ODS11 Onward

© Abvisie 2005, Arno Brinkman

The aims achieved by the new structure were:

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

Figure 8.1. Existing structure (ODS10 and lower)



header =

```
  typedef struct btr {
  struct pag btr_header;
    SLONG btr_sibling;       // right sibling page
    SLONG btr_left_sibling;  // left sibling page
    SLONG btr_prefix_total;  // sum of all prefixes on page
    USHORT btr_relation;     // relation id for consistency
    USHORT btr_length;       // length of data in bucket
    UCHAR btr_id;            // index id for consistency
    UCHAR btr_level;         // index level (0 = leaf)
    struct btn btr_nodes[1];
  };
```

node =

```
  struct btn {
    UCHAR btn_prefix;    // size of compressed prefix
    UCHAR btn_length;    // length of data in node
    UCHAR btn_number[4]; // page or record number
    UCHAR btn_data[1];
};
```

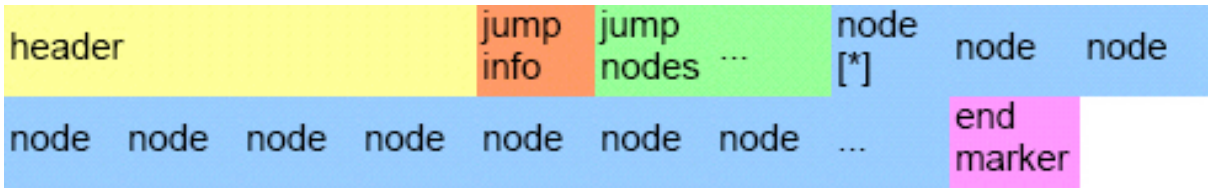
end marker = END_BUCKET or END_LEVEL

These are in place of record-number for leaf nodes and in place of page-number for non-leaf nodes.

If the node is a END_BUCKET marker then it should contain the same data as the first node on the next sibling page.

On an END_LEVEL marker prefix and length are zero, thus it contains no data. Also, every first node on a level (except leaf pages) contains a degeneration zero-length node.

Figure 8.2. New ODS11 structure



jump info =

```
struct IndexJumpInfo {
  USHORT firstNodeOffset; // offset to first node in page [*]
  USHORT jumpAreaSize;    // size area before a new jumpnode is made
  UCHAR  jumpers;         // nr of jump-nodes in page, with a maximum of 255
};
```

jump node =

```
struct IndexJumpNode {
  UCHAR* nodePointer; // pointer to where this node can be read from the page
  USHORT prefix;      // length of prefix against previous jump node
  USHORT length;      // length of data in jump node (together with prefix this
                      // is prefix for pointing node)
  USHORT offset;      // offset to node in page
  UCHAR* data;        // Data can be read from here
};
```

New flag for the new index structure

New flags are added to the `header->pag_flags`.

The flag `btr_large_keys (32)` is for storing compressed length/prefix and record-number. This meant also that length and prefix can be up to 1/4 of page-size (1024 for 4096 page-size) and is easy extensible in the future without changing disk-structure again.

Also the record-number can be easy extended to for example 40 bits. Those numbers are stored per 7-bits with 1 bit (highest) as marker (variable length encoding). Every new byte that needs to be stored is shifted by 7.

Examples

25 is stored as 1 byte 0x19, 130 = 2 bytes 0x82 0x01, 65535 = 3 bytes 0xFF 0xFF 0x03.

Duplicate nodes

A new flag is also added for storing record-number on every node (non-leaf pages). This speeds up index-retrieval on many duplicates. The flag is `btr_all_recordnumber (16)`.

With this added information, key-lookup on inserts/deletes with many duplicates (NULLs in foreign keys, for example) becomes much faster (such as the garbage collection!).

Beside that duplicate nodes (length = 0) don't store their length information, 3 bits from the first stored byte are used to determine if this nodes is a duplicate.

Beside the ZERO_LENGTH (4) there is also END_LEVEL (1), END_BUCKET (2), ZERO_PREFIX_ZERO_LENGTH (3) and ONE_LENGTH (5) marker. Number 6 and 7 are reserved for future use.

Jump nodes

A jump node is a reference to a node somewhere in the page.

It contains offset information about the specific node and the prefix data from the referenced node, but prefix compression is also done on the jump-nodes themselves.

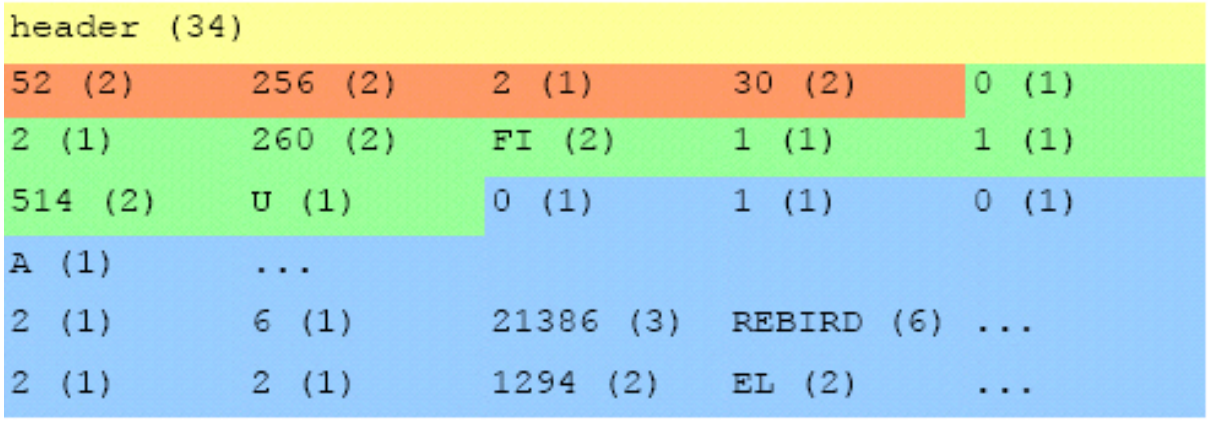
Ideally a new jump node is generated after the first node that is found after every `jumpAreaSize`, but that's only the case on deactivate/active an index or inserting nodes in the same order as they will be stored in the index.

If nodes are inserted between two jump node references only the offsets are updated, but only if the offsets don't exceed a specific threshold (+/-10 %).

When a node is deleted only offsets are updated or a jump node is removed. This means a little hole can exist between the last jump node and the first node, so we don't waste time on generating new jump-nodes.

The prefix and length are also stored by variable length encoding.

Figure 8.3. Example data ((x) = size in x bytes)



Pointer after fixed header = 0x22

Pointer after jump info = 0x29

Pointer to first jump node = 0x29 + 6 (jump node 1) + 5 (jump node 2) = 0x34

Jump node 1 is referencing to the node that represents FIREBIRD as data, because this node has a prefix of 2 the first 2 characters FI are stored also on the jump node.

Our next jump node points to a node that represents FUEL with also a prefix of 2. Thus jump node 2 should contain FU, but our previous node already contained the F so, due to prefix compression, this

one is ignored and only U is stored.

NULL state

The data that needs to be stored is determined in the procedure compress() in btr.cpp.

For ASC (ascending) indexes no data will be stored (key is zero length). This will automatically put them as first entry in the index and thus correct order (For single field index node length and prefix is zero).

DESC (descending) indexes will store a single byte with the value 0xFF (255). To distinguish between a value (empty string can be 255) and an NULL state we insert a byte of 0xFE (254) at the front of the data. This is only done for values that begin with 0xFF (255) or 0xFE (254), so we keep the right order.

Figure 8.4. Examples

nodes ASC index, 1 segment			
prefix	length	stored data	real value/state
0	0		NULL
0	0		NULL
0	1	x65 (A)	A
1	1	x65 (A)	AA
...	...	...	...
nodes DESC index, 1 segment			
prefix	length	stored data	real value/state
...	...	...	...
0	2	xFE xFE (þ) x4A (J)	0xFE 0x4A
1	1	xFF (ÿ)	0xFF
0	1	xFF	NULL
1	0	xFF	NULL
			END_LEVEL

nodes ASC index, 3 segment			
prefix	length	stored data	real value/state
0	0		NULL, NULL, NULL
0	10	x01(1) x70(F) x73(I) x82(R) x69(E) x01(1) x66(B) x73(I) x82(R) x68(D)	NULL, NULL, FIREBIRD
0	10	x02(2) x70(F) x73(I) x82(R) x69(E) x02(2) x66(B) x73(I) x82(R) x68(D)	NULL, FIREBIRD, NULL
0	10	x03(3) x70(F) x73(I) x82(R) x69(E) x03(3) x66(B) x73(I) x82(R) x68(D)	FIREBIRD, NULL, NULL
3	9	x00(0) x00(0) x02(2) x65(A) x00(0) x00(0) x00(0) x01(1) x66(B)	FI, A, B
...	...	...	...

nodes DESC index, 3 segment			
prefix	length	stored data	real value/state
0	12	xFC xB9 xB6 xFF xFF xFD xBE xFF xFF xFF xFE xBD	FI, A, B
3	17	xAD xBA xFC xBD xB6 xAD xBB xFD xFF xFF xFF xFF xFE xFF xFF xFF xFF	FIREBIRD, NULL, NULL
1	19	xFF xFF xFF xFF xFD xB9 xB6 xAD xBA xFD xBD xB6 xAD xBB xFE xFF xFF xFF xFF	NULL, FIREBIRD, NULL
6	14	xFF xFF xFF xFF xFE xB9 xB6 xAD xBA xFE xBD xB6 xAD xBB	NULL, NULL, FIREBIRD
11	4	xFF xFF xFF xFF	NULL, NULL, NULL
			END_LEVEL

Chapter 9

Optimizations

Improved PLAN Clause

D. Yemanov

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 in the DML section, Query Plans, Improvements in Handling User-specified Query Plans.

Optimizer Improvements

This chapter represents a collection of changes done in Firebird 2.0 to optimize many aspects of performance.

For All Databases

The following changes affect all databases.

Some General Improvements

O. Loa, D. Yemanov

- Much faster algorithms to process the dirty pages tree

  Firebird 2.0 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 information to come.

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 (
  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

Let's choose a 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

Content to come.

Segment-level Selectivities are Used

See Selectivity Maintenance per Segment in the Indexing chapter.

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.

Chapter 10

New Features for Text Data

New String Functions

Two new string functions were added:

LOWER()

A. dos Santos Fernandes

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

Example

```
isql -q -ch dos850

SQL> create database 'test.fdb';
SQL> create table t (c char(1) character set dos850);
SQL> insert into t values ('A');
SQL> insert into t values ('E');
SQL> insert into t values ('Á');;
SQL> insert into t values ('É');
SQL>
C      LOWER
====== ======
A      a
E      e
Á      á
É      é
```

TRIM()

A. dos Santos Fernandes

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

Syntax Pattern

```
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.
4. If <trim specification> and <trim character> is not specified, FROM should not be specified.

Examples

A)

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

B)

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

New String Size Functions

A. dos Santos Fernandes

Three new functions will return information about the size of strings:

1. BIT_LENGTH returns the length of a string in bits
2. CHAR_LENGTH/CHARACTER_LENGTH returns the length of a string in characters
3. OCTET_LENGTH returns the length of a string in bytes

Syntax Pattern

These three functions share a similar syntax pattern, as follows.-

```
<length function> ::=
{ BIT_LENGTH | CHAR_LENGTH | CHARACTER_LENGTH | OCTET_LENGTH } ( <value expression>
```

Example

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

New INTL Interface for Non-ASCII Character Sets

A. dos Santos Fernandes

A feature of Firebird 2 is the introduction of a new interface for international character sets. Originally described by N. Samofatov, the new interface features a number of 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 doesn't specifically included a CHARACTER SET clause to use it.

At attachment time you can 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 do not follow the rule that other charsets do 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 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 FB 1.5.X only ASCII characters are uppercased in a character set's default (binary) collation order, which 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 FB 2.0 the result is:

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

Maximum string length

In FB 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.

NONE as attachment character set

When NONE is used as the attachment (connection) character set, the sqlsubtype member of XSQLVAR stores the character set number of the read field, instead of always 0 as previously.

Enhancements for BLOBs

Content

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

Example of a Section from 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>
```

New Character Sets/Collations

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
```

Brazilian collations

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

Sort order and equality sample:

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

C
======
A
a
á
b

SQL> select * from t where c = 'â';

C
======
```

a
A
â

Drivers

New character sets and collations are implemented through dynamic libraries and installed in the server with a manifest file in the intl subdirectory. For an example, see fbintl.conf.

Not all implemented character sets and collations need to be listed in the manifest file. Only those listed are available and duplications are not loaded.

Adding More Character Sets to a Database

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

New Character Sets and Collations Implemented

ES_ES_CI_AI for ISO8859_1 Character Set

A. dos Santos Fernandes

Spanish language case- and accent-insensitive collation for ISO8859_1 character set.

KOI8-R

O. Loa, A. Karyakin

Russian language character set and dictionary collation.

KOI8-U

O. Loa, A. Karyakin

Ukrainian language character set and dictionary collation.

WIN1257_LV

O. Loa, A. Karyakin

Latvian dictionary collation.

WIN1257_LT

O. Loa, A. Karyakin

Lithuanian dictionary collation.

WIN1257_EE
O. Loa, A. Karyakin

Estonian dictionary collation.

UTF8
A. dos Santos Fernandes

Unicode 4.0 support with UTF8 character set and collations UCS_BASIC and UNICODE.

Brazilian collations
A. dos Santos Fernandes, P. H. Albanez

1. Collation PT_BR for ISO8859_character set

2. Collation WIN_PTBR for WIN1252 character set

Bosnian Collation
F. Hasovic

New Bosnian language collation BS_BA was added for WIN1250 character set.

Czech Collations
I. Prenosil, A. dos Santos Fernandes

- WIN_CZ: case-insensitive Czech language collation for WIN1250 character set
- WIN_CZ_CI_AI: case-insensitive, accent-insensitive Czech language collation for WIN1250 character set

Vietnamese Character Set
Nguyen The Phuong, A. dos Santos Fernandes

Charset WIN1258 for Vietnamese language.

Polish Collation
Jaroslaw Glowacki, A. dos Santos Fernandes

Added new collation ISO_PLK for ISO8859_2 charset (Polish language).

Character Set Bug Fixes
A. dos Santos Fernandes

The following bugs related to character sets and collations were fixed:

SF #1073212 An Order By on a big column with a COLLATE clause would terminate the server.

SF #939844 A query in a UNICODE database would throw a GDS Exception if it was longer than 263 characters.

SF #977785 Wrong character lengths were being returned from some multi-byte character sets (UTF-8, East-Asian charsets).

SF #536243 A correct result is now returned when the UPPER() function is applied to a UNICODE_FSS string.

SF #942726 UPPER did not convert aacute to Aacute for ISO8859_1

SF #544630 Some problems were reported when connecting using UNICODE. (More information to come.)

SF #540547 Some problems involving concatenation, numeric fields and character set were fixed.

Unregistered bug A query could produce different results, depending on the presence of an index, when the last character of the string was the first character of a compression pair.

Unregistered bug SUBSTRING did not work correctly with a BLOB in a character set.

Unregistered bug Pattern matching with multi-byte BLOBs was being performed in binary mode.

Unregistered bug Connecting with a multi-byte character set was unsafe if the database had columns using a different character set.

Chapter 11

Security in Firebird 2

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.

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 2.0

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 the Security Upgrade Script in the Appendix.

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.

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.

> **Caution**
>
> However, the level of Firebird security is still not satisfactory in one serious respect, so please read this section carefully before opening port 3050 to the Internet.

An important security problem with Firebird still remains unresolved: the transmission of poorly encrypted passwords "in clear" across the network. It is not possible to resolve this problem without breaking old clients.

To put it another way, a user who has set his/her password using a new secure method would be unable to use an older client to attach to the server. Taking this into account with plans to upgrade some aspects of the API in the next version, the decision was made not to change the password transmission method in Firebird 2.0.

The immediate problem can be solved easily by using any IP-tunneling software (such as ZeBeDee) to move data to and from a Firebird server, for both 1.5 and 2.0. It remains the recommended way to access your remote Firebird server across the Internet.

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 the Appendix to these notes: Security Upgrade Script.

Doing the Security Database Upgrade

To do the upgrade, follow these steps:

1. Put your old security database in some place known to you, but not in Firebird's new home directory. Keep a copy available at all times!
2. Start Firebird 2, using its new, native security2.fdb.
3. Convert your old security database to ODS11 (i.e. backup and restore it using Firebird 2.0). Without this step, running the security_database.sql script will fail!
4. Connect the restored security database as SYSDBA and run the script.
5. Stop the Firebird service.
6. Copy the upgraded database to the Firebird 2 home directory as security2.fdb.

7. Restart Firebird.

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

Nullability of RDB$PASSWD

In pre-2.0 versions of Firebird it was possible to have a user with NULL password. From v.2.0 onward, the RDB$PASSWD field in the security database is constrained as NOT NULL.

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

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

to be

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

Caution with LegacyHash

As long as you configure `LegacyHash = 1` in `firebird.conf`, Firebird's security does not work completely. To set this right, it is necessary to do as follows:

1. Change the SYSDBA password
2. Have the users change their passwords (in 2.0 each user can change his or her own password).
3. Set LegacyHash back to default value of 0, or comment it out.
4. Stop and restart Firebird for the configuration change to take effect.

Chapter 12

Command-line Utilities

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 backups of multi-file databases are 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

Content

New Switches, Changed Behaviours

V. Horsun

The new GBAK switch

```
-RECREATE_DATABASE [OVERWRITE]
```

replaces the old switch, making it harder for the unsuspecting to overwrite a database accidentally, as could occur with the shortened or long form of the now-defunct old switch:

```
-R[EPLACE_DATABASE]
```

In summary:

- gbak -R[ECREATE_DATABASE] and gbak -C[REATE_DATABASE] are now equivalent
- gbak -R[ECREATE_DATABASE] O[VERWRITE] is equivalent to the old gbak -R[EPLACE_DATABASE]

- The old gbak -R[EPLACE_DATABASE] is now -REP[LACE_DATABASE]

That is to say, now it will be necessary to include the O[VERWRITE] flag in order to have gbak restore over an existing database.

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

The following commands have been added or enhanced.

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.

SHOW SYSTEM now shows predefined UDFs

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.

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 ex-

ecuting 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
```

ISQL Bugs Fixed

SF #910430 ISQL and database dialect

fixed by C. Valderrama, B. Rodriguez Somoza

What was fixed When ISQL disconnected from a database, either by dropping it or by trying to connect to a non-existent database, it remembered the SQL dialect of the previous connection, which could lead to some inappropriate warning messages.

~ ~ ~

SF #223126 Misplaced collation when extracting metadadata with ISQL

fixed by B. Rodriguez Somoza

~ ~ ~

SF #223513 Ambiguity between tables and views

fixed by B. Rodriguez Somoza

~ ~ ~

SF #518349 ISQL SHOW mangles relationship

fixed by B. Rodriguez Somoza

~ ~ ~

Unregistered bug Possible crashes with long terminators

fixed by C. Valderrama

~ ~ ~

Unregistered bug Avoided several SQL> prompts when using the INPUT command interactively.

implemented by C. Valderrama

~ ~ ~

Unregistered bugs Some memory leaks

fixed by C. Valderrama

~ ~ ~

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
```

Chapter 13

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_srand()

F. Schlottmann-Goedde

IB_UDF_srand is now available in the IB_UDF library. Description to come.

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 FBCLIENT to be built.

Chapter 14

New Configuration Parameters and Changes

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. Horsun

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. More detail to come.

New parameter 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 15

Known Compatibility Issues

D. Yemanov

This chapter is intended as a set of alerts to those who are migrating Firebird 1.0 or 1.5 databases to Firebird 2.0. It should be studied before attempting to install any servers.

The FIREBIRD Variable

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

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

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

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

Security in Firebird 2 (All Platforms)

Be aware of the following changes that introduce incompatibilities with how your existing applications interface with Firebird's security:

Direct connections to the security database are no longer allowed

Apart from the enhancement this offers to server security, it also isolates the mechanisms of authentication from the implementation.

- User accounts can now be configured only by using the Services API or the GSEC utility.
- For backing up the security database, the Services API is now the only route. You can employ the `-se[rvice] hostname:service_mgr` switch when invoking the gbak utility for this purpose.

Non-SYSDBA users no longer can see other users' accounts in the security database

A non-privileged user can retrieve or modify only its own account and it can change its own password.

Remote attachments to the server without a login and password are now prohibited

- For attachments to Superserver, even root trying to connect locally without “localhost:” in the database file string, will be rejected by the remote interface if a correct login is not supplied.
- Embedded access without login/password works fine. On Windows, authentication is bypassed. On POSIX, the Unix user name is used to validate access to database files.

The security database is renamed to `security2.fdb`

If you upgrade an existing installation, be sure to upgrade the security database using the provided script in order to keep your existing user logins.

Before you begin the necessary alterations to commission an existing security database on the Firebird 2.0 server, you should create a gbak backup of your old security.fdb (from v.1.5) or isc4.gdb (from v.1.0) using the old server's version of gbak and then restore it using the Firebird 2.0 gbak.

> **Important**
>
> You must make sure that you restore the security database to have a page size of at least 4 Kb. The new security2.fdb will not work with a smaller page size.

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

For more details see the notes in the chapter on security, New Security Features. Also read the file `security_database.txt` in the *upgrade* directory beneath the root directory of your installation.

SQL Migration Issues

DDL

Views made updatable via triggers no longer perform direct table operations

In former versions, a naturally updatable view with triggers passed the DML operation to the underlying table and executed the triggers as well. The result was that, if you followed the official documentation and used triggers to perform a table update (inserted to, updated or deleted from the underlying table), the operation was done twice: once executing the view's trigger code and again executing the table's trigger code. This situation caused performance problems or exceptions, particularly if blobs were involved.

Now, if you define triggers for a naturally updatable view, it becomes effectively like a non-updatable view that has triggers to make it updatable, in that a DML request has to be defined on the view to make the operation on the underlying table happen, viz.

1. if the view's triggers define a DML operation on the underlying table, the operation in question is executed once and the table triggers will operate on the outcome of the view's triggers
2. if the view's triggers do not define any DML request on the underlying table then no DML operation will take place in that table

> **Important**
>
> Some existing code may depend on the assumption that requesting a DML operation on an updatable view with triggers defined would cause the said operation to occur automatically, as it does for an updatable view with no triggers. For example, this “feature” might have been used as a quick way to write records to a log table en route to the “real” update. Now, it will be necessary to adjust your view trigger code in order to make the update happen at all.

New Reserved Words (Keywords)

A number of new reserved keywords are introduced. The full list is available in the chapter New Reserved Words and Changes and also in Firebird's CVS tree in / doc/sql.extentions/README.keywords. You must ensure that your DSQL statements and procedure/trigger sources do not contain those keywords as identifiers.

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

CHECK Constraint Change

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

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

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

DML

Changed Ambiguity Rules in SQL

A. Brinkman

In summary, the changes are:

1. When an alias is present for a table, that alias, and not the table identifier, must be used to qualify columns; or no alias is used. Use of an alias makes it invalid to use the table identifier to qualify a column.

2. Columns can now be used without qualifiers in a higher scope level. The current scope level is checked first and ambiguous field checking is done at scope level.

Examples

a) 1. When an alias is present it must be used or no alias at all must be used.

This query was allowed in FB1.5 and earlier versions:

```
SELECT
  RDB$RELATIONS.RDB$RELATION_NAME
```

```
FROM RDB$RELATIONS R
```

Now, the engine will 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
```

a) 2. The next statement will now use the appropriate FieldID correctly 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**
>
> Although it is possible in Firebird to provide an alias in an update statement, many other database vendors do not support it. These SQL statement syntaxes provide better interchangeability with other SQL database products.

a) 3. This example ran incorrectly 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 rows, it would return 90 * 90 = 8100 rows, but in Firebird 2.0 it will correctly return 90 rows.

b) 1. This would fail in Firebird 1.5, but is possible in Firebird 2.0:

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

b) 2. Ambiguity checking in subqueries

This would run on Firebird 1.5 without reporting an ambiguity, but will report it in Firebird 2.0:

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

Multiple Hits to Same Column Now Illegal

It is no longer allowed to make multiple “hits” on the same column in an INSERT or UPDATE statement. Thus, a statement like

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

or

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

will be rejected in Firebird 2.n, even though it was tolerated in InterBase and previous Firebird versions.

Query Plans

Stricter validation of user-specified plans

User-specified plans are validated more strictly than they were formerly. If you encounter an exception related to plans, e.g. `Table T is not referenced in plan`, it will be necessary to inspect your procedure and trigger sources and adjust the plans to make them semantically correct.

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

Plan must refer to all tables in query

Using a plan without a reference to all tables in query is now illegal and will cause an exception. Some previous versions would accept plans with missing references, but it was a bug.

PSQL

Restrictions on assignment to context variables in triggers

- Assignments to the OLD context variables are now prohibited for every kind of trigger.
- Assignments to NEW context variables in AFTER-triggers are also prohibited.

> **Tip**
>
> If you get an unexpected error `Cannot update a read-only column` then violation of one of these restrictions will be the source of the exception.

Reference to "current of <cursor>" outside scope of loop

In Firebird 1.5 and earlier, referring to "current of <cursor>" outside the scope of the cursor loop was accepted by the PSQL parser, allowing the likelihood of run-time occurring as a result. Now, it will be rejected in the procedure or trigger definition.

NULLS are now "lowest" for sorts

NULL is now treated as the lowest possible value for ordering purposes and sets ordered on nullable criteria are sorted accordingly. Thus:

- for ascending sorts NULLs are placed at the beginning of the result set
- for descending sorts NULLs are placed at the end of the result set

> **Important**
>
> In former versions, NULLs were always on top. If you have client code or PSQL definitons that rely on the legacy NULLs placement, it will be necessary to use the NULLS FIRST option in your ORDER BY clauses.

CURRENT_TIMESTAMP now returns milliseconds by default

The context variable CURRENT_TIMESTAMP now returns milliseconds by default, while it truncated sub-seconds back to seconds in former versions. If you need to continue receiving the truncated value, you will now need to specify the required accuracy explicitly, i.e. specify `CURRENT_TIMESTAMP(0)`.

*ORDER BY <ordinal-number> now causes SELECT * expansion*

When columns are referred to by the "ordinal number" (degree) in an ORDER BY clause, when the output list uses `SELECT * FROM ...` syntax, the column list will be expanded and taken into account when determining which column the number refers to.

This means that, now, `SELECT T1.*, T2.COL FROM T1, T2 ORDER BY 2` sorts on the second column of table T1, while the previous versions sorted on T2.COL.

> **Tip**
>
> This change makes it possible to specify queries like `SELECT * FROM TAB ORDER BY 5`.

Configuration Parameters

Configuration parameter DeadThreadsCollection is deprecated

The parameter DeadThreadsCollection for Superserver in firebird.conf is deprecated and will be ignored if set. Firebird version 2 efficiently cleans up dead threads straight away.

Command-line Tools

Change to GBAK -R Semantics

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

If you actually *want* the former behaviour, you have two alternatives:

- Specify the full syntax `GBAK -REPLACE_DATABASE`. There is a new shortcut for the -REPLACE_DATABASE switch: `GBAK -REP`

  OR

- Use the new command `-R[ECREATE_DATABASE] OVERWRITE`. The `-R` shortcut now represents the -R[ECREATE_DATABASE] switch and the OVERWRITE keyword must be present in either the full or the abbreviated form.

> **Warning**
>
> If you use the full syntax, you are expected to know what this restore mode actually means and have some recovery strategy available if the backup subsequently turns out to be unrestorable.

Performance

The following changes should be noted as possible sources of performance loss:

ALL predicate may be slow

Formerly, using the ALL existence predicate could return wrong results in cases where an index was involved. To correct the bug, this version will never use an index with ALL. This change may degrade performance.

Superserver garbage collection changes

Formerly, Superserver performed only background garbage collection. By contrast, Classic performs “cooperative” GC, where multiple connections share the performance hit of GC.

Superserver's default behaviour for GC is now to combine cooperative and background modes. The new default behaviour generally guarantees better overall performance as the garbage collection is performed online, curtailing the growth of version chains under high load.

It means that some queries may be slower to start to return data if the volume of old record versions in the affected tables is especially high. ODS10 and lower databases, having ineffective garbage collection on indices, will be particularly prone to this problem.

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

Firebird API

Note the following changes affecting the API

isc_interprete is deprecated

isc_interprete() is deprecated as dangerous. Use fb_interpret() instead.

Events callback routine declaration corrected

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

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

In the Firebird 2.0 API it is:

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

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

Windows-Specific Issues

For installing, configuring and connecting to Windows servers, be aware of the following issues:

Windows Local Connection Protocol with XNet

The transport internals for the local protocol have been reimplemented (XNET instead of IPServer). With regard to the local protocol, the new client library is therefore incompatible with older servers and older client libraries are incompatible with the Firebird 2 servers.

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

Client Impersonation No Longer Works

WNET (a.k.a. NetBEUI, Named Pipes) protocol no longer performs client impersonation. For more information, refer to Change to WNET Protcol in the chapter about new features.

Interactive Option Added to instsvc.exe

D. Yemanov

The optional switch -i[nteractive] has been implemented in `instsvc.exe` to enable an interactive mode for LocalSystem services.

For v.1.5, it was required (as *Allow service to interact with desktop*) to run the local IPC protocol, as it used a windows message to connect the server. In v.2.0, it is no longer necessary and the server itself does not need this option.

However, some custom UDFs may use the Win32 messaging facilities and this option allows them to work as expected.

> **Note**
>
> `instsvc.exe` is a command-line utility for installing and uninstalling the Firebird service. It does not apply to Windows systems that do not have the ability to run services (Win9x, WinME).
>
> For detailed usage instructions, refer to the document `README.instsvc` in the `doc` directory of your Firebird installation.

Chapter 16

INSTALLATION NOTES

Please read the previous chapter, Known Compatibility Issues before you set out to install Firebird 2.0.

Windows 32-bit Installs

READ THIS FIRST!

On Windows, you have three server models to choose from: Superserver, Classic and Embedded Server. This means you have some decisions to make before installing Firebird 2.0.

- Make sure you are logged in as Administrator (doesn't apply on Win9x or ME)
- Check to make sure that there is no FIREBIRD environment variable defined that is visible to Administrator-level users or to the LocalSystem user--see the section called “The FIREBIRD Variable” at the start of the previous chapter.
- The Superserver and Classic models, as well as server tools-only and client-only, can be installed using the Windows installer application. For a full-release install, it is highly recommended to use the installer if there is one available.
- Use gbak to back up your old security.fdb (or, for a previous Firebird 1.0 installation, isc4.gdb) security database. You can restore it later as security2.fdb, using the directions in the chapter entitled New Security Features.
- If you have special settings in your existing firebird.conf (ibconfig, for Firebird 1.0) there may be some values that you want to transfer to equivalent parameters in the new firebird.conf. When you use the Windows control panel applet “Add/Remove Programs” to uninstall Firebird 1.5.x (recommended!), the uninstaller should preserve your existing firebird.conf and aliases.conf. However, it won't hurt to take the precaution of backing up these files to your home directory. The Firebird 1.0.x ibconfig file (if applicable) should certainly be backed up.

  > **Note**
  >
  > If you are upgrading from Firebird 1.0.x, go to the Downloads/Firebird Database Engine page at the Firebird website and download the Firebird 1.5.3 releasenotes for details of the correlation between settings in ibconfig and firebird.conf. Study the notes about firebird.conf to work out what can be copied directly and what parameters require new syntax.

- When reinstalling Firebird 2.0, certain configuration files in the installation directory will be preserved if you run the installer and OVERWRITTEN if you decompress a zip kit into the default location. The files are

  security.fdb
  firebird.log

firebird.conf
aliases.conf

- Each model can be installed from a zipfile. This method will be faster than the installer if you have plenty of experience installing Firebird from zipfiles. It will be highly exasperating if you are a Firebird newbie.
- It is assumed that.-
  1. you understand how your network works
  2. you understand why a client/server system needs both a server and clients
  3. you have read the rest of these release notes--or at least realise that you need to read them if something seems to have gone wrong
  4. you know to go to the firebird-support list if you get stuck. Join at http://www.yahoogroups.com/groups/firebird-support

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

- Use the existing version of GBAK to back up your database files in transportable format
- Go to your System directory and make backup copies of fbclient.dll and/or gds32.dll if you have applications that rely on finding those libraries there. You might want to name the backup "gds32.dll.fb15" or "gds32.dll.fb103" or something similarly informative; or hide it in another directory
- It might be a good idea to make a backup of the Microsoft C++ runtime, msvcp60.dll, too. The installer shouldn't overwrite your version of this file, but strange things have been known to happen.
- STOP ANY FIREBIRD OR INTERBASE SERVER THAT IS RUNNING

  The installer will try to detect if an existing version of Firebird or InterBase is installed and/or running. In a non-installer install, you are on your own!
- Provided you do not have a FIREBIRD environment variable defined, the default root location of Firebird 2.0 will be C:\Program Files\Firebird\Firebird_2_0.
- For installing Firebird as a service: if you want to make use of the secure login feature, create a "firebird service user" on the system--any name and password you like--as an ordinary user with appropriate privileges.

  You should read the document named README.instsvc.txt first. If you have a zip kit, you will find it in the /doc directory of the zipfile's root. If you don't have a zip kit available, the file won't be available until after the installation. You can read the same document at this URL: http:// firebird.cvs.sourceforge.net/viewcvs.py/firebird/firebird2/doc/README.instsvc

Naming databases on Windows

Note that the recommended extension for database files on Windows ME and XP is ".fdb" to avoid possible conflicts with "System Restore" feature of these Windows versions. Failure to address this issue on these platforms will give rise to the known problem of delay on first connection to a database whose primary file and/or secondary files are named using the ".gdb" extension that used to be the

Borland convention for suffixing InterBase database file names.

The issue is described in more detail in Other Win32 Issues at the end of the Windows installation notes.

Other Pre-installation Issues

Installing Multiple Servers

One of the design goals, since Firebird 1.5, has been to prepare the way for running multiple Firebird servers simultaneously on the same host machine. Firebird 2.0 does support this, although it is not well documented and very much requires intervention from a skilled user. A future sub-release of Firebird 2.0 will make this process far less complicated to install and manage.

If you are experienced with Firebird and have a requirement to run multiple Firebird servers side by side, please consult the Firebird 1.5.3 release notes for details of how to achieve it.

Installation of Microsoft system libraries

The problems associated with installing different versions of Microsoft system libraries are so notorious that it has acquired the name 'DLL Hell'. From the release of Windows 2000 onwards Microsoft have made it almost impossible to upgrade system dll's. To resolve this Microsoft now recommends that each application installs local copies of any system libraries that are required.

Firebird 1.5 follows this practice and places the required libraries in the \bin directory along with the server.

Installation of fbclient.dll

Since Firebird 1.5, gds32.dll is not the “native” name of the client library. It is now called fbclient.dll. Given the problems that Microsoft have had with DLL hell it wouldn't make much sense if we continued to store the Firebird client library in the system directory. Furthermore, as we want to allow multiple engines to run simultaneously we would be creating our own DLL hell if we continued the practice of using the system directory for the client library.

So, from Firebird 1.5 on, the client library resides in the \bin directory along with all the other binaries. The installer provides the option (unchecked) to copy the client to the system directory for those who have applications that require to load them from there.

Registry Key

A Registry key has been added and all Firebird 2.0 compliant applications should use this key if they need to read a Registry key to locate the correct version of Firebird that they wish to use. The new key is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Firebird Project\Firebird Server\Instances
```

Firebird will guarantee that one entry under this key always exists. It will be known as

```
"DefaultInstance"
```

and will store the path to the root directory of (yes, you've guessed it) the default installation. Those that don't care about particular installations can always use the default instance to locate the fbclient.dll.

Future versions of Firebird will see other entries under Instances. Applications will be able to enumerate the Registry entries to determine which Server instance they wish to load.

Supporting legacy applications and drivers

Traditionally, applications that use InterBase or Firebird have expected to load the gds32.dll client library from the system directory. Firebird 2.0 ships with a tool named 'instclient.exe' that can install a clone of fbclient.dll to the Windows System directory. This clone gets patched on the fly so that its file version information begins with "6.3", to provide compatibility for old applications that check the GDS32.DLL file version and can not make sense of a number string such as "2.0".

InstClient.exe Tool

This 'instclient.exe' tool can also install the FBCLIENT.DLL itself in the Windows system directory, if required. This will take care of tools or applications that need to load it from there.

The instclient.exe utility should be located in the 'bin' directory of your Firebird installation and must be run from there in a command shell.

Usage of instclient.exe:

```
instclient i[nstall] [ -f[orce] ] library
           q[uery] library
           r[emove] library
```

where library is: `fbclient | gds32`

`'-z'` can be used with any other option, prints version.

Version information and shared library counts are handled automatically. You may provide the `-f[orce]` option to override version checks.

> **Caution**
>
> If you `-f[orce]` the installation, it could break another Firebird or InterBase® version already installed. You might have to reboot the machine in order to finalize the copy.

For more details, see the document README.Win32LibraryInstallation.txt which is located in ..\doc.

Cleaning up release candidate installs

It should be noted that the installer removes fbclient.dll from the <system> directory if the file is found there. The installer also removes any deprecated HKLM\Software\Firebird* Registry keys.

Using the Win32 Firebird Installer

> **Important**
>
> Don't overlook the need to have the Microsoft® Visual C and Visual C++ runtimes (msvcrt.dll and msvcpp60.dll, respectively) present in the system directory of all Windows servers and clients, including Windows Embedded installations. For your convenience, copies of these libraries will be placed in the \bin directory of the Firebird install. However, you should check first whether later versions of these libraries are already present. Don't overwrite later versions.

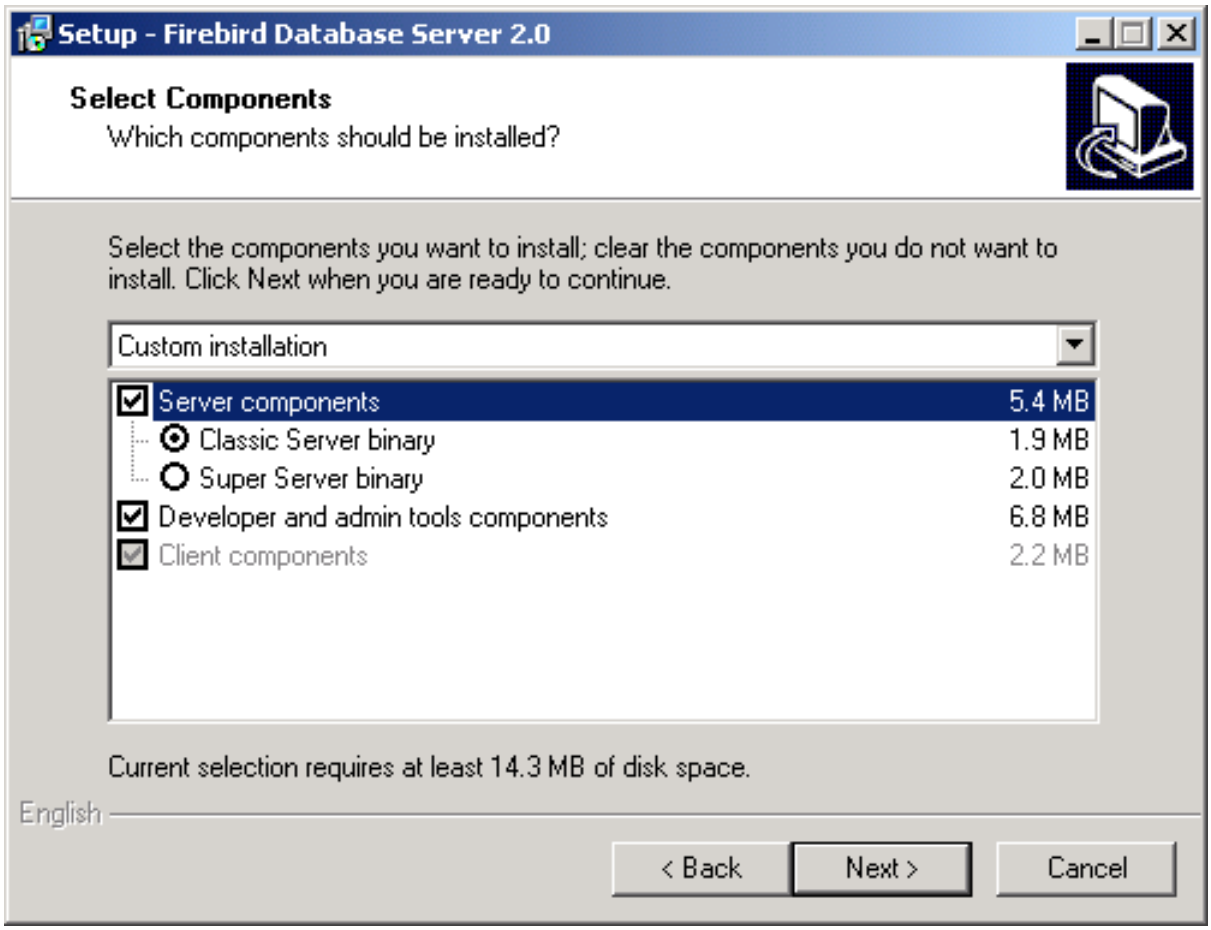
This is really the easy part: the actual install. Just run the executable and respond to the dialogs. After you have answered a few dialogs about licensing and installation notes, you should see one where you decide on the location of the Firebird root directory.

Installation (Root) directory

The installer should be showing “c:\Program Files\Firebird\Firebird_2_0” by default. If you decide not to use the default root location, browse to a location you have pre-created; or just type in the full path and let the installer find it. The path you type in doesn't have to exist: the installer will prompt you and create it if it doesn't exist.

Here you can also opt not to have the installer create Startup Menu icons by checking off the option. If you are installing on Windows 9x or WinMe, or you plan to run the server as an application in another Win32 environment, keep the icons option checked on.

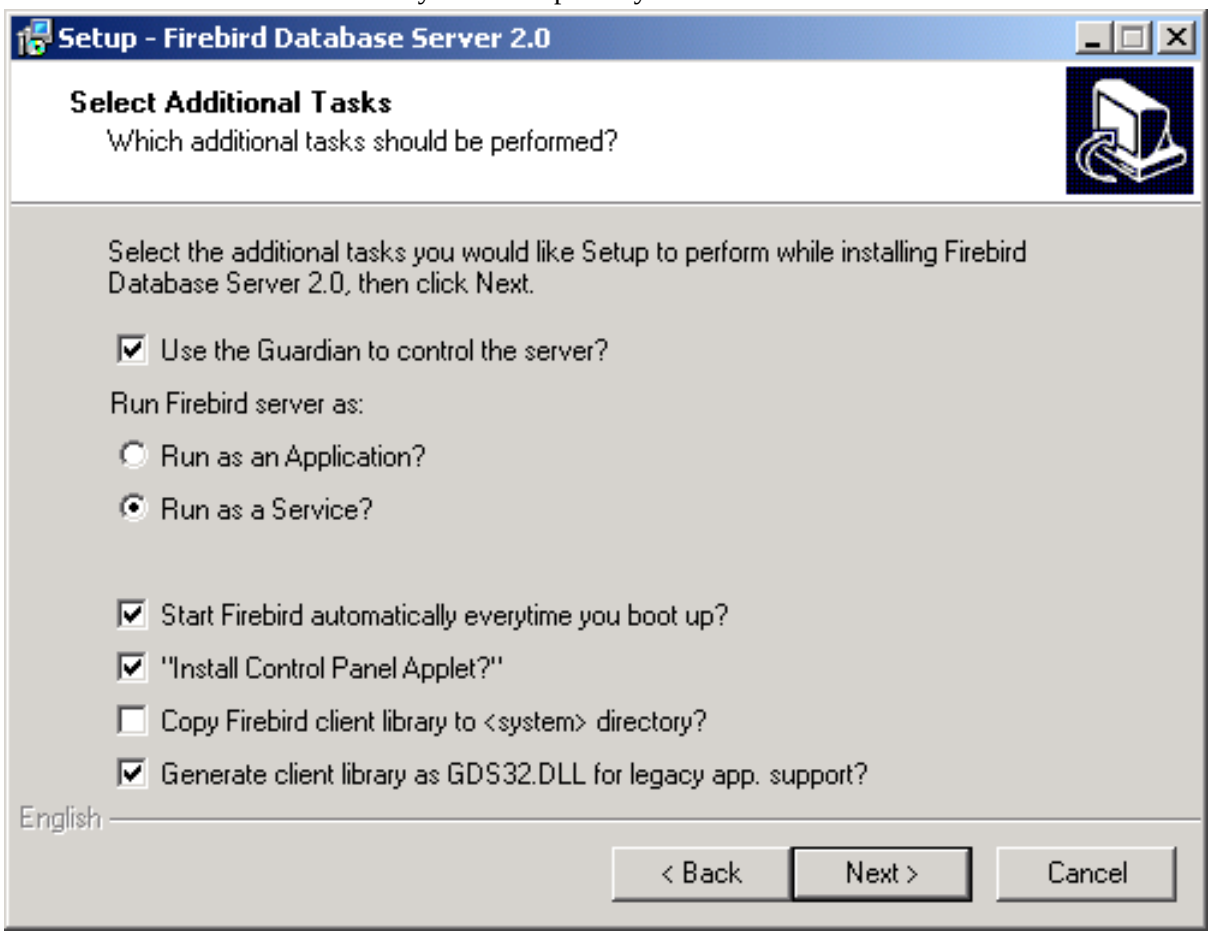
Next, you should see a screen where you choose the installation you want:



Choose the installation you want and hit the "Next" button to carry on responding to dialogs.

> **Note**
>
> If you're installing a server, you should choose Superserver (preselected by the installer) or Classic (as seen in the image above). Leave “Server components” and “Developer and admin tools components” checked on.
>
> For a client-only install, check off “Server components”, leaving “Client components” and, optionally, “Developer and admin tools components” checked on.
>
> There is also a drop-down for a custom installation which new users can safely ignore.

The next screen of interest enables you to set up how you want the server to run.



Choose the options you want, according to your choice of server model.

Use the Guardian...

Guardian is a utility than can run "over the top" of Superserver and restart it if it crashes for any reason. If you chose the Classic server, the Guardian option won't appear.

For deployment of Superserver on Win9x, WinME and WinNT 4.0, using Guardian can avoid the situation where the server stops serving and nobody can find the DBA to restart it. On other Win32 platforms, you can set the operating system to restart the service instead and not bother with the Guardian.

Service or application?

If you select to install Superserver or Classic, and your OS version supports services, you will be asked to choose whether to run Firebird as a service or as an application. Unless you have a compelling need to run the server as an application, choose service.

Manual or automatic?

With the automatic option, Firebird will start up whenever you boot the host machine. With the manual option you can start the server on demand from the Services applet in the Settings/Control Panel/ Administration Tools selection.

Use Control Panel Applet (Superserver only)

If Superserver is being installed, you will see an option to “Install Control Panel applet?”. It's a good idea to keep this as it places an applet in the Control Panel from which you can stop and [re]start the server.

Eventually, the dialogs will stop, you will press “Install” and the server will either silently start the server (if you requested it) or prompt you for permission to reboot. Reboot will be needed if the installer was unable to update a DLL due to its being already loaded when the installer started up.

Uninstallation

If you are going to uninstall Firebird, first shut down all connections to databases and then, if you are running Superserver, shut down the server. The Firebird uninstall routine (run from Add/Remove Programs in the Control Panel) preserves and renames the following key files:

preserves security2.fdb or renames it to security2.fbnnnn
preserves firebird.log
preserves firebird.conf or renames it to firebird.confnnnn
preserves aliases.conf or renames it to aliases.confnnnn

"nnnn" is the build number of the old installation.

No attempt is made to uninstall files that were not part of the original installation.

Shared files such as fbclient.dll and gds32.dll will be deleted if the share count indicates that no other application is using them.

The Registry keys that were created will be removed.

Installing Superserver from a zip kit

The installation of FB 2.0 is similiar in principle to previous versions. If you don't have a special setup program (it's distributed separately) the steps are the following:

- unzip the archive into a new directory
- change the current directory to $FIREBIRD\bin (here and below, $FIREBIRD refers to the directory where the v.2.0 files are located)
- run instreg.exe:

```
instreg.exe install
```

  It causes the installation path of the directory above to be written into the registry (HKLM\Software\Firebird Project\Firebird Server\Instances\DefaultInstance)

- if you want to register a service, also run instsvc.exe:

```
instsvc.exe install
```

- optionally, you may need to run instclient.exe to copy fbclient.dll or a specially-generated clone as gds32.dll to the OS system directory

Installing Classic Server from a zip kit

To install the CS engine, the only difference is the additional switch for instsvc.exe:

```
instsvc.exe install -classic
```

> **Important**
>
> Notice that this means that you may have only one architecture of the engine--either fbserver.exe (Superserver) or fb_inet_server.exe (the parent process for Classic)--installed as a service.
>
> The Control Panel applet is not installed with Classic--deliberately. Don't try to install and use it. The concept of terminating a service does not apply to the Classic model.

Simplified setup

If you don't need a registered service, then you may avoid running both instreg.exe and instsvc.exe. In this case you should just unzip the archive into a separate directory and run the server as an application:

```
fbserver.exe -a
```

It should treat its parent directory as the root directory in this case.

Uninstallation

To remove FB 1.5 without a Windows Uninstaller you should:

- stop the server
- run "instreg.exe remove"
- run "instsvc.exe remove"
- delete installation directory
- delete fbclient.dll and gds32.dll from the OS system directory

Other Win32 Issues

Winsock2

Firebird requires WinSock2. All Win32 platforms should have this, except for Win95. A test for

the Winsock2 library is made during install. If it is not found the install will fail. To find out how to go about upgrading, visit this link.

Windows ME and XP

Windows ME and XP (Home and Professional editions) there is a feature called *System Restore*, that causes auto-updating (backup caching?) of all files on the system having a ".gdb" suffix. The effect is to slow down InterBase/Firebird database access to a virtual standstill as the files are backed up every time an I/O operation occurs. (On XP there is no System Restore on the .NET Servers).

A file in the Windows directory of ME, c:\windows\system\filelist.xml, contains "protected file types". ".gdb" is named there. Charlie Caro originally recommended deleting the GDB extension from the "includes" section of this file. However, since then, it has been demonstrated that Win-ME might be rebuilding this list. In XP, it is not possible to edit filelist.xml at all.

On ME, the permanent workarounds suggested are one of:

- use FDB (Firebird DB) as the extension for your primary database files--RECOMMENDED
- move databases to C:\My Documents, which is ignored by System Restore
- switch off System Restore entirely (consult Windows doc for instructions).

On Windows XP Home and Professional editions you can move your databases to a separate partition and set System Restore to exclude that volume.

Windows XP uses smart copy, so the overhead seen in Windows ME may be less of an issue on XP, for smaller files at least. For larger files (e.g. Firebird database files, natch!) there doesn't seem to be a better answer as long as you have ".gdb" files located in the general filesystem.

Updated Notes for Windows Embedded

Some changes between Firebird 1.5 and Firebird 2.0 mean the existing docs are slightly out-of-date. For convenience, the following are the updated notes.

The embedded server is a fully functional server linked as a dynamic library (fbembed.dll). It has exactly the same features as the usual Superserver and exports the standard Firebird API entrypoints.

The embedded server acts as a true local server for a single client accessing databases on a local machine. It can also act as a remote gateway that redirects all network calls to other hosts, just as the regular client library does.

Registry

The Firebird Registry entries are ignored. The root directory of the embedded server is the same directory as the one where the embedded library binary is located.

Database Access

Client access can be only via the local (XNET) protocol, i.e. NOT a TCP/IP local loopback connection string that includes the server name “localhost” or the IP address 127.0.0.1. The embedded server supports only the local connect to an absolute database file path without a server name.

The client program gets exclusive access to the database file after successful connect.

Authentication and Security

The security database (security2.fdb) is not used in connecting to the embedded server. Hence it is not required. Any user is able to attach to any database. Since both the server and the client run in the same address space, security becomes just an agreement between the accessor and the accessed, which can be easily compromised.

> **Note**
>
> SQL privileges are still checked and enforced. Users that are assigned privileges in a Firebird database are not dependent on the existence of the user in the security database.

Compatibility

You may run any number of applications with the embedded server without any conflicts. Having a full Firebird or InterBase server running on the same machine is not a problem either.

However, be aware that you cannot access a single database from a number of embedded servers simultaneously, regardless of whether they be embedded or full servers. An embedded server has the SuperServer architecture and hence exclusively locks attached databases.

Installing an Embedded Server Application

Application Root

Just copy fbembed.dll, icudt30.dll, icuin30.dll and icuuc30.dll into the directory with your application exectable.

You should also copy firebird.msg and firebird.conf (if necessary) to the same directory.

> **Note**
>
> You will need firebird.conf only if it is necessary to set some non-default configuration parameter for the embedded server.

If **external libraries** are required for your application, such as INTL support (fbintl.dll and fbintl.conf) or UDF libraries, create subdirectories beneath the application root for them, emulating the Firebird server ones, e.g. /intl or /udf, respectively.

Rename fbembed.dll

Rename fbembed.dll to either fbclient.dll or gds32.dll, according to which is required by your database connectivity software.

Start your application

Now start your application and it will use the embedded server as a both a client library and a server and will be able to access local datasases via the XNET network emulation protocol.

Installation Structure Examples

```
c:\my_app\app.exe
c:\my_app\gds32.dll
c:\my_app\ib_util.dll
```

```
c:\my_app\icudt30.dll
c:\my_app\icuin30.dll
c:\my_app\icuuc30.dll
c:\my_app\firebird.conf
c:\my_app\firebird.msg
c:\my_app\intl\fbintl.dll
c:\my_app\intl\fbintl.conf
c:\my_app\udf\fbudf.dll
```

Suppose you want to place the Firebird files (excluding the renamed fbembed.dll) in another directory. In that case, you need to modify your firebird.conf and set RootDirectory to the Firebird directory tree that is parent to the Firebird files.

Example

```
c:\my_app\app.exe
c:\my_app\gds32.dll
c:\my_app\ib_util.dll
c:\my_app\icudt30.dll
c:\my_app\icuin30.dll
c:\my_app\icuuc30.dll
c:\my_app\firebird.conf
d:\fb\firebird.msg
d:\fb\intl\fbintl.dll
d:\fb\intl\fbintl.conf
d:\fb\udf\fbudf.dll
```

In firebird.conf:

```
RootDirectory = d:\fb
```

POSIX Platforms

(Originally by Mark O'Donohue, revised for 2.0)

The Firebird server comes in two forms, Classic, which runs as a service, and SuperServer, which runs as a background daemon. Classic is the more traditional UNIX service, while Superserver uses threads, rather than processes. For the user just starting out with Firebird, either will do, although the Classic server is likely to prove a better platform for initially experimenting with Firebird.

READ THIS FIRST

- You will need to be root user to install Firebird.
- Installation on Linuxen requires a glibc package installed that is equal to or greater than glibc-2.2.5 and a libstdc++.so equal to or greater than libstdc++-5.0.

> **Note**
>
> Some higher distros, e.g. Mandriva 10.2, might fail to complete the password-setting script at the end of a Classic installation because the local client, libfbembed.so needs libstdc++.so.5 and the installed version of this runtime is missing. An "impure" solution that should solve the immediate problem, at least, is to Google for “compat-libstdc++” and find one that was built for your kernel version.
>
> The *pure* solution of course is to compile Firebird on the same system that you are going to run it on! This might be necessary, anyway, if the compatibility runtimes cause problems with other applications.

- Do not try to use *rpm --update* to bring any existing Firebird package installation up to date. *The Firebird packages do not support it.*
- If you are installing Superserver on a Linux that supports the “new POSIX threading library ” (NPTL) then choose the NPTL build of Firebird. Most distros with the 2.6 kernel are built with NPTL enabled; some with later 2.4 kernels also enabled it, but it may be wise to prepare to revert to the regular build and set up to export the LD_ASSUME_KERNEL=2.2.5 variable if the 2.4 implementation of the NPTL causes problems. Details for doing this follow below.
- 64-bit builds are available for both Classic and Superserver. These should be installed only on a 64-bit Linux system. NPTL support is native on 64-bit Linux.

Setting Linux to Use the Old Threading Model

If the NPTL causes problems for SuperServer and locally compiled programs, including utilities such as gbak throwing a *Broken Pipe* error, you can try to solve the problem by forcing Linux to use the old threading model. To fix.-

1. In /etc/init.d/firebird

   ```
   LD_ASSUME_KERNEL=2.2.5
   export LD_ASSUME_KERNEL
   ```

   That takes care of the server instance.

2. You need to have the `LD_ASSUME_KERNEL` environment variable set up within the local environment as well, so add the following to /etc/profile, to ensure every user picks it up for the command line utilities.

   after

   ```
   HISTSIZE=1000
   ```

   add

   ```
   LD_ASSUME_KERNEL=2.2.5
   ```

   On the following line, export it (this is all in one line):

```
export PATH USER LOGNAME MAIL HOSTNAME
    HISTSIZE INPUT_RC LD_ASSUME_KERNEL
```

Installing on Linux

The following instructions describe the Classic installation. For installation of Superserver the "CS" in the package name is replaced by "SS". For example, the package `FirebirdCS-2.0.0-nnnnn.i686.rpm` is replaced by `FirebirdSS-2.0.0-nnnnn.i686.rpm`.

> **Note**
>
> For those who, in the past, have had trouble installing Firebird on Slackware, the good news is that the installers in this version *do* include Slackware support.

Log in as root, or open a root shell. In the example filenames, replace *nnnnn* with the build number of the kit you actually have.

RPM Installer

For the RPM installer, type:

```
$rpm -ivh FirebirdCS-2.0.0-nnnnn.i686.rpm
```

Installing the Tarball

To install the tarball, place the ".tar.gz" file and type:

```
$tar -xzf FirebirdCS-2.0.0-nnnnn.tar.gz
$cd FirebirdCS-2.0.0-nnnnn.i686
$./install.sh
```

What the Linux install scripts will do

The Linux install scripts will

1. Attempt to stop any currently running server
2. Add the user 'firebird' and the group 'firebird' if they do not already exist.
3. Install the software into the directory /opt/firebird and create links for libraries in /usr/lib and header files in /usr/include
4. Automatically add gds_db for port 3050 to /etc/services if the entry does not already exist
5. Automatically add localhost.localdomain and HOSTNAME to /etc/gds_hosts.equiv
6.
   a. SuperServer only installs a /etc/rc.d/init.d/firebird server start script.
   b. Classic server installs a /etc/xinetd.d/firebird start script or, for older inetd systems, adds an

entry to the /etc/inetd file

7. Specific to SuSE, a new rcfirebird link is created in /usr/bin for the init.d script and an /etc/rc.config Firebird entry is created.

8. Starts the server/service. Firebird should start automatically in runlevel 2, 3 or 5

9. Generates and sets a new random SYSDBA password and stores it in the file /opt/firebird/SYSDBA.password.

10. Adds an entry to aliases.conf for the sample database, employee.fdb.

Testing your Linux installation

Step 1 - Accessing a database

In a shell:

```
$cd /opt/firebird/bin
$./isql -user sysdba -password <password>[1]

SQL>connect localhost:employee.fdb /* this is an aliased path */

SQL>select * from sales;
SQL>select rdb$relation_name from rdb$relations;
SQL>help;

SQL>quit;
```

> **Note**
>
> [1]A password has been generated for you on installation. It can be obtained from the /opt/firebird/SYSDBA.password file, located in the Firebird root directory.

Step 2 - Creating a database

The Firebird server runs by default as the user 'firebird'. While this has always been the recommended configuration, the previous default was for the server to run as 'root' user. When running as root user, the server had quite wide-ranging ability to read, create and delete database files anywhere on the POSIX filesystem.

For security reasons, the service should have a more limited ability to read/delete and create files.

While the new configuration is better from a security perspective, it requires some special considerations to be taken into account for creating new databases:

1. the user 'firebird' has to have write permission to the directory in which you want to create the database.

2. the recommended value of the DatabaseAccess attribute in the /opt/firebird/firebird.conf file should be set to None, to permit access only through entries in the aliases.conf file.

3. use entries in aliases.conf to abstract users from the physical locations of databases. More notes

on aliases are to be found in the topic Database File Aliasing.

Procedures for creating a new database can vary with different configurations but the following configuration and steps are recommended:

1. If a directory that is owned by the user 'firebird' does not exist, then change to root user and create the directory:

```
$su - root
$mkdir -p /var/firebird
$chown firebird:firebird /var/firebird
```

2. Create a new physical database and set up an alias entry to point to it. As root or firebird user, run the following script:

```
$cd /opt/firebird/bin
$./createAliasDB.sh test.fdb /var/firebird/test.fdb
```

(Usage is: createAliasDB.sh <dbname> <pathtodb>)

3. As an alternative (for step 2) the steps in the createAliasDB.sh script can be performed manually by:

```
$vi /opt/firebird/aliases.conf
```

and add the line at the end of the file:

```
test.fdb /var/firebird/test.fdb
```

4. Then create the database:

```
$/opt/firebird/bin/isql -u sysdba -p <password>
SQL>create database 'localhost:test.fdb';
SQL>quit;
```

5. If the *DatabaseAccess* value in /opt/firebird/firebird.conf is set to Full or a restricted path value (for example: DatabaseAccess=/var/firebird) another alternative to step 2 is to create the physical database file directly, using the absolute path with the filename:

```
$/opt/firebird/bin/isql -u sysdba -p <password>
SQL>create database '/var/firebird/test.fdb';
SQL>quit;
```

If you use this configuration, the database file can also be directly accessed without an entry in the aliases file:

```
$/opt/firebird/bin/isql -u sysdba -p <password>
SQL>connect '/var/firebird/test.fdb';
SQL>quit;
```

Utility Scripts

In addition to the standard install files the following scripts are provided in the bin directory of this release.-

changeDBAPassword.sh
: Change the Firebird SYSDBA user password. For Superserver, this script will change the init script /etc/rc.d/init.d/firebird to use the new password as well.

createAliasDB.sh
: Usage: `createAliasDB.sh <dbname> <dbpath>`

  This script creates a new physical database and adds an entry in the aliases.conf file.

fb_config
: A script that can be used in makefiles to generate the required include paths and lib include directives for the installed version of Firebird. `fb_config -help` will give a complete list of options.

changeGdsLibraryCompatibleLink.sh
: Classic only-Change the client library link for libgds.so between the multithreaded libfbclient.so and the single threaded libfbembed.so library that allows an embedded direct open of the db file. For compatibility with previous installs, libgds.so by default points to libfbembed.so.

Linux Server Tips

"Embedded" or direct access to database files

The Classic install offers an "embedded" mode of access that allows programs to open database files directly. To operate in this mode, a database-enabled user requires privileged access to some of the Firebird configuration and status files.

Now that it is the 'firebird' user (not root) that is the default user to run the software, you need to know how to get a user into the firebird group to enable direct access to databases. It is documented in the readme notes, but the following steps should get you where you need to be.

To add a user (e.g. skywalker) to the firebird group, the root user needs to do:

```
$ usermod -G firebird skywalker
```

Next time 'skywalker' logs on, he can start working with firebird databases.

To list the groups that a user belongs to, type the following at the command line:

```
$ groups
```

Uninstalling on Linux

If you need to uninstall, do it as root user. The following examples use Classic server but the same

holds true for SuperServer by replacing the CS with SS.

Uninstalling an RPM package

For rpm packages:

```
$rpm -e FirebirdCS-2.0.0
```

Uninstalling a tarball installation

for the .tar.gz install:

```
$/opt/firebird/bin/uninstall.sh
```

Solaris

Install Firebird Classic & SuperServer on Solaris 2.7 Sparc, not currently available. Please refer older releasenotes as a reference to 2.0 installations.

MacOS X

Install Firebird Classic on MacOS X / Darwin, not currently available. Please refer to older release-notes as a reference to 2.0 installations.

FreeBSD

Not currently available. Please refer to older releasenotes as a reference to 2.0 installations.

Debian

Not currently available. Please refer to the relevant pages at the Debian site for your Debian version and Firebird 2.0 build.

Chapter 17

Bugs Fixed

The following bugs present in Firebird 1.5 were fixed. Note that, in many cases, the bug-fixes were backported to Firebird 1.5.x sub-releases.

> **Important**
>
> For the convenience of testers, RC1 bugs found and fixed are listed at the end of this chapter.

General Engine Bugs

Not registered Assignments to columns deleted by a concurrent transaction were being improperly allowed.

fixed by D. Yemanov

~ ~ ~

Not registered Error "invalid transaction handle" would be thrown when calling isc_array_lookup_bounds() from multiple threads.

fixed by D. Yemanov

~ ~ ~

Not registered Heavy concurrent load could cause index data corruption.

fixed by V. Horsun

~ ~ ~

SF #1446987 BLOBs could appear to be damaged during operations in PSQL, causing a "BLOB not found" error.

fixed by V. Horsun

~ ~ ~

SF #1434147 Bugs with COUNT (DISTINCT XXXX) when XXXX was a high integer.

fixed by V. Horsun

~ ~ ~

SF #1435997 A bug was causing a close database error -901 on the embedded server.

fixed by D. Yemanov

~ ~ ~

SF #1436066 Adding an index during database activity could cause logical errors in structure that GFIX would detect.

fixed by V. Horsun

~ ~ ~

Not registered A few types of subqueries were being wrongly treated as variant, causing performance issues.

fixed by D. Yemanov

~ ~ ~

Not registered Previously, the Transaction ID would silently (and dangerously) overflow. Now it will throw a consistency check when it reaches the limit (which is still 2^31).

fixed by V. Horsun

~ ~ ~

Not registered Read committed transactions would block garbage collection unnecessarily.

fixed by V. Horsun

~ ~ ~

Not registered The ALL predicate could return wrong results.

fixed by D. Yemanov

~ ~ ~

SF #1404157 DFW was not ready for RECREATE TABLE/VIEW

fixed by D. Yemanov

~ ~ ~

Not registered Restored the code which replaces ROLLBACK with COMMIT if a transaction has not modified any data.

fixed by D. Yemanov

~ ~ ~

Not registered There were some bugs producing wrong statistics:

- with relation/index data longer than 2^32 bytes
- when the average index key length rounded to an integer value

fixed by V. Horsun

~ ~ ~

Not registered Attaching with the isc_dpb_no_garbage_collect option was forcing a sweep.

fixed by V. Horsun

~ ~ ~

Not registered The system transaction was being reported as dead.

fixed by A. dos Santos Fernandes, V. Horsun

~ ~ ~

Not registered The server would lock up after an unsuccessful attach to the security database.

fixed by D. Yemanov, C. Valderrama

~ ~ ~

SF #1076858 Source of possible corruption in Classic server.

fixed by V. Horsun

~ ~ ~

SF #1116809 Incorrect data type conversion.

fixed by A. dos Santos Fernandes

~ ~ ~

SF #1111570 Problem dropping a table having a check constraint referencing more than one column.

fixed by C. Valderrama

~ ~ ~

Not registered Usage of an invalid index in an explicit plan caused garbage to be shown in the error message instead of the rejected index name.

fixed by C. Valderrama

~ ~ ~

SF #543106 Bug with ALL keyword. MORE INFO REQUIRED.

fixed by D. Yemanov

~ ~ ~

Not registered System users "AUTHENTICATOR" and "SWEEPER" were lost, causing "SQL SERVER" to be reported instead.

fixed by A. Peshkov

~ ~ ~

Not registered Don't rollback prepared 2PC sub-transaction. (Description needs clarifying, Vlad!)

fixed by V. Horsun

~ ~ ~

Not registered Memory consumption became exorbitant when blobs were converted from strings during request processing. For example, the problem would appear when running a script with a series of statements like

```
insert into t(a,b)
   values(N, <literal_string>);
```

when b was blob and the engine was performing the conversion internally.

fixed by N. Samofatov

~ ~ ~

Not registered Materialization of BLOBs was not invalidating temporary BLOB IDs soon enough.

A blob is created as an orphan. This blob has a blob id of {0,slot}. It is volatile, meaning that, if the connection terminates, it will become eligible for garbage collection. Once a blob is assigned to field in a table, it is said to be materialized. If the transaction that did the assignment commits, the blob has an anchor in the table and will be considered permanent. Its blob id is {relation_id,slot}.

In situations where internal code is referencing the blob by its old, volatile blob id, the references are "routed" to the materialized blob, until the session is closed.

fixed by N. Samofatov

Solution Now, the references to a volatile blob are checked and, when there are no more references to it, it is invalidated.

~ ~ ~

Not registered Conversion from string to blob had a memory leak.

fixed by N. Samofatov

~ ~ ~

SF #750664 Issues with read-only databases and transactions.

fixed by N. Samofatov

~ ~ ~

Not registered When one classic process dropped a foreign key and another process was trying to delete master record, the error 'partner index not found' would be thrown.

fixed by V. Horsun

~ ~ ~

Various server bugs

1. eliminated redundant attempts to get an exclusive database lock during shutdown
2. corrected inaccurate timeout counting
3. database lock was not being released after bringing database online in the exclusive mode

4. removed a 5 sec timeout when bringing database online in the shared mode

fixed by D. Yemanov

~ ~ ~

SF #1186607 Foreign key relation VARCHAR <-> INT should not have caused an exception.

fixed by V. Horsun

~ ~ ~

SF #1211325 Fixed problems with BLOBs in external tables.

fixed by V. Horsun

~ ~ ~

Not registered After an attempt to "create view v(c1) as select 1 from v" all clones of the system request would remain active forever.

fixed by A. Peshkov

~ ~ ~

SF #1191006 Use of WHERE params in SUM would return incorrect results.

fixed by A. Brinkman

~ ~ ~

SF #750662 Fixed a bug involving multiple declaration of blob filters.

fixed by D. Yemanov

~ ~ ~

SF #743679 FIRST / SKIP was not as well implemented as it could be.

fixed by D. Yemanov

~ ~ ~

Not registered CPU load would rise to 100% when an I/O error caused a rollover to a non-existent shadow.

fixed by D. Yemanov

~ ~ ~

Not registered "Cannot find record fragment" bugcheck could occur during garbage collection on the system tables.

fixed by V. Horsun

~ ~ ~

SF #1211328 Error reporting cited maximum BLOB size wrongly.

fixed by D. Yemanov

~ ~ ~

SF #1292007 Duplicated field names in INSERT and UPDATE statements were getting through.

fixed by C. Valderrama

~ ~ ~

Not registered The SQL string was being stored truncated within the RDB$*_SOURCE columns in some cases

fixed by D. Yemanov

~ ~ ~

Not registered Broken implementation of the MATCHES predicate in GDML

fixed by D. Yemanov

~ ~ ~

SF bug #1404215 Column dependencies were not being stored for views.

fixed by D. Yemanov

~ ~ ~

SF bug #1191206 A few constraint issues.

fixed by D. Yemanov

~ ~ ~

SF bug #609538 Alter Index on a Foreign Key index should cause an exception and it did, but the error message was not appropriate.

fixed by D. Yemanov

~ ~ ~

SF bug #1175157 An error in the thread scheduler was causing the server to lock up.

fixed by V. Horsun

~ ~ ~

Not registered

1. Improper thread data operations were occurring during the protocol port cleanup
2. Transaction rollback and attachment cleanup for broken TCP connections was faulty

fixed by V. Horsun, D. Yemanov

~ ~ ~

Not registered A wrong error message was decoded when firebird.msg was missing or outdated.

fixed by D. Yemanov

~ ~ ~

Not registered Buffer overflows inside the BLR->ASCII blob filter were causing memory corruption and server crashes.

fixed by D. Yemanov

~ ~ ~

Not registered A successful status vector could be reported to the user after a failed DDL operation.

fixed by V. Horsun

~ ~ ~

Not registered Threading issues in the DSQL metadata cache were causing unexpected “invalid transaction handle” errors under load.

fixed by D. Yemanov

~ ~ ~

Not registered Wrong results would be returned by the division operation after DDL changes.

Example

```
create table test(fld numeric(18, 2));
insert into test (fld) values (1);
commit;
alter table test alter fld type numeric(18,3);
select fld/3 from test; -- returns 0.033 instead of expected 0.333
```

fixed by D. Yemanov

~ ~ ~

SF #1184099 Incorrect padding was exhibited when using character set OCTETS.

fixed by C. Valderrama, A. dos Santos Fernandes

~ ~ ~

Not registered Unexpected errors were occurring because of improperly handled dead record versions created by the system transaction during DDL operations.

fixed by A. Harrison

~ ~ ~

SF #223060 Processing of the GREATER-THAN operator was too slow.

fixed by V. Horsun

~ ~ ~

Not registered CHECK constraints were not SQL-compliant with regard to the handling of

NULL. Until now, if NULL were to be allowed, it had to be specified explicitly in the constraint definition. Under the standard, NULL is allowed unless explicitly constrained by NOT NULL or CHECK (.. IS NOT NULL).

Example of Problem

The following definition now allows NULL in DEPTNO, where previously it did not:

```
CHECK (DEPTNO IN (10, 20, 30))
```

fixed by P. Ruizendaal, D. Yemanov

~ ~ ~

Not registered It was possible to create a primary key constraint on a column consisting of NULLs.

Example of Problem

```
create table bug (f1 int not null, f2 int not null);
insert into bug (f1, f2) values (1, 1);
commit;
alter table bug add pk int not null primary key;
```

fixed by V. Horsun

~ ~ ~

SF #1334034 REVOKE was damaging the ACL (Access Control List).

fixed by D. Yemanov

~ ~ ~

Services Manager

Not registered Incorrect encryption of password when the Services Manager was invoked by the Embedded client.

fixed by A. Peshkov

~ ~ ~

GFix Bugs

SF #1242106 Shutdown bugs:

1. Incorrect commit instead of rollback during shutdown
2. Crash or bugcheck during SuperServer shutdown with active attachments

fixed by D. Yemanov

~ ~ ~

Not registered Crash occurred in service gfix code when it tried to reattach to a currently unavailable database. Since a service cannot interact with the end-user, an endless loop leads to overflowing the service buffer and causing a crash as a result.

fixed by V. Horsun

~ ~ ~

DSQL Bugs

SF #1408079 The parser was not validating string literal markers.

fixed by C. Valderrama

~ ~ ~

Not registered The engine would fail to parse the SQL ROLE keyword properly.

fixed by C. Valderrama

~ ~ ~

Not registered EXECUTE PROCEDURE did not check SQL permissions at the prepare stage.

fixed by D. Yemanov

~ ~ ~

SF #217042 Weird SQL constructions are not always properly validated.

Partly fixed by C. Valderrama

~ ~ ~

SF #1108909 View could be created without rights on a table name like "a b"

fixed by C. Valderrama

~ ~ ~

SF #512975 Clear embedded spaces and CR+LF before DEFAULT clauses when storing them in system tables

Implemented by C. Valderrama

~ ~ ~

SF #910423 Anomaly with ALTER TABLE altering a column's type to VARCHAR, when determining valid length of the string.

```
SQL> CREATE TABLE tab ( i INTEGER );
SQL> INSERT INTO tab VALUES (2000000000);
SQL> COMMIT;
```

```
SQL> ALTER TABLE tab ALTER i TYPE VARCHAR(5);
Statement failed, SQLCODE = -607
unsuccessful metadata update
-New size specified for column I must be at least 11 characters.
```

i.e., it would need potentially 10 characters for the numerals and one for the negative sign.

```
SQL> ALTER TABLE tab ALTER i TYPE VARCHAR(9);
```

This command should fail with the same error, but it did not, which could later lead to unreadable data:

```
SQL> SELECT * FROM tab;
I
=========
Statement failed, SQLCODE = -413
conversion error from string "2000000000"
```

fixed by C. Valderrama

~ ~ ~

Not registered There were some rounding problems in date/time arithmetic.

fixed by N. Samofatov

~ ~ ~

Not registered Line numbers in DSQL parser were being miscounted when multi-line literals and identifiers were used.

fixed by N. Samofatov

~ ~ ~

SF #784121 Some expressions in outer join conditions were causing problems.

fixed by C. Valderrama

~ ~ ~

Not registered There were some dialect- specific arithmetic bugs:

Dialect 1

1. '1.5' / '0.5' did not work
2. avg ('1.5') did not work
3. 5 * '1.5' produced an INT result instead of DOUBLE PRECISION
4. sum ('1.5') produced a NUMERIC(15, 2) result instead of DOUBLE PRECISION
5. - '1.5' did not work

Dialect 3

- '1.5' * '0.5' and '1.5' / '0.5' were not forbidden, but they should have been.

fixed by D. Yemanov

~ ~ ~

SF #1250150 There was a situation where a procedure could not be dropped.

fixed by V. Horsun

~ ~ ~

SF #1238104 Internal sweep report was incorrect.

fixed by C. Valderrama

~ ~ ~

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

fixed by D. Yemanov

~ ~ ~

SF #1292106 ORDER BY with FOR UPDATE WITH LOCK would trash the index.

fixed by D. Yemanov

~ ~ ~

SF #1368741 UPPER() was returning wrong results.

fixed by A. dos Santos Fernandes

~ ~ ~

PSQL Bugs

SF #1422471 A memory leak was exhibited in EXECUTE STATEMENT.

fixed by A. Peshkov

~ ~ ~

Not registered ROW_COUNT was getting cleared after SUSPEND execution.

fixed by D. Yemanov

~ ~ ~

SF #1124720 Problem with "FOR EXECUTE STATEMENT ... DO SUSPEND;"

fixed by A. Peshkov

~ ~ ~

Not registered Memory leakage was occurring when selectable stored procedures were called from PSQL or in subqueries.

fixed by N. Samofatov

~ ~ ~

Not registered The wrong error would be reported when non-active contexts were accessed in multi-action triggers.

fixed by D. Yemanov

~ ~ ~

Not registered An internal error was reported when attempting to pass/return blobs to/from string functions inside PSQL.

fixed by D. Yemanov

~ ~ ~

Crash Conditions

Not registered A crash could occur if some bad client passed more than the supported number of remote protocol versions.

fixed by A. Karyakin, A. Peshkov

~ ~ ~

Not registered An AV could occur when the server was configured to use TCP packets as large as 32 Kb.

fixed by C. Valderrama, A. Peshkov

~ ~ ~

Not registered Server would crash if a positioned UPDATE/DELETE executed via DSQL was referencing a cursor that had already been released.

fixed by V. Horsun

~ ~ ~

Not registered Certain DDL actions could crash the server.

Example of a problem action

```
alter table rdb$relations
  add rdb$garbage varchar(30);
```

fixed by J. Starkey

~ ~ ~

Not registered An overflow in the plan buffer would cause the server to crash.

fixed by D. Yemanov

~ ~ ~

Not registered Possible server lockup/crash when 'RELEASE SAVEPOINT xxx ONLY' syntax is used or when existing savepoint name is reused in transaction context

fixed by N. Samofatov

~ ~ ~

Not registered Rare client crashes caused by improperly cleaned XDR packets.

fixed by D. Yemanov

~ ~ ~

Not registered Server crash during SuperServer shutdown

fixed by A. Peshkov

~ ~ ~

SF #1057538 The server would crash if the output parameter of a UDF was not the last parameter.

fixed by C. Valderrama

~ ~ ~

Not registered A number of possible server crash conditions had been reported by Valgrind.

fixed by N. Samofatov

~ ~ ~

Not registered Server would crash when a wrong type or domain name was specified when changing the data type for a column.

fixed by N. Samofatov

~ ~ ~

Not registered Incorrect accounting of attachment pointers used inside the lock structure was causing the server to crash.

fixed by N. Samofatov

~ ~ ~

Not registered In v.1.5, random crashes would occur during a restore.

fixed by J. Starkey

~ ~ ~

Not registered Crash/lock-up with multiple calls of isc_dsql_prepare for a single statement.

fixed by N. Samofatov

~ ~ ~

Not registered Server would crash when the system year was set too high or too low.

fixed by D. Yemanov

~ ~ ~

Not registered Server would crash when the stream number exceeded the limit.

fixed by D. Yemanov

~ ~ ~

Not registered Server would crash when outer aggregation was performed and explicit plans were used in subqueries.

fixed by D. Yemanov

~ ~ ~

Not registered DECLARE FILTER would cause the server to crash.

fixed by A. Peshkov

~ ~ ~

Not registered The server would crash when a PLAN for a VIEW was specified but no table alias was given.

fixed by V. Horsun

~ ~ ~

Not registered Server would crash during the table metadata scan in some cases.

fixed by D. Yemanov

~ ~ ~

Not registered Server would crash when too big a key was specified for an index retrieval.

fixed by D. Yemanov

~ ~ ~

Not registered Server would crash when manipulating input DPB due to memory corruption in Parameter Blocks management.

fixed by C. Valderrama

~ ~ ~

Not registered Server would crash when attempting to restore a database backup with corrupted VARCHAR data.

fixed by D. Yemanov

~ ~ ~

Remote Interface Bugs

Not registered A TCP/IP buffer size larger than 32 Kb was not being processed correctly.

fixed by A. Peshkov

~ ~ ~

Not registered The NO_NAGLE option was working improperly.

fixed by F. Polizo, A. Peshkov

~ ~ ~

Not registered NO_NAGLE and KEEPALIVE socket options were not enabled for CS builds.

fixed by D. Yemanov

~ ~ ~

SF #1385092 A TCP/IP connection would appear to freeze the Superserver if it was disconnected abnormally while a large packet, e.g. a BLOB or a large SQL request, was being passed across the interface.

This was a long-standing InterBase/Firebird bug in the implementation of the protocol layer for Superserver on Windows. Borland invented two different thread management strategies: one for TCP/IP and one for the other protocols that only Windows supports, i.e. Named Pipes (sometimes referred to as “NetBEUI”) and the IPServer local connection. This bug occurred only with TCP/IP connections.

For TCP/IP, a multiplexing loop (main server loop), which is common for all ports, receives API packets from clients, creates requests and sends them to threads for processing. When it detects an incoming packet, it starts to receive it from the port.

Before this fix, it needed the entire API packet to come at once. However, in the course of converting a packet to a request (done by the XDR protocol), in cases where the size of the API packet happened to be greater than that of the network packet, the server had to wait for the next network packet from the port.

At this point, ports were being scanned for incoming packets only by calculating (timeout - interval since last packet received) for each port in the loop. If the next packet from a particular port did not come, for example because of an unplugged jack, the only way to interrupt this receive and allow the main server loop to carry on processing the other ports was to wait for the keepalive TCP timeout to elapse on the abandoned connection. Given that the default keepalive value is two hours, it would appear that the Superserver was “hung”.

fixed by A. Peshkov

~ ~ ~

SF #1260310 Nessus vulnerability scanning could cause the server to drop connections.

fixed by A. Peshkov

~ ~ ~

SF #1065511 Clients on Windows XP SP2 were slow connecting to a Linux server.

fixed by N. Samofatov

~ ~ ~

SF #1065511 Clients on Windows XP SP2 were slow connecting to a Linux server.

fixed by N. Samofatov

~ ~ ~

SF #571026 INET/INET_connect: gethostbyname was not working properly.

fixed by D. Yemanov

~ ~ ~

SF #223058 Multi-hop server capability was broken.

fixed by D. Yemanov

~ ~ ~

Not registered Fixed memory leak from connection pool in isc_database_info.

fixed by N. Samofatov

~ ~ ~

Not registered Database aliases were not working in WNET.

fixed by D. Yemanov

~ ~ ~

Not registered Client would crash while disconnecting with an active event listener.

fixed by D. Yemanov

~ ~ ~

Not registered The client library would not react to environment variables being set via SetEnvironmentVariable().

fixed by C. Valderrama

~ ~ ~

Indexing & Optimization

SF #459059D Index breaks = ANY result. MORE INFO REQUIRED.

fixed by N. Samofatov

~ ~ ~

Not registered Ambiguous queries were still possible under some conditions.

fixed by A. Brinkman

~ ~ ~

SF #735720 SELECT ... STARTING WITH :v was wrong when :v = ''

fixed by A. Brinkman

~ ~ ~

Not registered There were issues with negative dates, i.e. those below Julian date [zero], when stored in indices.

fixed by A. Brinkman

~ ~ ~

SF #1211354 Redundant evaluations were occurring in COALESCE.

fixed by A. Brinkman

~ ~ ~

Not registered Error "index key too big" would occur when creating a descending index.

fixed by V. Horsun

~ ~ ~

SF #1242982 Bug in compound index key mangling.

fixed by A. Brinkman

~ ~ ~

Vulnerabilities

SF #1466193 Semaphore array`s permissions in fb_lock_mgr were 0666 - i.e., anyone could lock them and block all subsequent queries.

fixed by A. Peshkov

~ ~ ~

Not registered Possible buffer overflow in WNET.

fixed by A. Peshkov

~ ~ ~

Not registered Several buffer overflows were fixed.

fixed by A. Peshkov

~ ~ ~

SF #1155520 Fixed a vulnerability that could make it possible for a user who was neither SYSDBA nor owner to create a database that would overwrite an existing database.

fixed by A. dos Santos Fernandes

~ ~ ~

ISQL Bugs

SF #781610 Comments in ISQL using '--' were causing problems.

fixed by J. Bellardo, B. Rodriguez Samoza

~ ~ ~

Not registered ISQL_disconnect_database was overwriting the Quiet flag permanently.

fixed by M. Penchev, C. Valderrama

~ ~ ~

SF #1208932 SHOW GRANT did not distinguish object types.

fixed by C. Valderrama

~ ~ ~

SF #494981 Bad exception report.

fixed by C. Valderrama

~ ~ ~

SF #450404 ISQL would uppercase role in the command line.

fixed by C. Valderrama

~ ~ ~

Various, not registered

1. Fix for the -b (Bail On Error) option when SQL commands are issued and no db connection exists yet.

2. Applied Miroslav Penchev's patch for bug with -Q always returning 1 to the operating system, discovered by Ivan Prenosil.

fixed by M. Penchev, C. Valderrama

~ ~ ~

Not registered Metadata extraction for triggers, check constraints and views with check option was wrong.

fixed by C. Valderrama, D. Yemanov

~ ~ ~

International Character Set Bugs

SF #1016040 Missing external libraries would cause an engine exception.

fixed by A. dos Santos Fernandes

~ ~ ~

Not registered

1. Charset/collation issues for expression-based view columns
2. Lost charset/collation for local PSQL variables

fixed by D. Yemanov

~ ~ ~

Not registered Comparisons between strings in NONE and another character set would cause an error.

fixed by D. Yemanov, A. dos Santos Fernandes

~ ~ ~

SF #1244126 There was a problem updating some text BLOBs when connected with character set NONE.

fixed by A. dos Santos Fernandes

~ ~ ~

SF #1242379 Applying a collation could change a VARCHAR's length

fixed by A. dos Santos Fernandes

~ ~ ~

SQL Privileges

Not registered Permissions were not being checked for view columns.

fixed by D. Yemanov

~ ~ ~

Not registered Privileges granted to procedures/triggers/views were being preserved after the object had been dropped.

fixed by D. Yemanov

~ ~ ~

Not registered Column-level SQL privileges were being preserved after the affected column was dropped.

fixed by D. Yemanov

~ ~ ~

SF #223128 SYSDBA could grant non-existent roles

fixed by D. Yemanov

~ ~ ~

UDF Bugs

Not registered There were thread safety issues in datetime functions of the FBUDF library.

fixed by C. Valderrama

~ ~ ~

Not registered The UDF AddMonth() in the UDF library FBUDF had a bug that displayed itself when the calculation rolled the month past the end of the year.

fixed by C. Valderrama

~ ~ ~

Not registered Diagnostics when a UDF module was missing/unusable needed improvement.

fixed by A. Peshkov

~ ~ ~

Not registered There were some problems with the mapping of UDF arguments to parameters.

fixed by N. Samofatov

~ ~ ~

Not registered UDF arguments were being prepared/optimized twice.

fixed by D. Yemanov

~ ~ ~

SF #544132, #728839 Nulls handling in UDFs was causing problems.

fixed by C. Valderrama

~ ~ ~

Not registered UDF access checking was incorrect.

fixed by D. Yemanov

~ ~ ~

GBak

Not registered There were issues with restoring if indexes used in explicit plans inside PSQL code had been dropped.

fixed by A. dos Santos Fernandes

~ ~ ~

Not registered GBAK could not restore a database containing broken foreign keys.

Now, the restore continues to run, the user gets a diagnostic indicating which FK caused the problem. The affected index becomes inactive and, after restore, the database is left in shutdown state.

fixed by A. Peshkov

~ ~ ~

Not registered GBAK would stall when used via the Services Manager and an invalid command line was passed.

fixed by V. Horsun

~ ~ ~

Not registered A computed column of a blob or array type would zero values in the first column of the table being restored.

fixed by D. Yemanov

~ ~ ~

Not registered Fixed some backup issues with stream BLOBs that caused them to be truncated under some conditions.

fixed by N. Samofatov

~ ~ ~

Not registered Interdependent views caused problems during the restore process.

fixed by A. Brinkman

~ ~ ~

SF #750659 If you want to start a fresh db, you should be able to restore a backup done with the metadata-only option. Generator values were resisting metadata-only backup and retaining latest values from the live database, instead of resetting the generators to zero.

fixed by C. Valderrama, D. Yemanov

~ ~ ~

SF #908319 In v.1.5, wrong error messages would appear when using gbak with service_mgr.

fixed by V. Horsun

~ ~ ~

SF #1122344 gbak -kill option would drop an existing shadow.

fixed by D. Yemanov

~ ~ ~

Not registered gbak was adding garbage bytes to the SPB when called in the -se[rvice_mgr] mode.

fixed by A. dos Santos Fernandes, C. Valderrama, V. Horsun

~ ~ ~

GPre

SF #504978 GPRE variable names were being truncated.

fixed by C. Valderrama

~ ~ ~

SF #527677 GPRE "ANSI85 compatible COBOL" switch was broken.

fixed by C. Valderrama

~ ~ ~

SF #1103666 GPRE was using inconsistent lengths

fixed by C. Valderrama

~ ~ ~

SF #1103670 GPRE would invalidate a quoted cursor name after it was opened.

fixed by C. Valderrama

~ ~ ~

SF #1103683 GPRE was not checking the length of the DB alias.

fixed by C. Valderrama

~ ~ ~

SF #1103740 GPRE did not detect duplicate quoted cursor names

fixed by C. Valderrama

~ ~ ~

Not registered GPRE could not generate more than 32,000 identifiers.

fixed by A. Harrison

~ ~ ~

GSTAT

Not registered Error output by GSTAT on Windows 32 was incorrect.

fixed by C. Valderrama

~ ~ ~

fb_lock_print

Not registered fb_lock_print could fail, with an exception message "*the requested operation cannot be performed on a file with a user-mapped section open.*"

fixed by V. Horsun

~ ~ ~

Linux Installs

SF #1011401 The start/stop script was breaking halt/reboot on Slackware.

by A. Peshkov

~ ~ ~

Code Clean-up

(Not a bug) -L[ocal] command-line switch for SS on Win32 is gone

by D. Yemanov

~ ~ ~

Assorted clean-up

- Extensive, ongoing code cleanup and style standardization
- Broken write-ahead logging (WAL) and journalling code is fully cleaned out

by C. Valderrama

~ ~ ~

Bugs Introduced in RC2

Not registered GSTAT and NBAK were exhibiting an unnecessary requirement for firebird.conf on Win32.

fixed by D. Yemanov

~ ~ ~

Not registered The server could crash with complex joins or derived tables and grouping.

fixed by G. Sergeev

~ ~ ~

Not registered Server could lock up when an exception occurred during evaluation of an expression index.

fixed by V. Horsun

~ ~ ~

Not registered START[S | ING] WITH was not working properly on an indexed column with character set UTF8.

fixed by A. Dos Santos Fernandes

~ ~ ~

Not registered Views WITH CHECK OPTION could not be restored.

fixed by D. Yemanov

~ ~ ~

Not registered Server would crash occasionally when an explicit plan wass present in a stored procedure.

fixed by D. Yemanov

~ ~ ~

Not registered NBACKUP utility was missing in the Linux SS packages.

fixed by A. Peshkov

~ ~ ~

Not registered A sub-optimal execution path could be chosen if an explicit plan was presented for a complex inner join.

fixed by D. Yemanov

~ ~ ~

Not registered Expression: " CONTAINING " wrongly returned FALSE.

fixed by A. Dos Santos Fernandes

~ ~ ~

Not registered The ODS11 optimizer was returning wrong results ALL predicates.

fixed by D. Yemanov

~ ~ ~

Not registered The SPB tags were being decoded wrongly.

fixed by A. Peshkov

~ ~ ~

Chapter 18

Appendix to Firebird 2 Release Notes

Security Upgrade Script

A. Peshkov

```
/* 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.