Firebird 2.0 Release Notes

Helen Borrie (Collator/Editor)

8 November 2005 - Document version 0200_43 - for Firebird 2.0 Beta 01 Release

Firebird 2.0 Release Notes

8 November 2005 - Document version 0200_43 - for Firebird 2.0 Beta 01 Release

Helen Borrie (Collator/Editor)

Table of Contents

List of Figures

Chapter 1: General Notes

These Notes

...have gaps, for which we apologise. They will be updated as the beta cycle proceeds.

ALERT

The on-disk structure (ODS) of the beta build is wholly incompatible with the ODS as it was in the foregoing alphas. This means that any databases created with Alpha 1, 2 or 3 will not be usable under beta. (Yes, you were warned!)

Note that this also means that an attempt to install this beta into a directory that has or had an alpha install, with the security database preserved, will fail with an unreadable security database. (The Win32 uninstall, for example, preserves the security database.)

Back up your Alpha databases *before* installing beta if you wish to continue using them.

If you unwisely neglected to check these notes before ripping in and over-installing, you will have to do a "rewind" in order to get back to the state where you can do the backup. That is to say, uninstall the beta, hunt around for the alpha, re-install it, back up and then re-install the beta.

Once again, we remind you that the software in this distribution of Firebird 2 is *a beta version*. Test it till it chokes, but do not put it into production use and do not try it out on any databases that you care about!

All new changes and new features are subject to further change and/or withdrawal in subsequent beta releases, leading up to final release. Do not assume that databases created by or upgraded to the on-disk structure of this beta will be upwardly compatible with subsequent test builds/releases.

Bug Reporting and Support

The aim of this beta is to find 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 Field Test 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

XNET is now used as the default local protocol for Windows and is supported also for connecting to a Classic server. More information to come.

Change to WNET ("NetBEUI") Protocol

WNET (a.k.a. NetBEUI) protocol no longer performs client impersonation. More information to come.

Reworking of Garbage Collection

V. Horsun

New GC thread implementation and combined cooperative + background activity. More information to come.

Storing Databases on Raw Devices

E. Kunze, N. Samofatov

You can now store databases on raw devices 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. More information to come.

Reworking of Constraint Checking

V. Horsun

More precise checks for PK/UK/FK constraints. More information to come.

Lock Timeout for WAIT Transactions

A. Karyakin, D. Yemanov

Added lock timeouts for WAIT transactions (see new TPB value isc_tpb_lock_timeout). More information to come.

New Implementation of String Search Operators

N. Samofatov

1. The operators now work correctly with BLOBs
2. Pattern matching now uses a single-pass Knuth-Morris-Pratt algorithm
3. The engine no longer crashes when NULL is used as ESCAPE character for LIKE

More information to come.

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. More information to come.

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. *(Coming in Beta 2.)*

Run-time Checking for Concatenation Overflow

D. Yemanov

Compile-time checking for concatenation overflow has been replaced by run-time checking. More information to come.

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

Experimental Support for 64-bit Platforms

A. Peshkov, D. Yemanov

Details to come.

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)

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

The ROLLBACK RETAIN statement is now supported in DSQL. More information to come.

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) Extensions

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.

API Identifies Client Version

N. Samofatov

C/C++ client interface version FB_API_VER is defined as 20 for Firebird 2.0 in ibase.h. More information to come.

Improved Services API

D. Yemanov

Services are now executed as threads rather than processes on some threadable CS builds.

ODS (On-Disk Structure) Changes

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

New ODS Number

Firebird 2 Beta 1 creates databases with an ODS (On-Disk Structure) version of 11.

Size limit for exception messages increased

Maximum size of exception messages raised from 78 to 1021 bytes. (V. Horsun)

New Description Field for Generators

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

New Description Field for SQL Roles

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

“ODS Type” Recognition

Introduced a concept of ODS type to distinguish between InterBase and Firebird databases. (N. Samofatov)

Smarter DSQL Error Reporting

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

Other

This list is not complete. To be updated.

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. More information to come.

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. More information to come.

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 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 Implementation

D. Yemanov

UNION DISTINCT is now allowed as a synonym for simple UNION, in accordance with theSQL-99 specification. More information to come.

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
V. Horsun

```
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
```

Built-in Function SUBSTRING() Enhanced
O. Loa, D. Yemanov

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

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 Equivalence Rule Relaxed

D. Yemanov

NULL can now be treated as a value in an equivalence test without returning a syntax error. You may now specify A = NULL, B > NULL, etc. (all of them evaluate to FALSE). More information to come.

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.

More information to come.

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. More information to come.

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. More information to come.

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 to come.

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
```

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 re-

turn 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
```

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
CLOSE
OPEN
```

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.

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

More information to come.

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

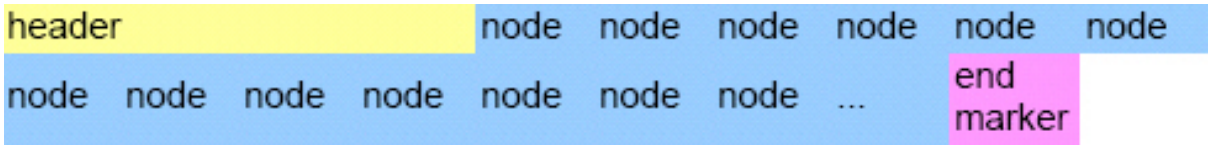
Per-segment selectivity information is now available to the optimizer, opening more possibilities for clever access path decisions. More information to come.

Firebird Index Structure from ODS11 Onward

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];
};
```

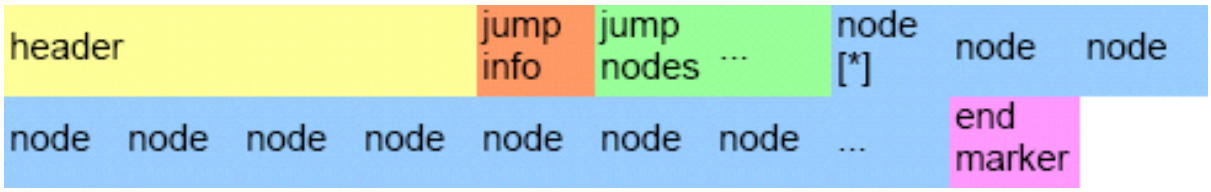
end marker = END_BUCKET or END_LEVEL

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 on the jump-nodes self is also prefix compression done.

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)

header (34)				
52 (2)	256 (2)	2 (1)	30 (2)	0 (1)
2 (1)	260 (2)	FI (2)	1 (1)	1 (1)
514 (2)	U (1)	0 (1)	1 (1)	0 (1)
A (1)	...			
2 (1)	6 (1)	21386 (3)	REBIRD (6)	...
2 (1)	2 (1)	1294 (2)	EL (2)	...

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.

Buffer Cache Improvements

O. Loa, D. Yemanov

- Better choice of streams order in joins and better index usage in general
- Much faster algorithms to process the dirty pages tree
- Increased maximum page cache size to 128K pages (2GB for 16K page size)

More information to come.

Optimizer Improvements

Content

For All Databases

Content

Faster Evaluation of IN() and OR

O. Loa

Constant IN predicate or multiple OR booleans are now evaluated faster. More information to come.

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

Distribute HAVING clause conjunctions to the WHERE clause when possible. More info to come.

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

Segment-level Selectivities are Used

Info to come.

Better Support for IS NULL

Info to come.

Better Support for STARTING WITH

Info to come.

Matching of Both OR and AND Nodes to Indexes

Info to come.

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 to come, using a non-ASCII character set.

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. Server writes to the log when conflicts exist.

More information and test scenario to come.

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: PT_BR/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
á
```

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.

After being installed in the server, 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

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.

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

Vulnerabilities have been closed

A. Peshkov, C. Valderrama

Several known vulnerabilities in the API have been closed.

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.

Some Protection from Hacking

Given the 8-byte maximum length of the traditional Firebird password, the hacker had a reasonable chance to break into the firebird installation by way of a brute-force attack. Version 2.0 has some protection from this. After too many attempts to access the server using a wrong password, the authentication process is locked for a period, minimizing the opportunity for a hacker to find the correct pass-

word in time.

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**
>
> The upgrade script may be excluded from the Beta 1 distributions due to tagging issues. The script is also in the `../src/misc/upgrade/v2` directory of the firebird2 CVS tree at Sourceforge, in the **T2_0_0_Beta1** tagged branch.
>
> 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. Connect to your old security database as SYSDBA and run the script.
4. Stop the Firebird service.
5. Copy the upgraded database to the Firebird 2 home directory.
6. Open firebird.conf and set the parameter LegacyHash to 1 (remembering to erase the "#" comment marker). TAKE NOTE OF THE CAUTION BELOW!
7. Restart Firebird.

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

> **Caution**
>
> 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
-U <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
```

> **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 -R[EPLACE_DATABASE] switch and makes it harder for the unsuspecting to overwrite a database accidentally.

- 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]

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.
- The line number of the failure is not yet known. It has been a private test feature for some years but needs more work to be included in the official isql.
- DML errors will be caught when being prepared or executed, depending on the type of error.
- 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.
- ven 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

A. dos Santos Fernandes

SQLDA_DISPLAY command shows the input SQLDA parameters of INSERTs, UPDATEs and DELETEs.

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 switched -e for echoing commands, -o to send the output to a file and -m to merge the error output to the same file. This way, you could observe the commands isql executed and the errors if they exist. The script continued executing to the end. The server only gives a line number related to the single command (statement) that it's executing, for some DSQL failures. For other errors, you only know the statement caused problems.

With the addition of -b for bail as described in (1), the user is given the power to tell isql to stop executing scripts when an error happens, but you still need to echo the commands to the output file to discover which statement caused the failure.

Now, the ability to signal a script-related line number of a failure enables the user to go to the script directly and find the offending statement. When the server provides line and column information, you will be told the exact line in the script that caused the problem. When the server only indicates a failure, you will be told the starting line of the statement that caused the failure, 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.

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 nt 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()

A. dos Santos Fernandes

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';
```

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 temporarily configure Firebird 2's new security to run with your old passwords in an upgraded security database (security.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.

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.

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.

DeadThreadsCollection is no longer used

A. Peshkov

The DeadThreadsCollection parameter is no longer used at all. Dead threads are now efficiently re-

leased "on the fly", making configuration unnecessary. Firebird 2.0 silently ignores this parameter.

Chapter 15: Installation and Compatibility Notes

Mostly unwritten so far!

Known Compatibility Issues

D. Yemanov

These issues are to be collated and published with the Beta 2 release.

All Platforms

Information to come in Beta 2 notes.

GSEC in Firebird 2

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. Remote client connection directly to the security database completely impossible--by design!

To find out more, read the chapter devoted to the New Security Features.

Windows-Specific Issues

Information to come in Beta 2 notes.

Windows Local Connection Protocol with XNet

Information to come in Beta 2.

Client Impersonation No Longer Works

WNET (a.k.a. NetBEUI, Named Pipes) protocol no longer performs client impersonation. Information to come in Beta 2 notes.

Installation

Content still to come.

Windows

Interactive Option Added to instsvc.exe

D. Yemanov

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

> **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.

POSIX

Content still to come.

Chapter 16: 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 those testing this beta after testing the alpha 3 release, bug-fixes done since alpha 3 can be found in the document `WhatsNew` in the Firebird 2.0 `doc` directory.

General Engine Bugs

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

~ ~ ~

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

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 Embed spaces and CR+LF before DEFAULT

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

~ ~ ~

PSQL Bugs

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 EXECUTE STATEMENT had a memory leak.

fixed by A. Peshkov

~ ~ ~

Crash Conditions

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

~ ~ ~

Remote Interface Bugs

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

~ ~ ~

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 dates below Julian date [zero?] 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

Not registered Several buffer overflows were fixed.

fixed by A. Peshkov

~ ~ ~

Not registered A few internal buffer overflows are 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

~ ~ ~

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 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 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. Valderram

~ ~ ~

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 GBAK could not restore a database containing broken foreign keys.

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

~ ~ ~

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

~ ~ ~

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

~ ~ ~

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

~ ~ ~

Chapter 17: Appendix to Firebird 2 Release Notes

Security Upgrade Script

A. Peshkov

```
/* Script security_database.sql, Revision 1.2 tagged T2_0_0_Beta1
 *
 *  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,

  /* 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;
```