# Table of Contents

1. General Notes .......................................................... 4
   - Compatibility with Older Versions ............................. 4
   - Bug Reporting ......................................................... 4
   - Documentation ....................................................... 4

2. New In Firebird 5.0 ..................................................... 6
   - Summary of New Features ....................................... 6
     - Complete In Firebird 5.0 Beta 1 ............................. 6

3. Changes in the Firebird Engine .................................. 10
   - Quick Links ........................................................ 10
   - Support for parallel operations ............................... 10
   - Inline minor ODS upgrade ..................................... 11
   - More cursor-related details in the plan output .......... 12
   - Denser compression of records ............................... 13
   - Compiled statement cache .................................... 13
   - SQL and PSQL profiler .......................................... 13
     - Package routines .............................................. 16
     - Snapshot tables ............................................... 19
     - Auxiliary views ............................................... 22
   - RDB$LOB_UTIL package .......................................... 25
     - Package routines .............................................. 25
     - Examples ....................................................... 27

4. Changes to the Firebird API and ODS .......................... 30
   - ODS (On-Disk Structure) Changes ............................. 30
     - New Minor ODS Number ....................................... 30
     - New System Tables ............................................ 30
     - New Columns in System Tables ............................. 30
   - Application Programming Interfaces ........................ 30
     - Main API Extensions .......................................... 30
     - Extensions to various getInfo() Methods ............... 31
     - Services API Extensions .................................... 31

5. Reserved Words and Changes .................................... 32
   - New Keywords in Firebird 5.0 ................................ 32
     - Non-reserved .................................................. 32

6. Configuration Additions and Changes .......................... 33
   - Parameters for Parallel Operations ........................ 33
     - MaxParallelWorkers .......................................... 33
     - ParallelWorkers .............................................. 33
   - Other Parameters ............................................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12. Bugs Fixed</td>
<td>49</td>
</tr>
<tr>
<td>Firebird 5.0 Beta 1 Release: Bug Fixes</td>
<td>49</td>
</tr>
<tr>
<td>Core Engine</td>
<td>49</td>
</tr>
<tr>
<td>Server Crashes/Hangups</td>
<td>50</td>
</tr>
<tr>
<td>Utilities</td>
<td>51</td>
</tr>
<tr>
<td>13. Firebird 5.0 Project Teams</td>
<td>52</td>
</tr>
<tr>
<td>Appendix A: Licence Notice</td>
<td>53</td>
</tr>
</tbody>
</table>
Chapter 1. General Notes

Thank you for choosing Firebird 5.0. We cordially invite you to test it hard against your expectations and engage with us in identifying and fixing any bugs you might encounter.

ODS (On-Disk Structure) 13.1 is introduced. It’s a minor ODS upgrade, so databases in ODS 13.0 (created by Firebird 4.0) may still be opened with a Firebird 5.0 server (with some new features being unavailable), but databases in older ODS cannot be opened.

Databases created in early (pre-Beta) builds of Firebird 5.0 may be inaccessible in the Beta 1 release and have to be recreated. ODS was changed a few times during the development cycle and the Firebird Project generally does not guarantee ODS being stable before the Beta stage.

The engine library is still named engine13.dll (Windows) and libEngine13.so (POSIX). The security database is named security5.fdb. Binaries layout and configuration are unchanged from Firebird 4.

That said, you can copy the Firebird engine library from the Firebird 3.0 distribution package (named engine12.dll (Windows) and libEngine12.so (POSIX), and located inside the /plugins sub-directory) to continue working with databases in ODS12 without needing a backup/restore. However, new features introduced with Firebird 5.0 will not be accessible.

Compatibility with Older Versions

Known incompatibilities are detailed in the Compatibility Issues chapter.

Bug Reporting

Bugs fixed in this release are listed and described in the chapter entitled Bugs Fixed.

• If you think you have discovered a new bug in this release, please make a point of reading the instructions for bug reporting in the article How to Report Bugs Effectively, at the Firebird Project website.

• If you think a bug fix has not worked, or has caused a regression, please locate the original bug report in the Tracker, reopen it if necessary, and follow the instructions below.

Follow these guidelines as you attempt to analyse your bug:

1. Write detailed bug reports, supplying the exact build number of your Firebird kit. Also provide details of the OS platform.

2. Include reproducible test data in your report and post it to our Tracker.

Documentation

You will find all the README documents referred to in these notes—as well as many others not
referred to — in the doc subdirectory of your Firebird 5.0 installation.

— *The Firebird Project*
Chapter 2. New In Firebird 5.0

Summary of New Features

Firebird 5.0 introduces many improvements without any changes in architecture or operation, the most important are:

- Parallel (multi-threaded) operation for backup/restore, sweep and index creation;
- Partial indices;
- SKIP LOCKED clause for SELECT WITH LOCK, UPDATE and DELETE statements;
- Inline minor ODS upgrade;
- Compiled statement cache;
- PSQL and SQL profiler;
- Support for WHEN NOT MATCHED BY SOURCE for MERGE statement;
- Support multiple rows for DML RETURNING;
- New built-in functions and packages;
- Denser record-level compression;
- Network support for scrollable cursors;

The following list summarises the features and changes, with links to the chapters and topics where more detailed information can be found.

Complete In Firebird 5.0 Beta 1

Parallel (multi-threaded) operations

Such operations as logical backup/restore, sweeping and CREATE INDEX statement execution can be executed in parallel by multiple threads, thus decreasing the total operation time.

Tracker references: #1783, #3374, #7447

See chapters Support for parallel operations, Parallel backup/restore and Parallel sweep for more details.

Support for partial indices

The CREATE INDEX DDL statement has been extended to support partial indices, i.e. an index may now declare a condition that defines the subset of records to be indexed.

Tracker reference: #7257

SKIP LOCKED clause

New clause SKIP LOCKED was introduced for statements SELECT WITH LOCK, UPDATE and DELETE. It allows to skip the already locked records while reading the table.

Tracker reference: #7350
Inline minor ODS upgrade

An ability to upgrade the database to the latest minor ODS version has been introduced, it does not require a backup/restore cycle.

Tracker reference: #7397

Compiled statement cache

Per-attachment cache of compiled SQL statements has been implemented.

Tracker reference: #7144

PSQL and SQL profiler

A built-in ability to profile SQL and PSQL statements has been added, thus making possible to measure execution time at different levels.

Tracker reference: #7086

Support for WHEN NOT MATCHED BY SOURCE in the MERGE statement

The MERGE statement has been extended to support the WHEN NOT MATCHED BY SOURCE clause.

Tracker reference: #6681

Built-in functions UNICODE_CHAR and UNICODE_VAL

New built-in functions UNICODE_CHAR and UNICODE_VAL have been added to allow conversion between Unicode code point and character.

Tracker reference: #6798

RDB$BLOB_UTIL new system package

New system package RDB$BLOB_UTIL has been added to allow various operations with BLOBs in the PSQL modules.

Tracker reference: #281

Support multiple rows being returned by DML with the RETURNING clause

The RETURNING clause, if used in DSQL queries, now allows multiple rows to be returned.

Tracker reference: #6815

Optimize the record-level RLE algorithm for a denser compression of shorter-than-declared strings and sets of subsequent NULLs

The built-in compression algorithm has been improved to allow denser compression of records.

Tracker reference: #4723

More cursor-related details in the plan output

Execution plan now contains more information about cursors.

Tracker reference: #7441
Other improvements are briefly listed below, please follow the tracker references for more information.

**Unify display of system procedures & packages with other system objects**
Tracker reference: #7411

**Simplify client library build**
Tracker reference: #7399

**Performance improvement for BLOB copying**
Tracker reference: #7382

**Cost-based choice between nested loop join and hash join**
Tracker reference: #7331

**Create Android packages with all necessary files in all architectures (x86, x64, arm32, arm64)**
Tracker reference: #7293

**Unify release filenames**
Tracker reference: #7284

**Improve ICU version mismatch diagnostics**
Tracker reference: #7169

**Provide ability to see in the trace log events related to missing security context**
Tracker reference: #7165

**ResultSet.getInfo() new API method**
Tracker reference: #7083

**Network support for scrollable cursors**
Tracker reference: #7051

**Add table MON$COMPILED_STATEMENTS and also column MON$COMPILED_STATEMENT_ID to both MON$STATEMENTS and MON$CALL_STACK tables**
Tracker reference: #7050

**Make ability to add comment to mapping (‘COMMENT ON MAPPING ... IS ...’)**
Tracker reference: #7046

**Results of negation must be the same for each datatype (SMALLINT / INT / BIGINT / INT128) when argument is minimum value for this type**
Tracker reference: #7025

**Transform OUTER joins into INNER ones if the WHERE condition violates the outer join rules**
Tracker reference: #6992
Add way to retrieve statement BLR with Statement.getInfo() and ISQL's SET EXEC_PATH_DISPLAY BLR
 Tracker reference: #6910

SIMILAR TO should use index when pattern starts with non-wildcard character (as LIKE does)
 Tracker reference: #6873

Add column MON$SESSION_TIMEZONE to the table MON$ATTACHMENTS
 Tracker reference: #6794

Allow parenthesized query expression for standard-compliance
 Tracker reference: #6740

System table with keywords
 Tracker reference: #6713

Support full SQL standard character string literal syntax
 Tracker reference: #5589

Support full SQL standard binary string literal syntax
 Tracker reference: #5588

Allow sub-routines to access variables/parameters defined at the outer/parent level
 Tracker reference: #4769

Avoid data retrieval if the WHERE clause always evaluates to FALSE
 Tracker reference: #1708
Chapter 3. Changes in the Firebird Engine

Quick Links

- Support for parallel operations
- Inline minor ODS upgrade
- More cursor-related details in the plan output
- Compiled statement cache
- Denser compression of records
- SQL and PSQL profiler
- RDB$BLOB_UTIL package

Support for parallel operations

Vlad Khorsun

Tracker ticket: #7447

The Firebird engine can now execute some tasks using multiple threads in parallel. Currently, parallel execution is implemented for the sweep and the index creation tasks. Parallel execution is supported for both automatic and manual sweep.

To handle a task with multiple threads, the engine runs additional worker threads and creates internal worker attachments. By default, parallel execution is not enabled. There are two ways to enable parallelism in a user attachment:

1. set the number of parallel workers in DPB using new tag `isc_dpb_parallel_workers`,
2. set the default number of parallel workers using new setting `ParallelWorkers` in firebird.conf.

The `gfix` utility has a new command-line switch, `-parallel`, that allows to set the number of parallel workers for the sweep task. For example:

```
gfix -sweep -parallel 4 <database>
```

will run sweep on the given database and asks the engine to use 4 workers. `gfix` uses DPB tag `isc_dpb_parallel_workers` when attaches to `<database>`, if switch `-parallel` is present.

The new firebird.conf setting `ParallelWorkers` sets the default number of parallel workers that can be used by any user attachment running parallelizable task. The default value is 1 and means no use of additional parallel workers. The value in the DPB has a higher priority than the setting in firebird.conf.

To control the number of additional workers that can be created by the engine, there are two new settings in firebird.conf:
**ParallelWorkers**

Sets default number of parallel workers that used by user attachments. Can be overridden by attachment using tag `isc_dpb_parallel_workers` in DPB.

**MaxParallelWorkers**

Limits the number of simultaneously used workers for the given database and Firebird process.

Internal worker attachments are created and managed by the engine itself. The engine maintains per-database pools of worker attachments. The number of threads in each pool is limited by the value of the `MaxParallelWorkers` setting. The pools are created by each Firebird process independently.

In SuperServer architecture worker attachments are implemented as light-weight system attachments, while in Classic and SuperClassic they look like usual user attachments. All worker attachments are embedded into creating server process. Thus, in Classic architectures there is no additional server processes. Worker attachments are present in monitoring tables. Idle worker attachments are destroyed after 60 seconds of inactivity. Also, in Classic architectures, worker attachments are destroyed immediately after last user connection detaches the from database.

Examples:

Set in firebird.conf `ParallelWorkers = 4, MaxParallelWorkers = 8` and restart Firebird server.

1. Connect to test database not using `isc_dpb_parallel_workers` in DPB and execute `CREATE INDEX ...` SQL statement. On commit, the index will be actually created and engine will use 3 additional worker attachments. In total, 4 attachments in 4 threads will work on index creation.

2. Ensure auto-sweep is enabled for test database. When auto-sweep will run on that database, it also will use 3 additional workers (and run within 4 threads).

3. More than one single task at time can be parallelized: make 2 attachments and execute `CREATE INDEX ...` in each of them (of course indices to be built should be different). Each index will be created using 4 attachments (1 user and 3 worker) and 4 threads.

4. Run `gfix -sweep <database>` without specifying switch `-parallel`: sweep will run using 4 attachments in 4 threads.

5. Run `gfix -sweep -parallel 2 <database>`: sweep will run using 2 attachments in 2 threads. This shows that value in DPB tag `isc_dpb_parallel_workers` overrides value of setting `ParallelWorkers`.

**Inline minor ODS upgrade**

Dmitry Yemanov

Tracker ticket: [#7397](#7397)

This feature allows to upgrade the existing database to the newest ODS version without backup/restore, provided that the database belongs to the same major ODS version.

For example, a database created by Firebird 4.0 uses ODS 13.0 and thus can be upgraded to the ODS 13.1 used by Firebird 5.0.
Notes:

- Upgrade must be done manually, using `gfix -upgrade` command
- It requires exclusive access to the database, error is thrown otherwise
- `USE_GFIX_UTILITY` system privilege is required
- Upgrade is transactional, all changes are reverted if any error happens

Usage:

```
gfix -upgrade <database>
```

See also **ODS upgrade by gfix**.

This is a one-way modification, downgrading backward is impossible. So please make a database copy before upgrading, just to have a recovery point if something goes wrong during the process.

**More cursor-related details in the plan output**

Dmitry Yemanov

Tracker ticket: [#7441](#7441)

Detailed plan output now distinguishes between user-specified `SELECT` statements (reported as `select expressions`), PSQL declared cursors and sub-queries. Both legacy and detailed plans now also include information about cursor's position (line/column) inside their PSQL module.

Examples:

```
-- line 23, column 2
PLAN (DISTRICT INDEX (DISTRICT_PK))
-- line 28, column 2
PLAN JOIN (CUSTOMER INDEX (CUSTOMER_PK), WAREHOUSE INDEX(WAREHOUSE_PK))
```

Select Expression (line 23, column 2)

- -> Singularity Check
  - -> Filter
    - -> Table "DISTRICT" Access By ID
      - -> Bitmap
        - -> Index "DISTRICT_PK" Unique Scan

Select Expression (line 28, column 2)

- -> Singularity Check
  - -> Nested Loop Join (inner)
    - -> Filter
      - -> Table "CUSTOMER" Access By ID
        - -> Bitmap
Line/column numbers (as well as PSQL declared cursors) cannot be seen directly in the plan for user-specified SQL queries, except if the query is EXECUTE BLOCK. However, they are accessible in the MON$EXPLAINED_PLAN column in either MON$STATEMENTS or MON$COMPILED_STATEMENTS tables.

### Denser compression of records

**Dmitry Yemanov**

Tracker ticket: [#4723](#4723)

Starting with ODS 13.1, the engine uses an advanced RLE compression method (with variable-length counter) that stores repeating byte sequences more effectively, thus reducing the storage overhead. This improves compression for long VARCHAR fields (especially UTF8 encoded) that are filled only partially.

### Compiled statement cache

**Adriano dos Santos Fernandes**

Tracker ticket: [#7144](#7144)

The engine now maintains a per-attachment cache of compiled SQL statements. By default, caching is enabled, the caching threshold is defined by the `MaxStatementCacheSize` parameter in firebird.conf. It can be disabled by setting `MaxStatementCacheSize` to zero.

The cache is maintained automatically; cached statements are invalidated when required (usually when some DDL statement is executed).

### SQL and PSQL profiler

**Adriano dos Santos Fernandes**

Tracker ticket: [#7086](#7086)

The profiler allows users to measure performance cost of SQL and PSQL code. It's implemented with a system package in the engine passing data to a profiler plugin.

This documentation treats the engine and plugin parts as a single thing, in the way the default profiler (Default_Profiler) is going to be used.

The RDB$PROFILER package can profile execution of PSQL code, collecting statistics of how many times each line was executed along with its minimum, maximum and accumulated execution times (with nanoseconds precision), as well as open and fetch statistics of implicit and explicit SQL
To collect profile data, a user must first start a profile session with `RDB$PROFILER.START_SESSION`. This function returns a profile session ID which is later stored in the profiler snapshot tables to be queried and analyzed by the user. A profiler session may be local (same attachment) or remote (another attachment).

Remote profiling just forwards commands to the remote attachment. So, it's possible that a client profiles multiple attachments simultaneously. It's also possible that a locally or remotely started profile session have commands issued by another attachment.

Remotely issued commands require that the target attachment is in an idle state, i.e. not executing others requests. When the target attachment is not idle, the call blocks waiting for that state.

If the remote attachment is from a different user, the calling user must have the system privilege `PROFILE_ANY_ATTACHMENT`.

After a session is started, PSQL and SQL statements statistics are collected in memory. A profile session collects data only of statements executed in the same attachment associated with the session. Data is aggregated and stored per requests (i.e. a statement execution). When querying snapshot tables, the user may do extra aggregation per statement, or use the auxiliary views that do that automatically.

A session may be paused to temporarily disable statistics collecting. It may be resumed later to return statistics collection in the same session.

A new session may be started when a session is already active. In that case, it has the same semantics of finishing the current session with `RDB$PROFILER.FINISH_SESSION(FALSE)`, so snapshots tables are not updated.

To analyze the collected data, the user must flush the data to the snapshot tables, which can be done by finishing or pausing a session (with `FLUSH` parameter set to `TRUE`), or calling `RDB$PROFILER.FLUSH`. Data is flushed using an autonomous transaction (a transaction started and finished for the specific purpose of profiler data update).

Below is a sample profile session and queries for data analysis.

1. Preparation — create table and routines that will be analyzed

```sql
CREATE TABLE tab (
    id integer NOT NULL,
    val integer NOT NULL
);

SET TERM !;

CREATE OR ALTER FUNCTION mult(p1 integer, p2 integer) RETURNS integer AS BEGIN RETURN p1 * p2;
```
create or alter procedure ins
as
  declare n integer = 1;
begin
  while (n <= 1000)
  begin
    if (mod(n, 2) = 1) then
      insert into tab values (:n, mult(:n, 2));
    n = n + 1;
  end
end!

set term ;!

2. Start profiling

select rdb$profiler.start_session('Profile Session 1') from rdb$database;
set term !;
execute block
as
begin
  execute procedure ins;
  delete from tab;
end!
set term ;!
execute procedure rdb$profiler.finish_session(true);
execute procedure ins;
select rdb$profiler.start_session('Profile Session 2') from rdb$database;
select mod(id, 5),
  sum(val)
from tab
where id <= 50
  group by mod(id, 5)
  order by sum(val);
execute procedure rdb$profiler.finish_session(true);

3. Data analysis
set transaction read committed;

select * from plg$prof_sessions;

select * from plg$prof_psql_stats_view;

select * from plg$prof_record_source_stats_view;

select preq.*
  from plg$prof_requests preq
  join plg$prof_sessions pses
    on pses.profile_id = preq.profile_id and
    pses.description = 'Profile Session 1';

select pstat.*
  from plg$prof_psql_stats pstat
  join plg$prof_sessions pses
    on pses.profile_id = pstat.profile_id and
    pses.description = 'Profile Session 1'
  order by pstat.profile_id,
            pstat.request_id,
            pstat.line_num,
            pstat.column_num;

select pstat.*
  from plg$prof_record_source_stats pstat
  join plg$prof_sessions pses
    on pses.profile_id = pstat.profile_id and
    pses.description = 'Profile Session 2'
  order by pstat.profile_id,
            pstat.request_id,
            pstat.cursor_id,
            pstat.record_source_id;

---

**Package routines**

**Function START_SESSION**

RDB$PROFILER.START_SESSION starts a new profiler session, makes it the current session (of the given ATTACHMENT_ID) and returns its identifier.

If FLUSH_INTERVAL is different from NULL, auto-flush is set up in the same way as manually calling RDB$PROFILER.SET_FLUSH_INTERVAL.

If PLUGIN_NAME is NULL (the default), it uses the database configuration DefaultProfilerPlugin.

PLUGIN_OPTIONS are plugin specific options and currently should be NULL for the Default_Profiler plugin.

Input parameter(s):
Procedure PAUSE_SESSION

RDB$PROFILER.PAUSE_SESSION pauses the current profiler session (of the given ATTACHMENT_ID), so the next executed statements statistics are not collected.

If FLUSH is TRUE, the snapshot tables are updated with data up to the current moment, otherwise data remains only in memory for later update.

Calling RDB$PROFILER.PAUSE_SESSION(TRUE) has the same semantics of calling RDB$PROFILER.PAUSE_SESSION(FALSE) followed by RDB$PROFILER.FLUSH (using the same ATTACHMENT_ID).

Input parameter(s):

• FLUSH type BOOLEAN NOT NULL default FALSE
• ATTACHMENT_ID type BIGINT NOT NULL default CURRENT_CONNECTION

Procedure RESUME_SESSION

RDB$PROFILER.RESUME_SESSION resumes the current profiler session (of the given ATTACHMENT_ID), if it was paused, so the next executed statements statistics are collected again.

Input parameter(s):

• ATTACHMENT_ID type BIGINT NOT NULL default CURRENT_CONNECTION

Procedure FINISH_SESSION

RDB$PROFILER.FINISH_SESSION finishes the current profiler session (of the given ATTACHMENT_ID).

If FLUSH is TRUE, the snapshot tables are updated with data of the finished session (and old finished sessions not yet present in the snapshot), otherwise data remains only in memory for later update.

Calling RDB$PROFILER.FINISH_SESSION(TRUE) has the same semantics of calling RDB$PROFILER.FINISH_SESSION(FALSE) followed by RDB$PROFILER.FLUSH (using the same ATTACHMENT_ID).

Input parameter(s):

• FLUSH type BOOLEAN NOT NULL default TRUE
• ATTACHMENT_ID type BIGINT NOT NULL default CURRENT_CONNECTION
Procedure CANCEL_SESSION

RDB$PROFILER.CANCEL_SESSION cancels the current profiler session (of the given ATTACHMENT_ID).

All session data present in the profiler plugin is discarded and will not be flushed.

Data already flushed is not deleted automatically.

Input parameter(s):

- ATTACHMENT_ID type BIGINT NOT NULL default CURRENT_CONNECTION

Procedure DISCARD

RDB$PROFILER.DISCARD removes all sessions (of the given ATTACHMENT_ID) from memory, without flushing them.

If there is an active session, it is cancelled.

Input parameter(s):

- ATTACHMENT_ID type BIGINT NOT NULL default CURRENT_CONNECTION

Procedure FLUSH

RDB$PROFILER.FLUSH updates the snapshot tables with data from the profile sessions (of the given ATTACHMENT_ID) in memory.

After flushing, the data is stored in tables PLG$PROF_SESSIONS, PLG$PROF_STATEMENTS, PLG$PROF_RECORD_SOURCES, PLG$PROF_REQUESTS, PLG$PROF_PSQL_STATS and PLG$PROF_RECORD_SOURCE_STATS and may be read and analyzed by the user.

Data is updated using an autonomous transaction, so if the procedure is called in a snapshot transaction, data will not be directly readable in the same transaction.

Once flush happens, finished sessions are removed from memory.

Input parameter(s):

- ATTACHMENT_ID type BIGINT NOT NULL default CURRENT_CONNECTION

Procedure SET_FLUSH_INTERVAL

RDB$PROFILER.SET_FLUSH_INTERVAL turns periodic auto-flush on (when FLUSH_INTERVAL is greater than 0) or off (when FLUSH_INTERVAL is equal to 0).

FLUSH_INTERVAL is interpreted as number of seconds.

Input parameter(s):

- FLUSH_INTERVAL type INTEGER NOT NULL
- ATTACHMENT_ID type BIGINT NOT NULL default CURRENT_CONNECTION
Snapshot tables

Snapshot tables (as well views and sequence) are automatically created in the first usage of the profiler. They are owned by the database owner, with read/write permissions for \texttt{PUBLIC}.

When a session is deleted, the related data in other profiler snapshot tables are automatically deleted too through foreign keys with \texttt{DELETE  CASCADE} option.

Below is the list of tables that stores profile data.

\textbf{Table PLG$PROF_SESSIONS}

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE_ID</td>
<td>BIGINT</td>
<td>Profile session ID</td>
</tr>
<tr>
<td>ATTACHMENT_ID</td>
<td>BIGINT</td>
<td>Attachment ID</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>CHAR(63)</td>
<td>Username</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR(255)</td>
<td>Description passed in</td>
</tr>
<tr>
<td>START_TIMESTAMP</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Moment the profile session was started</td>
</tr>
<tr>
<td>FINISH_TIMESTAMP</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Moment the profile session was finished</td>
</tr>
</tbody>
</table>

\textbf{Primary key}

\texttt{PROFILE_ID}

\textbf{Table PLG$PROF_STATEMENTS}

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE_ID</td>
<td>BIGINT</td>
<td>Profile session ID</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>BIGINT</td>
<td>Statement ID</td>
</tr>
<tr>
<td>PARENT_STATEMENT_ID</td>
<td>BIGINT</td>
<td>Parent statement ID — related to sub routines</td>
</tr>
<tr>
<td>STATEMENT_TYPE</td>
<td>VARCHAR(20)</td>
<td>BLOCK, FUNCTION, PROCEDURE or TRIGGER</td>
</tr>
<tr>
<td>PACKAGE_NAME</td>
<td>CHAR(63)</td>
<td>Package of FUNCTION or PROCEDURE</td>
</tr>
<tr>
<td>ROUTINE_NAME</td>
<td>CHAR(63)</td>
<td>Routine name of FUNCTION, PROCEDURE or TRIGGER</td>
</tr>
<tr>
<td>SQL_TEXT</td>
<td>BLOB</td>
<td>SQL text for BLOCK</td>
</tr>
</tbody>
</table>

\textbf{Primary key}

\texttt{PROFILE_ID, STATEMENT_ID}

\textbf{Table PLG$PROF_CURSORS}

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE_ID</td>
<td>BIGINT</td>
<td>Profile session ID</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>BIGINT</td>
<td>Statement ID</td>
</tr>
</tbody>
</table>

Chapter 3. Changes in the Firebird Engine
<table>
<thead>
<tr>
<th>CURSOR_ID type INTEGER</th>
<th>Cursor ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME type CHAR(63) CHARACTER SET UTF8</td>
<td>Name of explicit cursor</td>
</tr>
<tr>
<td>LINE_NUM type INTEGER</td>
<td>Line number of the cursor</td>
</tr>
<tr>
<td>COLUMN_NUM type INTEGER</td>
<td>Column number of the cursor</td>
</tr>
<tr>
<td><strong>Primary key</strong></td>
<td>PROFILE_ID, STATEMENT_ID, CURSOR_ID</td>
</tr>
</tbody>
</table>

**Table PLG$PROF_RECORD_SOURCES**

| PROFILE_ID type BIGINT | Profile session ID |
| STATEMENT_ID type BIGINT | Statement ID |
| CURSOR_ID type INTEGER | Cursor ID |
| RECORD_SOURCE_ID type INTEGER | Record source ID |
| PARENT_RECORD_SOURCE_ID type INTEGER | Parent record source ID |
| ACCESS_PATH type VARCHAR(255) CHARACTER SET UTF8 | Access path for the record source |
| **Primary key** | PROFILE_ID, STATEMENT_ID, CURSOR_ID, RECORD_SOURCE_ID |

**Table PLG$PROF_REQUESTS**

| PROFILE_ID type BIGINT | Profile session ID |
| REQUEST_ID type BIGINT | Request ID |
| STATEMENT_ID type BIGINT | Statement ID |
| CALLER_REQUEST_ID type BIGINT | Caller request ID |
| START_TIMESTAMP type TIMESTAMP WITH TIME ZONE | Moment this request was first gathered profile data |
| FINISH_TIMESTAMP type TIMESTAMP WITH TIME ZONE | Moment this request was finished |
| TOTAL_ELAPSED_TIME type BIGINT | Accumulated elapsed time (in nanoseconds) of the request |
| **Primary key** | PROFILE_ID, REQUEST_ID |

**Table PLG$PROF_PSQL_STATS**

| PROFILE_ID type BIGINT | Profile session ID |
| REQUEST_ID type BIGINT | Request ID |
Chapter 3. Changes in the Firebird Engine

| LINE_NUM | INTEGER | Line number of the statement |
| COLUMN_NUM | INTEGER | Column number of the statement |
| STATEMENT_ID | BIGINT | Statement ID |
| COUNTER | BIGINT | Number of executed times of the line/column |
| MIN_ELAPSED_TIME | BIGINT | Minimal elapsed time (in nanoseconds) of a line/column execution |
| MAX_ELAPSED_TIME | BIGINT | Maximum elapsed time (in nanoseconds) of a line/column execution |
| TOTAL_ELAPSED_TIME | BIGINT | Accumulated elapsed time (in nanoseconds) of the line/column executions |

**Primary key**

PROFILE_ID, REQUEST_ID, LINE_NUM, COLUMN_NUM

### Table PLG$PROF_RECORD_SOURCE_STATS

| PROFILE_ID | BIGINT | Profile session ID |
| REQUEST_ID | BIGINT | Request ID |
| CURSOR_ID | INTEGER | Cursor ID |
| RECORD_SOURCE_ID | INTEGER | Record source ID |
| STATEMENT_ID | BIGINT | Statement ID |
| OPEN_COUNTER | BIGINT | Number of open times of the record source |
| OPEN_MIN_ELAPSED_TIME | BIGINT | Minimal elapsed time (in nanoseconds) of a record source open |
| OPEN_MAX_ELAPSED_TIME | BIGINT | Maximum elapsed time (in nanoseconds) of a record source open |
| OPEN_TOTAL_ELAPSED_TIME | BIGINT | Accumulated elapsed time (in nanoseconds) of the record source openings |
| FETCH_COUNTER | BIGINT | Number of fetch times of the record source |
| FETCH_MIN_ELAPSED_TIME | BIGINT | Minimal elapsed time (in nanoseconds) of a record source fetch |
| FETCH_MAX_ELAPSED_TIME | BIGINT | Maximum elapsed time (in nanoseconds) of a record source fetch |
| FETCH_TOTAL_ELAPSED_TIME | BIGINT | Accumulated elapsed time (in nanoseconds) of the record source fetches |
Primary key

PROFILE_ID, REQUEST_ID, CURSOR_ID, RECORD_SOURCE_ID

Auxiliary views

These views help profile data extraction aggregated at statement level.

They should be the preferred way to analyze the collected data. They can also be used together with the tables to get additional data not present on the views.

After hotspots are found, one can drill down in the data at the request level through the tables.

View PLG$PROF_STATEMENT_STATS_VIEW

```sql
select req.profile_id,
       req.statement_id,
       sta.statement_type,
       sta.package_name,
       sta.routine_name,
       sta.parent_statement_id,
       sta_parent.statement_type parent_statement_type,
       sta_parent.routine_name parent_routine_name,
       (select sql_text
        from plg$prof_statements
        where profile_id = req.profile_id and
              statement_id = coalesce(sta.parent_statement_id, req.statement_id)
        ) sql_text,
       count(*) counter,
       min(req.total_elapsed_time) min_elapsed_time,
       max(req.total_elapsed_time) max_elapsed_time,
       cast(sum(req.total_elapsed_time) as bigint) total_elapsed_time,
       cast(sum(req.total_elapsed_time) / count(*) as bigint) avg_elapsed_time
from plg$prof_requests req
join plg$prof_statements sta
    on sta.profile_id = req.profile_id and
       sta.statement_id = req.statement_id
left join plg$prof_statements sta_parent
    on sta_parent.profile_id = sta.profile_id and
       sta_parent.statement_id = sta.parent_statement_id
group by req.profile_id,
        req.statement_id,
        sta.statement_type,
        sta.package_name,
        sta.routine_name,
        sta.parent_statement_id,
        sta_parent.statement_type,
        sta_parent.routine_name
order by sum(req.total_elapsed_time) desc
```
**View PLG$PROF_PSQL_STATS_VIEW**

```sql
select pstat.profile_id,
    pstat.statement_id,
    sta.statement_type,
    sta.package_name,
    sta.routine_name,
    sta.parent_statement_id,
    sta_parent.statement_type parent_statement_type,
    sta_parent.routine_name parent_routine_name,
    (select sql_text
        from plg$prof_statements
        where profile_id = pstat.profile_id and
            statement_id = coalesce(sta.parent_statement_id, pstat.statement_id)
    ) sql_text,
    pstat.line_num,
    pstat.column_num,
    cast(sum(pstat.counter) as bigint) counter,
    min(pstat.min_elapsed_time) min_elapsed_time,
    max(pstat.max_elapsed_time) max_elapsed_time,
    cast(sum(pstat.total_elapsed_time) as bigint) total_elapsed_time,
    cast(sum(pstat.total_elapsed_time) / nullif(sum(pstat.counter), 0) as bigint) avg_elapsed_time
from plg$prof_psql_stats pstat
join plg$prof_statements sta
    on sta.profile_id = pstat.profile_id and
        sta.statement_id = pstat.statement_id
left join plg$prof_statements sta_parent
    on sta_parent.profile_id = sta.profile_id and
        sta_parent.statement_id = sta.parent_statement_id
group by pstat.profile_id,
    pstat.statement_id,
    sta.statement_type,
    sta.package_name,
    sta.routine_name,
    sta.parent_statement_id,
    sta_parent.statement_type,
    sta_parent.routine_name,
    pstat.line_num,
    pstat.column_num
order by sum(pstat.total_elapsed_time) desc
```

**View PLG$PROF_RECORD_SOURCE_STATS_VIEW**

```sql
select rstat.profile_id,
    rstat.statement_id,
    sta.statement_type,
    sta.package_name,
    sta.routine_name
```


```
sta.parent_statement_id,
sta_parent.statement_type parent_statement_type,
sta_parent.routine_name parent_routine_name,
(select sql_text
   from plg$prof_statements
   where profile_id = rstat.profile_id and
     statement_id = coalesce(sta.parent_statement_id, rstat.statement_id)
 ) sql_text,
 rstat.cursor_id,
 cur.name cursor_name,
 cur.line_num cursor_line_num,
 cur.column_num cursor_column_num,
 rstat.record_source_id,
 recsrc.parent_record_source_id,
 recsrc.access_path,
 cast(sum(rstat.open_counter) as bigint) open_counter,
 min(rstat.open_min_elapsed_time) open_min_elapsed_time,
 max(rstat.open_max_elapsed_time) open_max_elapsed_time,
 cast(sum(rstat.open_total_elapsed_time) as bigint) open_total_elapsed_time,
 cast(sum(rstat.open_total_elapsed_time) / nullif(sum(rstat.open_counter), 0) as bigint) open_avg_elapsed_time,
 cast(sum(rstat.fetch_counter) as bigint) fetch_counter,
 min(rstat.fetch_min_elapsed_time) fetch_min_elapsed_time,
 max(rstat.fetch_max_elapsed_time) fetch_max_elapsed_time,
 cast(sum(rstat.fetch_total_elapsed_time) as bigint) fetch_total_elapsed_time,
 cast(sum(rstat.fetch_total_elapsed_time) / nullif(sum(rstat.fetch_counter), 0) as bigint) fetch_avg_elapsed_time,
 cast(coalesce(sum(rstat.open_total_elapsed_time), 0) +
   coalesce(sum(rstat.fetch_total_elapsed_time), 0) as bigint)
open_fetch_total_elapsed_time
   from plg$prof_record_source_stats rstat
 join plg$prof_cursors cur
  on cur.profile_id = rstat.profile_id and
     cur.statement_id = rstat.statement_id and
     cur.cursor_id = rstat.cursor_id
 join plg$prof_record_sources recsrc
  on recsrc.profile_id = rstat.profile_id and
     recsrc.statement_id = rstat.statement_id and
     recsrc.cursor_id = rstat.cursor_id and
     recsrc.record_source_id = rstat.record_source_id
 join plg$prof_statements sta
  on sta.profile_id = rstat.profile_id and
     sta.statement_id = rstat.statement_id
 left join plg$prof_statements sta_parent
  on sta_parent.profile_id = sta.profile_id and
     sta_parent.statement_id = sta.parent_statement_id
 group by rstat.profile_id,
    rstat.statement_id,
    sta.statement_type,
    sta.package_name,
    sta.routine_name,
```
RDB$BLOB_UTIL package

Adriano dos Santos Fernandes

Tracker ticket: #281

This package provides procedures and functions to manipulate BLOBs in a way that standard Firebird functions, like BLOB_APPEND and SUBSTRING, cannot do or are very slow.

These routines operate on binary data directly, even for text BLOBs.

Package routines

Function NEW_BLOB

RDB$BLOB_UTIL.NEW_BLOB is used to create a new BLOB. It returns a BLOB suitable for data appending, like BLOB_APPEND does.

The advantage over BLOB_APPEND is that it's possible to set custom SEGMENTED and TEMP_STORAGE options.

BLOB_APPEND always creates BLOB in temporary storage, which may not always be the best approach if the created BLOB is going to be stored in a permanent table, as it will require copy.

The BLOB returned from this function, even when TEMP_STORAGE = FALSE, may be used with BLOB_APPEND for appending data.

Input parameter(s):

• SEGMENTED type BOOLEAN NOT NULL
• TEMP_STORAGE type BOOLEAN NOT NULL

Return type: BLOB NOT NULL.

Function OPEN_BLOB

RDB$BLOB_UTIL.OPEN_BLOB is used to open an existing BLOB for read. It returns a handle (an integer
bound to the transaction) suitable for use with other functions of this package, like \texttt{SEEK}, \texttt{READ\_DATA} and \texttt{CLOSE\_HANDLE}.

Input parameter(s):
- \texttt{BLOB} type \texttt{BLOB NOT NULL}

Return type: \texttt{INTEGER NOT NULL}.

\textbf{Function IS\_WRITABLE}

\texttt{RDB\$BLOB\_UTIL.IS\_WRITABLE} returns \texttt{TRUE} when \texttt{BLOB} is suitable for data appending without copying using \texttt{BLOB\_APPEND}.

Input parameter(s):
- \texttt{BLOB} type \texttt{BLOB NOT NULL}

Return type: \texttt{BOOLEAN NOT NULL}.

\textbf{Function READ\_DATA}

\texttt{RDB\$BLOB\_UTIL.READ\_DATA} is used to read chunks of data of a \texttt{BLOB} handle opened with \texttt{RDB\$BLOB\_UTIL.OPEN\_BLOB}. When the \texttt{BLOB} is fully read and there is no more data, it returns \texttt{NULL}.

If \texttt{LENGTH} is passed with a positive number, it returns a \texttt{VARBINARY} with its maximum length.

If \texttt{LENGTH} is \texttt{NULL} it returns just a segment of the \texttt{BLOB} with a maximum length of 32765.

Input parameter(s):
- \texttt{HANDLE} type \texttt{INTEGER NOT NULL}
- \texttt{LENGTH} type \texttt{INTEGER}

Return type: \texttt{VARBINARY(32767)}.

\textbf{Function SEEK}

\texttt{RDB\$BLOB\_UTIL.SEEK} is used to set the position for the next \texttt{READ\_DATA}, it returns the new position.

\texttt{MODE} may be \texttt{0} (from the start), \texttt{1} (from current position) or \texttt{2} (from end).

When \texttt{MODE} is \texttt{2}, \texttt{OFFSET} should be zero or negative.

Input parameter(s):
- \texttt{HANDLE} type \texttt{INTEGER NOT NULL}
- \texttt{MODE} type \texttt{INTEGER NOT NULL}
- \texttt{OFFSET} type \texttt{INTEGER NOT NULL}

Return type: \texttt{INTEGER NOT NULL}. 
**Procedure CANCEL_BLOB**

RDB$BLOB_UTIL.CANCEL_BLOB is used to immediately release a temporary BLOB, like one created with BLOB_APPEND.

Note that if the same BLOB is used after cancel, using the same variable or another one with the same BLOB id reference, invalid blob id error will be raised.

**Procedure CLOSE_HANDLE**

RDB$BLOB_UTIL.CLOSE_HANDLE is used to close a BLOB handle opened with RDB$BLOB_UTIL.OPEN_BLOB.

Handles which are not explicitly closed are only closed automatically when the transaction ends.

Input parameter(s):

- HANDLE type INTEGER NOT NULL

**Examples**

*Create a BLOB in temporary space and return it in EXECUTE BLOCK*

```execute block returns (b blob)
 as 
 begin 
   -- Create a BLOB handle in the temporary space. 
   b = rdb$blob_util.new_blob(false, true);

   -- Add chunks of data.
   b = blob_append(b, '12345');
   b = blob_append(b, '67');

   suspend;
 end
```

*Open a BLOB and return chunks of it with EXECUTE BLOCK*

```execute block returns (s varchar(10))
 as 
 declare b blob = '1234567';
 declare bhandle integer;
 begin 
   -- Open the BLOB and get a BLOB handle.
   bhandle = rdb$blob_util.open_blob(b);

   -- Get chunks of data as string and return.
   s = rdb$blob_util.read_data(bhandle, 3);
   suspend;
 end
```
suspend;

s = rdb$blob_util.read_data(bhandle, 3);
suspend;

-- Here EOF is found, so it returns NULL.
s = rdb$blob_util.read_data(bhandle, 3);
suspend;

-- Close the BLOB handle.
execute procedure rdb$blob_util.close_handle(bhandle);
end

Seek in a blob

set term !;

execute block returns (s varchar(10))
as
    declare b blob;
    declare bhandle integer;
begin
    -- Create a stream BLOB handle.
    b = rdb$blob_util.new_blob(false, true);

    -- Add data.
    b = blob_append(b, '0123456789');

    -- Open the BLOB.
    bhandle = rdb$blob_util.open_blob(b);

    -- Seek to 5 since the start.
    rdb$blob_util.seek(bhandle, 0, 5);
    s = rdb$blob_util.read_data(bhandle, 3);
suspend;

    -- Seek to 2 since the start.
    rdb$blob_util.seek(bhandle, 0, 2);
    s = rdb$blob_util.read_data(bhandle, 3);
suspend;

    -- Advance 2.
    rdb$blob_util.seek(bhandle, 1, 2);
    s = rdb$blob_util.read_data(bhandle, 3);
suspend;

    -- Seek to -1 since the end.
    rdb$blob_util.seek(bhandle, 2, -1);
    s = rdb$blob_util.read_data(bhandle, 3);
suspend;
Check if blobs are writable

create table t(b blob);

set term !;

execute block returns (bool boolean)
as
  declare b blob;
begin
  b = blob_append(null, 'writable');
  bool = rdb$blob_util.is_writable(b);
  suspend;

  insert into t (b) values ('not writable') returning b into b;
  bool = rdb$blob_util.is_writable(b);
  suspend;
end!

set term ;!
Chapter 4. Changes to the Firebird API and ODS

since Firebird 4.0 release

ODS (On-Disk Structure) Changes

New Minor ODS Number

Firebird 5.0 creates databases with an ODS (On-Disk Structure) version of 13.1. It can also work with databases created in ODS 13.0 (by Firebird 4.0), but some new features will be unavailable.

New System Tables

System tables added in ODS 13.1:

- RDB$KEYWORDS
  Virtual table that enumerates keywords used by the SQL parser
- MON$COMPILED_STATEMENTS
  Virtual table that reports compiled statements

New Columns in System Tables

Columns RDB$CONDITION_SOURCE and RDB$CONDITION_BLR were added to the system table RDB$INDICES, they belong to the partial indices feature.

Virtual table MON$ATTACHMENTS was extended with the new MON$SESSION_TIMEZONE column. Also, column MON$COMPiled_STATEMENT_ID was added to the system tables MON$STATEMENTS and MON$CALL_STACK. Virtual table SEC$GLOBAL_AUTH_MAPPING now has the new column SEC$DESCRIPTION.

Application Programming Interfaces

The wire protocol version for the Firebird 5.0 API is 18. Additions and changes are described in the sections below.

Main API Extensions

A number of new methods have been added to the following interfaces.

ResultSet

```java
void getInfo(Status status,
  uint itemsLength, const uchar* items,
  uint bufferLength, uchar* buffer);
```

Allows to query the cursor information. Currently, only one information request is supported, INF_RECORD_COUNT. INF_RECORD_COUNT returns the number of records cached by the scrollable cursor,
or -1 for a uni-directional (forward-only) cursor.

**Extensions to various getInfo() Methods**

**Statement::getInfo()**

The following actions were added:

- `isc_info_sql_exec_path_blr_bytes`  Execution path as BLR (binary format)
- `isc_info_sql_exec_path_blr_text`   Execution path as BLR (textual format)

**Services API Extensions**

**Support for parallel operations**

Added support for parallel operations.

The following options were added:

- `isc_spb_bkp_parallel_workers`  number of parallel workers for backup
- `isc_spb_res_parallel_workers`  number of parallel workers for restore
- `isc_spb_rpr_par_workers`       number of parallel workers for sweep

Examples of use of new parameters in `fbsvcmgr` utility (login and password were left out for brevity):

```
fbsvcmgr -action_backup -bkp_parallel_workers 4 <dbname> <backupname>
fbsvcmgr -action_restore -res_parallel_workers 4 <backupname> <dbname>
fbsvcmgr -action_repair -rpr_sweep_db -rpr_par_workers 4 <dbname>
```

**Support for gfix -upgrade**

Added support for minor ODS upgrade.

The following option was added:

- `isc_spb_rpr_upgrade_db` upgrade database

Example of use of new parameter in `fbsvcmgr` utility (login and password were left out for brevity):

```
fbsvcmgr -action_repair -rpr_upgrade_db <dbname>
```
Chapter 5. Reserved Words and Changes

New Keywords in Firebird 5.0

Non-reserved

LOCKED
TARGET
TIMEZONE_NAME
UNICODE_CHAR
UNICODE_VAL
Chapter 6. Configuration Additions and Changes

New configuration parameters:

**Parameters for Parallel Operations**

**MaxParallelWorkers**

Limits the total number of parallel workers that can be created within a single Firebird process for each attached database. Integer values in the range between 1 (no parallelism) and 64 are allowed. All other values are silently ignored and the default value of 1 is used.

- Workers are accounted for each attached database independently.

**ParallelWorkers**

Specifies the default number of parallel workers for a single task. Integer values in the range between 1 (no parallelism) and `MaxParallelWorkers` (see above) are allowed. All other values are silently ignored and the default value of 1 is used.

**Other Parameters**

**MaxStatementCacheSize**

Defines the maximum amount of memory used to cache unused DSQL compiled statements. Value of zero means no statement caching is used. Default value is 2 megabytes.

**OnDisconnectTriggerTimeout**

Configures a timeout (in seconds) that is applied to the ON DISCONNECT trigger execution. The trigger will be automatically cancelled by the engine after the specified time is passed. Value of zero ('0') means no timeout is set. Default value is 180 seconds.

**DefaultProfilerPlugin**

Specifies the default profiler plugin used to profile connections using the RDB$PROFILER package.

---

**Changed configuration parameters**

**WireCryptPlugin**

A new variant (using 64-bit internal counter rather than 32-bit) of the ChaCha#20 plugin was added. The new default value of this parameter is now ChaCha64, ChaCha, Arc4.
Removed configuration parameters:

**RemotePipeName**

This parameter was removed along with the removal of WNET (aka named pipes) protocol support for Windows.

**TcpLoopbackFastPath**

This parameter was removed because Microsoft discourages using the SIO_LOOPBACK_FAST_PATH socket option.

**Replication Configuration Additions and Changes**

**cascade_replication**

New parameter that specifies whether changes applied to the replica database will be also subject of further replication (if any configured). Default value is false (cascading is disabled).

**Allow macros in replication.conf**

Configuration file macros are now also supported in replication.conf.
Chapter 7. Security

Security enhancements in Firebird 5 include:

**System privilege PROFILE_ANY_ATTACHMENT**

New system privilege PROFILE_ANY_ATTACHMENT has been added to the engine.

When remote SQL profiling is used and the attachment being profiled is from a different user, the calling user must have this system privilege.

See more details in the SQL and PSQL profiler chapter.
Chapter 8. Data Definition Language (DDL)

Quick Links

- Support for partial indices
- COMMENT ON MAPPING

Support for partial indices

Dmitry Yemanov

Tracker ticket: #7257

This feature allows to index only a subset of table rows defined by the search condition specified during index creation.

Syntax rules:

```sql
CREATE [UNIQUE] [{ASC[ENDING] | DESC[ENDING]}] INDEX <index_name> ON <table_name>
 { (<column_list>) | COMPUTED [BY] ( <value_expression> ) }
WHERE <search_condition>
```

Examples:

```sql
-- 1.
CREATE INDEX IT1_COL ON T1 (COL) WHERE COL < 100;
SELECT * FROM T1 WHERE COL < 100;
-- PLAN (T1 INDEX (IT1_COL))

-- 2.
CREATE INDEX IT1_COL2 ON T1 (COL) WHERE COL IS NOT NULL;
SELECT * FROM T1 WHERE COL > 100;
PLAN (T1 INDEX IT1_COL2)

-- 3.
CREATE INDEX IT1_COL3 ON T1 (COL) WHERE COL = 1 OR COL = 2;
SELECT * FROM T1 WHERE COL = 2;
PLAN (T1 INDEX IT1_COL3)
```

Notes:

1. A partial index definition may include the UNIQUE specification. In this case, every key in the index is required to be unique. This allows to enforce uniqueness across some subset of table rows.

2. A partial index is usable only in the following cases:
The WHERE condition includes exactly the same boolean expression as the one defined for the index;

- The search condition defined for the index contains ORed boolean expressions and one of them is explicitly included in the WHERE condition;

- The search condition defined for the index specifies IS NOT NULL and the WHERE condition includes an expression on the same field that is known to ignore NULLs.

**COMMENT ON MAPPING**

Alex Peshkov

Tracker ticket: #7046

The *COMMENT ON* statement was extended to be able to add a comment to a *MAPPING*.

```sql
COMMENT ON MAPPING <mapping name> IS {<comment> | NULL};
```
Chapter 9. Data Manipulation Language (DML)

Quick Links

- SKIP LOCKED clause
- Support for WHEN NOT MATCHED BY SOURCE in the MERGE statement
- Support multiple rows for DML RETURNING
- Allow parenthesized query expressions
- Changes to literals
- New Expressions and Built-in Functions

SKIP LOCKED clause

Adriano dos Santos Fernandes

Tracker ticket: #7350

SKIP LOCKED can be used with SELECT ... WITH LOCK, UPDATE and DELETE statements. It makes the engine skip records locked by other transactions instead of wait on them or raise conflict errors.

This is very useful to implement work queues where one or more processes post work to a table and issue an event, while workers listen for events and read/delete items from the table. Using SKIP LOCKED multiple workers can get exclusive work items from the table without conflicts.

Syntax:

```sql
SELECT  
    [FIRST ...]  
    [SKIP ...]  
FROM <sometable>  
    [WHERE ...]  
    [PLAN ...]  
    [ORDER BY ...]  
    [{ ROWS ... } | {OFFSET ...} | {FETCH ...}]  
    [FOR UPDATE [OF ...]]  
    [WITH LOCK [SKIP LOCKED]]

UPDATE <sometable>  
    SET ...  
    [WHERE ...]  
    [PLAN ...]  
    [ORDER BY ...]  
    [ROWS ...]  
    [SKIP LOCKED]```
As it happens with subclauses FIRST/SKIP/ROWS/OFFSET/FETCH, record lock (and "skip locked" check) is done in between of skip (SKIP/ROWS/OFFSET/FETCH) and limit (FIRST/ROWS/OFFSET/FETCH) checks.

Examples:

- Prepare metadata

```sql
create table emails_queue (
    subject varchar(60) not null,
    text blob sub_type text not null
);
set term !;
create trigger emails_queue_ins after insert on emails_queue
as
begin
    post_event('EMAILS_QUEUE');
end!
set term ;!
```

- Sender application or routine

```sql
insert into emails_queue (subject, text)
values ('E-mail subject', 'E-mail text...');
commit;
```

- Client application

```
-- Client application can listen to event 'EMAILS_QUEUE' to actually send e-mails using this query:

delete from emails_queue
    rows 10
    skip locked
```
More than one instance of the application may be running, for example to load balance work.

**Support for WHEN NOT MATCHED BY SOURCE in the MERGE statement**

Adriano dos Santos Fernandes

Tracker ticket: #6681

Syntax:

```
<merge when> ::=  
  <merge when matched> |  
  <merge when not matched>  
  <merge when not matched by target> |  
  <merge when not matched by source>

<merge when not matched by target> ::=  
  WHEN NOT MATCHED [ BY TARGET ] [ AND <condition> ] THEN  
  INSERT [ <left paren> <column list> <right paren> ]  
  VALUES <left paren> <value list> <right paren>

<merge when not matched by source> ::=  
  WHEN NOT MATCHED BY SOURCE [ AND <condition> ] THEN  
  { UPDATE SET <assignment list> | DELETE }
```

<merge when not matched by target> is called when a source record matches no record in target. INSERT will change the target table.

<merge when not matched by source> is called when a target record matches no record in source. UPDATE or DELETE will change the target table.

Example:

```
MERGE  
  INTO customers c  
  USING new_customers nc  
  ON (c.id = nc.id)  
  WHEN MATCHED THEN  
    UPDATE SET name = cd.name  
  WHEN NOT MATCHED BY SOURCE THEN  
    DELETE
```
Support multiple rows for DML RETURNING

Adriano dos Santos Fernandes

Tracker ticket: #6815

In DSQL, the RETURNING clause is now able to return multiple rows for DML statements than can affect multiple rows.

See compatibility notes on RETURNING for more information.

Allow parenthesised query expressions

Adriano dos Santos Fernandes

Tracker ticket: #6740

The DML syntax was extended to allow a parenthesised query expression (select including order by, offset and fetch clauses, but without with clause) to occur where previously only a query specification (select without with, order by, offset and fetch clauses) was allowed.

This allows more expressive queries, especially in UNION statements, and offers more compatibility with statements generated by certain ORMs.

Using parenthesised query expressions comes at a cost, as they consume an additional query context compared to a plain query specification. The maximum number of query contexts in a statement is 255.

Example:

```sql
( 
    select emp_no, salary, 'lowest' as type 
    from employee 
    order by salary asc 
    fetch first row only 
) 
union all 
( 
    select emp_no, salary, 'highest' as type 
    from employee 
    order by salary desc 
    fetch first row only 
); 
```

Changes to literals
Full SQL standard character string literal syntax

Adriano dos Santos Fernandes

Tracker ticket: https://github.com/FirebirdSQL/firebird/issues/5589

The syntax of character string literals was changed to support the full SQL standard syntax. This means a literal can be “interrupted” by whitespace or a comment. This can be used, for example, to break up a long literal over several lines, or provide inline comments.

```
<character string literal> ::= 
    [ <introducer> <character set specification> ]
    <quote> [ <character representation>... ] <quote>
    [ { <separator> <quote> [ <character representation>... ] <quote> }... ]

<separator> ::= 
    { <comment> | <white space> }...
```


Examples:

```
-- whitespace between literal
select 'ab'
   'cd'
from RDB$DATABASE;
-- output: 'abcd'

-- comment and whitespace between literal
select 'ab' /* comment */ 'cd'
from RDB$DATABASE;
-- output: 'abcd'
```

Full SQL standard binary string literal syntax

Adriano dos Santos Fernandes

Tracker ticket: https://github.com/FirebirdSQL/firebird/issues/5588

The syntax of binary string literals was changed to support the full SQL standard syntax. This means a literal can contain spaces to separate hexadecimal characters, and it can be “interrupted” by whitespace or a comment. This can be used, for example, to make the hex string more readable by grouping characters, or to break up a long literal over several lines, or provide inline comments.

```
<binary string literal> ::= 
    X <quote> [ <space>... ] [ { <hexit> [ <space>... ] <hexit> [ <space>... ] }... ] <quote>
    [ { <separator> <quote> [ <space>... ] [ { <hexit> [ <space>... ] ... ]

Chapter 9. Data Manipulation Language (DML)

42
Examples

-- Group per byte (whitespace inside literal)
select _win1252 x'42 49 4e 41 52 59'
from RDB$DATABASE;
-- output: BINARY

-- whitespace between literal
select _win1252 x'42494e'
   '415259'
from RDB$DATABASE;
-- output: BINARY

The usage of the _win1252 introducer in above example is a non-standard extension and equivalent to an explicit cast to a CHAR of appropriate length with character set WIN1252.

New Expressions and Built-in Functions

UNICODE_CHAR and UNICODE_VAL

Adriano dos Santos Fernandes

UNICODE_CHAR

Returns the UNICODE character with the specified code point.

Syntax:

UNICODE_CHAR( <number> )

The argument to UNICODE_CHAR must be a valid UNICODE code point and not in the range of high/low surrogates (0xD800 to 0xDFFF), otherwise it throws an error.

Example:

select unicode_char(x) from y;

UNICODE_VAL

Returns the UNICODE code point of the first character of the specified string, or zero if the string is
empty.

Syntax:

```
UNICODE_VAL( <string> )
```

Example:

```
select unicode_val(x) from y;
```
Chapter 10. Monitoring & Command-line Utilities

Improvements and additions to the Firebird utilities continue.

Monitoring

New virtual tables:

RDB$KEYWORDS:

- RDB$KEYWORD_NAME: Keyword name
- RDB$KEYWORD_RESERVED: Whether keyword is a reserved word

MON$COMPILED_STATEMENTS:

- MON$COMPILED_STATEMENT_ID: Compiled statement ID
- MON$SQL_TEXT: Text of the SQL query
- MON$EXPLAINED_PLAN: Plan (in the explained form) of the SQL query
- MON$OBJECT_NAME: PSQL object name
- MON$OBJECT_TYPE: PSQL object type
- MON$PACKAGE_NAME: Package name of the PSQL object
- MON$STAT_ID: Runtime statistics ID (references MON$*_STATS tables)

New columns in the tables:

In MON$ATTACHMENTS:

- MON$SESSION_TIMEZONE: Actual timezone of the session

In MON$STATEMENTS:

- MON$COMPiled_STATEMENT_ID: Compiled statement ID (references MON$COMPiled_STATEMENTS)

In MON$CALL_STACK:

- MON$COMPiled_STATEMENT_ID: Compiled statement ID (references MON$COMPiled_STATEMENTS)

In SEC$GLOBAL_AUTH_MAPPING:

- SEC$DESCRIPTION: Textual description
**isql**

**Unify display of system procedures & packages with other system objects**

Alex Peshkov

Tracker ticket: #7411

The `SHOW SYSTEM` command of *isql* now lists system packages and their procedures.

> Functions of system packages are currently not shown. This is tracked by #7475.

**gbak**

**Parallel backup/restore**

Vlad Khorsun

Tracker tickets: #1783, #3374

A new command-line switch has been added to *gbak*: `-PARALLEL <N>`.

It defines how many parallel workers will be used for the requested task.

Usage examples:

```
gbak -b -par 4 -user <username> -pass <password> <dbname> <backupname>
```

```
gbak -r -par 4 -user <username> -pass <password> <backupname> <dbname>
```

**gfix**

**Parallel sweep**

Vlad Khorsun

Tracker tickets: #7447

A new command-line switch has been added to *gfix*: `-PARALLEL <N>`.

It defines how many parallel workers will be used for the requested task.

Usage example:

```
gfix -sweep -par 4 -user <username> -pass <password> <dbname>
```
ODS upgrade

Dmitry Yemanov

Tracker tickets: #7397

A new command-line switch has been added to gfix: -UPGRADE.

It allows to upgrade ODS of the database to the latest supported minor version (within the supported major version).

Usage example(s):

```
gfix -upgrade <dbname> -user <username> -pass <password>
```
Chapter 11. Compatibility Issues

This section lists features and modifications that might affect the way you have installed and used Firebird in earlier releases.

SQL

Changes that may affect existing SQL code:

Multi-row RETURNING behaviour

Client-side INSERT … SELECT, UPDATE, DELETE, MERGE and UPDATE OR INSERT queries containing the RETURNING clause may now return multiple records instead of raising error “multiple rows in singleton select” as it happened before.

These queries are now described as `isc_info_sql_stmt_select` during preparation, while in previous versions they were described as `isc_info_sql_stmt_exec_procedure`.

Singleton INSERT … VALUES statements, as well as positioned UPDATE and DELETE statements (i.e. the ones containing the WHERE CURRENT OF clause) preserve the existing behaviour, being described as `isc_info_sql_stmt_exec_procedure`. They also preserve the ability of being executed within a single protocol roundtrip to the server.

However, all these queries, if used in PSQL and the RETURNING clause is applied, are still treated as singleton.

Removal of WNET protocol

Network protocol WNET (a.k.a. Named Pipes, a.k.a. NetBEIU) previously supported on Windows platform is removed in Firebird 5.0. Those Windows users who operated with any WNET connection string (\server\dbname or wnet://server/dbname) should switch to INET (TCP) protocol instead (connection string server:dbname, server/port:dbname, inet://server/dbname, or inet://server:port/dbname).

Removal of QLI

Command-line utility QLI is removed in Firebird 5.0, in accordance with its deprecation announcement published in the Firebird 4.0 release notes.
Chapter 12. Bugs Fixed

Firebird 5.0 Beta 1 Release: Bug Fixes

This sections enumerates only bugfixes not already fixed in priorly released Firebird versions.

Core Engine

#7422 — Seek in temporary blob level 0 makes read return wrong data

Fixed by Adriano dos Santos Fernandes

#7388 — Different invariants optimization between views and CTEs

Fixed by Dmitry Yemanov

#7304 — Events in system attachments (like garbage collector) are not traced

Fixed by Alex Peshkov

#7227 — Dependencies of subroutines are not preserved after backup restore

Fixed by Adriano dos Santos Fernandes

#7220 — TYPE OF COLUMN dependency not tracked in package header and external routines

Fixed by Adriano dos Santos Fernandes

#7183 — Regression when derived table has column evaluated as result of subquery with IN, ANY or ALL predicate: “invalid BLR at offset … / context already in use”

Fixed by Adriano dos Santos Fernandes

#7164 — Multi-way hash/merge joins are impossible for expression-based keys

Fixed by Dmitry Yemanov

#7133 — ORDER BY for big (>34 digits) int128 values is broken when index on that field is used
Chapter 12. Bugs Fixed

Fixed by Alex Peshkov

[#7077] — EXECUTE BLOCK (without RETURNS) do not work with batches
Fixed by Adriano dos Santos Fernandes

[#7009] — IReplicatedTransaction receives wrong savepoint event
Fixed by Dmitry Sibiryakov, Dmitry Yemanov

[#6942] — Incorrect singleton error with MERGE and RETURNING
Fixed by Adriano dos Santos Fernandes

[#6869] — Domain CHECK-expression can be ignored when we DROP objects that are involved in it
Fixed by Adriano dos Santos Fernandes

[#6807] — Regression: error "Unexpected end of command" with incorrect line/column info
Fixed by Adriano dos Santos Fernandes

[#5749] — "Token unknown" error on formfeed in query
Fixed by Adriano dos Santos Fernandes

[#3812] — Query with a stored procedure doesn’t accept explicit plan
Fixed by Dmitry Yemanov

[#3218] — Optimizer fails applying stream-local predicates before merging
Fixed by Dmitry Yemanov

Server Crashes/Hangups

[#7195] — Crash when accessing already cleared memory in the sorting module
Utilities

gbak

#7436 — Backup error for wide table

Fixed by Alex Peshkov
**Chapter 13. Firebird 5.0 Project Teams**

*Table 1. Firebird Development Teams*

<table>
<thead>
<tr>
<th>Developer</th>
<th>Country</th>
<th>Major Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dmitry Yemanov</td>
<td>Russian Federation</td>
<td>Full-time database engineer/implementor; core team leader</td>
</tr>
<tr>
<td>Alexander Peshkov</td>
<td>Russian Federation</td>
<td>Full-time security features coordinator; buildmaster; porting authority</td>
</tr>
<tr>
<td>Vladyslav Khorsun</td>
<td>Ukraine</td>
<td>Full-time DB engineer; SQL feature designer/implementor</td>
</tr>
<tr>
<td>Adriano dos Santos Fernandes</td>
<td>Brazil</td>
<td>International character-set handling; text and text BLOB enhancements; new DSQL features; code scrutineering</td>
</tr>
<tr>
<td>Roman Simakov</td>
<td>Russian Federation</td>
<td>Engine contributions</td>
</tr>
<tr>
<td>Dimitry Sibiryakov</td>
<td>Czech Republic</td>
<td>Engine and replication contributions</td>
</tr>
<tr>
<td>Ilya Eremin</td>
<td>Russian Federation</td>
<td>Engine contributions</td>
</tr>
<tr>
<td>Paul Beach</td>
<td>France</td>
<td>Release Manager; MacOS Builds;</td>
</tr>
<tr>
<td>Pavel Cisar</td>
<td>Czech Republic</td>
<td>QA tools designer/coordinator; Firebird Butler coordinator; Python driver developer</td>
</tr>
<tr>
<td>Pavel Zotov</td>
<td>Russian Federation</td>
<td>QA tester and tools developer</td>
</tr>
<tr>
<td>Paul Reeves</td>
<td>France</td>
<td>Windows installers and builds</td>
</tr>
<tr>
<td>Mark Rotteveel</td>
<td>The Netherlands</td>
<td>Jaybird implementer and co-coordinator; Documentation writer</td>
</tr>
<tr>
<td>Jiri Cincura</td>
<td>Czech Republic</td>
<td>Developer and coordinator of .NET providers</td>
</tr>
<tr>
<td>Martin Koeditz</td>
<td>Germany</td>
<td>Developer and coordinator of PHP driver Documentation translator</td>
</tr>
<tr>
<td>Alexey Kovyazin</td>
<td>Russian Federation</td>
<td>Website coordinator</td>
</tr>
<tr>
<td>Helen Borrie</td>
<td>Australia</td>
<td>Release notes editor; Chief of Thought Police</td>
</tr>
</tbody>
</table>
Appendix A: Licence Notice

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

The Original Documentation is entitled Firebird 5.0 Release Notes.

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

Copyright © 2004-2023. All Rights Reserved. Initial Writer contact: helebor at users dot sourceforge dot net.