Firebird® Version 2.0

Release Notes v.200.02 Initial Alpha

20 March 2005

CONTENTS

General Notes

The software in this early distribution of Firebird 2 is **an alpha 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 alpha and beta releases, leading up to final release. Do not assume that databases created by or upgraded to the on-disk structure of this alpha will be upwardly compatible with subsequent test builds/releases.

Bug Reporting and Support

The aim of this alpha 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 a 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 alpha 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

New Features

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.

Win32 local protocol is re-implemented D. Yemanov

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.

Garbage Collection has been reworked V. Horsun

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

Store 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.

Classic now supports full Services API 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.

Constraint checking logic has been reworked V. Horsun

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

Lock timeouts 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.

Re-implementation of LIKE/CONTAINING/STARTING WITH 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.

Updatable views logic has been reworked D. Yemanov

This solves problems with views that are implicitly updatable, but still have update triggers. More information to come.

New database shutdown modes N. Samofatov

Single-user and full shutdown modes are implemented. More information to come.

ODS Changes Various contributors

- Firebird 2 Alpha creates databases with an ODS (On-Disk Structure) version of 11.
- Maximum size of exception messages raised from 78 to 1021 bytes. (V. Horsun)
- Added RDB$DESCRIPTION to RDB$GENERATORS, so now you can include description text when creating generators. (C. Valderrama)
- Added RDB$DESCRIPTION and RDB$SYSTEM_FLAG to RDB$ROLES to allow description text and to flag user-defined roles, respectively. (C. Valderrama)
- Introduced a concept of ODS type to be able to distinguish between InterBase databases, Firebird databases, databases created by debug builds of Firebird and private forks. (N. Samofatov)
- More information to come.

UDF Enhancements C. Valderrama

- Ability to signal SQL NULL via a NULL pointer (more information to come).
- 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.

More information...

Changes to WNET protocol D. Yemanov

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

More effective checking for concatenation overflow O. Loa, D. Yemanov

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

Changes to synchronization 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

---

Miscellaneous

- 64-bit platform support, including detection of on-disk structure. (N. Samofatov)
- Introduced 40-bit (64-bit internally) record numbers to avoid ~30GB table size limit. (N. Samofatov)
- BUGCHECK log messages now include file name and line number. (A. Brinkman)
- Thread-safe and signal-safe debug logging facilities have been implemented. (N. Samofatov)
- Routines that print out various internal structures (DSQL node tree, BLR, DYN, etc) have been updated. (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. (N. Samofatov)
- More information to come.

API Changes in Firebird 2

Extended isc_dsql_info() API call D. Yemanov

The function call isc_dsql_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.

Data Definition Language (DDL)

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 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 notes about NEXT VALUE FOR.

D. Yemanov

REVOKE ADMIN OPTION FROM

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

However, there's a second form that involves roles. Instead of specifying the same rights for many users (soon it becomes a maintenance nightmare) 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.

Changed view updates logic D. Yemanov

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

Data Types

BLOB SUB_TYPE BINARY C. Valderrama

Introduced as a synonym for SUB_TYPE 0.

Data Manipulation Language (DML)

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 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
  ```

ROWS syntax

D. Yemanov

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

UNION DISTINCT syntax enabled

D. Yemanov

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

New DISTINCT equivalence predicate treats (NULL=NULL) as True

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';
```

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 can now be a value for syntactic purposes

D. Yemanov

You may now specify A = NULL, B > NULL, etc. (all of them evaluate to FALSE). More information to come.

View specification language extended

D. Yemanov

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

CROSS JOIN implemented

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
```

More information to come.

Subqueries and INSERT statements can now take union sets

D. Yemanov

SELECT specifications used in subqueries and in INSERT INTO SELECT.. statements can now specify a UNION set. More information to come.

Improved type coercion logic 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

1. Let DTS be the set of data types over which we must determine the final result data type.
2. All of the data types in DTS shall be comparable.
3. 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 :: 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 datetime data type, then every data type in DTS shall be a datetime data type having the same datetime 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.

UPDATE and DELETE statement syntax extended

O. Loa

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

Context variable ROW_COUNT returns select count

D. Yemanov

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

N. Samofatov

Get context variables via system functions

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.

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 is a namespace that offers access to session-specific user-defined variables. You can define and set values for variables with any name in this context.
- USER_TRANSACTION is a namespace offering similar possibilities for individual transactions.
- The SYSTEM namespace provides read-only access to the following variables:

Variable Name	Value
NETWORK_PROTOCOL	The network protocol used by client to connect. Currently-used values: "TCPv4", "WNET", "XNET" and NULL
CLIENT_ADDRESS	The wire protocol address of the remote client, represented as a string. The value is an IP address in form "xxx.xxx.xxx.xxx" for TCPv4 protocol; the local process ID for XNET protocol; and NULL for any other protocol.
DB_NAME	Canonical name of the current database. It is either the alias name (if connection via file names is disallowed DatabaseAccess = NONE) or, otherwise, the fully expanded database file name.
ISOLATION_LEVEL	The isolation level of the current transaction. The returned value will be one of "READ COMMITTED", "CONSISTENCY", "SNAPSHOT"
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

```
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;

execute procedure set_context('skidder', 1);

insert into journal(jrn_id) values(0);

commit;
```

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

CURRENT_TIMESTAMP and 'NOW' now return milliseconds

D. Yemanov

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

Built-in function IIF() added

O. Loa

The function

```
   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.

GROUP BY arbitrary expressions

A. Brinkman

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)
```

Nulls ordering changed

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.

Improvements in 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 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

a. 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.
b. 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.
c. 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.

DSQL parsing of table aliases is stricter

A. Brinkman

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

a. 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.
b. 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.

   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
```

   a. 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.

b. 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
```

Improved GROUP BY and ORDER BY clauses

A. Brinkman

Column aliases are now allowed in both these clauses.

Examples:

1.
```
SELECT RDB$RELATION_ID AS ID
FROM RDB$RELATIONS
ORDER BY ID
```
2.
```
SELECT RDB$RELATION_NAME AS ID, COUNT(*)
FROM RDB$RELATION_FIELDS
GROUP BY ID
```

Improved ORDER BY clause

A. Brinkman

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

Example

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

NEXT VALUE FOR expression

D. Yemanov

Added SQL-99 compliant NEXT VALUE FOR 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');
```

Points to 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 be never used in insert/update statements, as it produces a high risk of uniqueness violations in a concurrent environment.

Stored Procedure Language (PSQL)

Explicit cursors in PSQL

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  [,  ...];
    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
```

- 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 ... ;

CREATE PROCEDURE P1 (X INTEGER = 123)
RETURNS (Y INTEGER)
AS
BEGIN
  Y = X;
  SUSPEND;
END;
COMMIT;

 SELECT * FROM P1;

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

         123

EXECUTE PROCEDURE P1;

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


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

SELECT * FROM P2;

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

         123

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

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.

D. Yemanov

LEAVE <label> syntax support

New LEAVE syntax now allows PSQL loops now 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 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. More information to come.

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 '!';

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

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.

New Reserved Words & 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.

Added as reserved words

CLOSE	CROSS	FETCH
OPEN	ROWS	

Changed from non-reserved words to reserved

USING

Added as non-reserved words

BACKUP *	BLOCK *	DIFFERENCE *
IIF *	NEXT	RESTART
SCALAR_ARRAY *	SEQUENCE	

Changed from reserved words to non-reserved

ACTION	CASCADE	FREE_IT *
RESTRICT	ROLE	TYPE
WEEKDAY *	YEARDAY *	

No longer reserved words

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

Indexing Enhancements

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)
```

Points to 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).

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.

More effective index compression A. Brinkman

Information to come.

Per-segment index selectivity maintenance 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.

Optimizations

Optimizer improvements

A. Brinkman,
D. Yemanov

- Distribute HAVING clause conjunctions to the WHERE clause when possible
- Distribute UNION conjunctions to the inner streams when possible
- Improved cross join and merge/sort handling
- Better optimization of mixed AND/OR predicates
- Let's choose a reasonable join order for intermixed inner and outer joins
- MERGE PLAN may now be generated for joins using equality comparsion on expressions
- Better logic regarding unique indices handling
- Improved logic for OR expressions

Available in ODS 11.0 only:

- Usage of segment-level selectivities
- Better support for IS NULL
- Better support for STARTING WITH
- Matching both OR and AND nodes to indices
- Better cost estimations and hence better join orders
- Allowed indexed order (navigational walk) for outer joins

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, Improvements in user-specified query plans.

Buffer cache improvements

O. Loa,
D. Yemanov

- 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.

Faster evaluation of IN and OR

O. Loa

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

Security

Summary of Changes

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

Users can now modify their own passwords.

GSEC now uses the Services API

A. Peshkov

Non-server access to security database is rejected

A. Peshkov

The server now rejects any access to security2.fdb except through the Services Manager.

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.

API Vulnerabilities closed

C. Valderrama, A. Peshkov

Several known vulnerabilities in the API have been closed. More information to come.

Details of the Security Changes in Firebird 2.0

A. Peshkov

Several changes have been implemented to improve security. Aspects that have received special attention include:

- 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

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 Hashing Algorithm - SHA-1

The hashing algorithm selected for Firebird 2.0 is SHA-1. Data stored in the PASSWORD field of the USERS table in the security database contains two parts - a random number, used as salt for calculating this particular hash and the hash itself. It is calculated as SHA1 (salt || username || password)).

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 USERS.PASSWORD 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.

GSEC in Firebird 2

One of the problems addressed during the security review was gsec, the utility for maintaining user profiles and authentication passwords. Though nobody can change data in the security database without knowing the correct password, the old gsec is relatively easy to use. It will write the bad old DES hash in USERS.PASSWORD field and, if THE LegacyHash parameter IN firebird.conf is set to 0, as it should be, once the security database is upgraded, it becomes impossible to log in to the server. 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.

- The structure of security database was changed. In general, now it contains a patch by Ivan Prenosil, enabling any user to change his/her own password, with some minor differences.
    - 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 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.
- The syntax for invoking GSEC has changed.

  Because GSEC now uses the Services API, the switch

  ```
  -database <security database name>
  ```

  has been removed. In its place is a new switch,

  ```
  -server <server to manage>
  ```

  was added. For most purposes, the new switch is much more convenient: it is not necessary to supply the exact path and name of the security database on a remote server in order to manage it.

  <u>Examples</u>

  TCP/IP:

  ```
  gsec -user sysdba -password masterkey -server firebird.company.com:
  ```

  (Notice the colon suffixed to the hostname argument of the server switch.)

Win32 Named Pipes:

```
  gsec -user sysdba -password masterkey -server \\nt_srv\
```

(Notice the Named Pipes terminator suffixed to the hostname argument of the server switch.)

- 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 password 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.

==============================

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 ../misc/upgrade sub-directory of your Firebird server installation. (It is currently in the ../src/misc/upgrade/v2 directory of the firebird2 CVS tree at Sourceforge.)

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 have LegacyHash = 1, 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.

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 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 firebird.conf. Classic supports only cooperative. More detail to come.

UsePriorityScheduler — A. Peshkov

Setting this parameter to zero disables switching of thread priorities completely. It affects only the Win32 SuperServer.

DeadThreadsCollection is no longer used — A. Peshkov

Dead threads are now efficiently released "on the fly", making configuration unnecessary. Firebird 2.0 silently ignores this parameter.

Utilities

On-line incremental backup

N. Samofatov

Fast, on-line, page-level incremental backup facilities have been implemented. More information to come.

ISQL improvements

Various developers

- 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. (D. Ivanov, C. Valderrama)

  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's 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.

  <u>Some features</u>

  - 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 SET BAIL [ON | OFF] command. As it's the case with other SET commands, simply using SET BAIL will toggle the state between activated and deactivated. Using SET will display the state of the switch among many others.

- Even if BAIL is activated, it doesn't mean it will change isql behavior. An additional requirement should be met: the session should be non-interactive. A non-interactive session happens when the user calls isql in batch mode, giving it a script as input.

  Example

  ```
    isql -b -i my_fb.sql -o results.log -m -m2
  ```

  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.

- Command-line option -M2 to send the statistics and plans to the same output file as the other output (via the -o[utput] switch). (C. Valderrama)

  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
    or
  - 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 goes 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.

- ODS version is now returned in the SHOW DATABASE command (C. Valderrama)
- New command SET HEADING ON/OFF toggle

  Some people consider it useful to be able to do a SELECT inside isql and have the output sent to a file, for additional processing later, especially if the number of columns makes isql display impracticable. However, isql by default prints column headers and in this scenario, they are a nuisance. Therefore, printing the column headers -- previously a fixed feature -- can now be 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: this switch cannot be deactivated with a command line parameter.

  Using SET will display the state of SET HEAD, along with other switches that can be toggled on/off in the isql shell.

ISQL Bugs fixed

- SF #910430 - ISQL and database dialect (C. Valderrama, B. Rodriguez Somoza)

  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 (B. Rodriguez Somoza)
- SF #223513 - Ambiguity between tables and views (B. Rodriguez Somoza)
- SF #518349 - ISQL show mangles relationship (B. Rodriguez Somoza)

Better descriptions required.

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.

External Functions (UDFs)

Ability to signal SQL NULL via a NULL pointer

C. Valderrama C.

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. 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 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.

Bugs Fixed

Bug ID: SF #1124720	A. Peshkov

Problem with "FOR EXECUTE STATEMENT ... DO SUSPEND;"

Bug ID: SF #1076858	V. Horsun

Possible corruption in classic server

Bug ID: SF #1116809	A. dos Santos Fernandes

Incorrect data type conversion

Bug ID: SF #1111570	C. Valderrama

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

Bug ID: Not registered	N. Samofatov

Possible server lockup/crash when 'RELEASE SAVEPOINT xxx ONLY' syntax is used or when existing savepoint name is reused in transaction context

---

Bug ID: SF #217042 (part fix)	C. Valderrama

IB doesn't validate weird constructions

Bug ID: SF #1108909	C. Valderrama

View can be created w/o rights on table name like "a b"

Bug ID: Not registered	D. Yemanov

Rare client crashes caused by improperly cleaned XDR packets

Bug ID: Not registered	C. Valderrama

Usage of an invalid index in an explicit plan causes garbage to be shown in the error message instead of the rejected index name.

Bug ID: SF #504978	C. Valderrama

GPRE variable names being truncated

Bug ID: SF #527677	C. Valderrama

"ANSI85 compatible COBOL" switch broken

Bug ID: SF #1103666	C. Valderrama

GPRE uses inconsistent lengths

Bug ID: SF #1103670	C. Valderrama

GPRE invalidates a quoted cursor name after it's opened

Bug ID: SF #1103683 C. Valderrama
GPRE doesn't check the length of the db alias

Bug ID: SF #1103740 C. Valderrama
GPRE doesn't detect duplicate quoted cursors names

Bug ID: SF #512975 C. Valderrama
Embed spaces and CR+LF before DEFAULT

Bug ID: Not registered A. Peshkov
Server crash during SuperServer shutdown

Bug ID: not registered D. Yemanov
Column-level SQL privileges were being preserved after the affected column was dropped.

Bug ID: not registered N. Samofatov
Memory leakage was occurring when selectable stored procedures were called from PSQL or in subqueries.

Bug ID: not registered N. Samofatov
Fixed some backup issues with stream BLOBs that caused them to be truncated under some conditions.

Bug ID: not registered A. Peshkov
Diagnostics about missing/unusable UDF module needed improvement. Details in the UDF section.

Bug ID: SF #1065511 N. Samofatov
Clients on Windows XP SP2 were slow connecting to a Linux server.

Bug ID: SF #459059 D. Yemanov
Index breaks = ANY result. MORE INFO REQUIRED.

Bug ID: SF #543106 D. Yemanov
Bug with ALL keyword. MORE INFO REQUIRED

Bug ID: SF #1057538 C. Valderrama
The server would crash if the output parameter of a UDF was not the last parameter.

Bug ID: not registered A. Peshkov

System users "AUTHENTICATOR" and "SWEEPER" were lost, causing "SQL SERVER" to be reported instead.

Bug ID: not registered A. Brinkman

Ambiguous queries were still possible under some conditions.

Bug ID: not registered V. Horsun

Don't rollback prepared 2PC sub-transaction. Vlad, it is very unclear what you mean here.

Bug ID: SF #571026 D. Yemanov

INET/INET_connect: gethostbyname was not working properly.

Bug ID: SF #223058 D. Yemanov

Multi-hop server capability was broken.

Bug ID: not registered N. Samofatov

A number of possible server crash conditions had been reported by Valgrind. (This report needs some explanation.)

Bug ID: SF #735720 A. Brinkman

SELECT ... STARTING WITH :v was wrong when :v = ''

Bug ID: not registered N. Samofatov

Server would crash when a wrong type or domain name was specified when changing the data type for a column. More info: what is meant by "wrong type"?

Bug ID: not cited N. Samofatov

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.

Bug ID: not cited N. Samofatov

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

Bug ID: not registered N. Samofatov

There were some problems with the mapping of UDF arguments to parameters.

Bug ID: not registered N. Samofatov

Incorrect accounting of attachment pointers used inside the lock structure was causing the server to crash.

Bug ID: SF #910423 C. Valderrama

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"
```

Bug ID: not registered N. Samofatov
There were some rounding problems in date/time arithmetic.

Bug ID: not registered N. Samofatov
Line numbers in DSQL parser were being miscounted when multi-line literals and identifiers were used.

Bug ID: not registered J. Starkey
In v.1.5, random crashes would occur during a restore.

Bug ID: not registered N. Samofatov
Crash/lock-up with multiple calls of isc_dsql_prepare for a single statement (like IBO does). Since IBO does not make multiple calls to isc_dsql_prepare per statement, some more sensible description is required here.

Bug ID: not registered D. Yemanov
Server would crash when the system year was set too high or too low.

Bug ID: not registered D. Yemanov
Server would crash when the stream number exceeded the limit.

Bug ID: not registered A. Peshkov
EXECUTE STATEMENT had a memory leak.

Bug ID: not registered D. Yemanov
UDF arguments were being prepared/optimized twice.

Bug ID: not registered N. Samofatov
Conversion from string to blob had a memory leak.

Bug ID: not registered A. Brinkman
Interdependent views caused problems during the restore process.

Bug ID: SF #750664 N. Samofatov
Issues with read-only databases and transactions. WHAT ISSUES?

Bug ID: not registered N. Samofatov
Fixed memory leak from connection pool in isc_database_info. WHAT DOES THIS ACTUALLY MEAN TO SAY?

Bug ID: not registered D. Yemanov
Server would crash when outer aggregation was performed and explicit plans were used in subqueries.

Bug ID: not registered A. Peshkov

DECLARE FILTER would cause the server to crash.

Bug ID: not registered A. Brinkman

There were issues with dates below Julian date [zero?] stored in indices.

Bug ID: SF #781610 J. Bellardo, B. Rodriguez Somoza

Comments in ISQL using '--' were causing problems.

Bug ID: SF #544132, #728839 C. Valderrama

Nulls handling in UDFs was causing problems.

Bug ID: SF #784121 C. Valderrama

Some expressions in outer join conditions were causing problems.

Bug ID: SF #750659 C. Valderrama

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.

NOTE :: The patch posted to the Firebird 2 branch may not have fully fixed the bug. The same patch in Firebird 1.5.1 caused garbage numbers to be stored in the generators, not zero. Testers please report.

Code Cleanup

-L[ocal] command-line switch for SS on Win32 is gone D. Yemanov

Command line switch L for SuperServer on Windows is no longer recognized, since the old local protocol was deleted.

C. Valderrama

Assorted cleanup

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

---