Firebird 3.0 Release Notes

Helen Borrie (Collator/Editor)

28 April 2016 - Document v.0300-38 - for Firebird 3.0.0 Release
# Table of Contents

1. General Notes ................................................................. 1  
   Bug Reporting ................................................................. 1  
   Documentation ................................................................. 1  
2. New In Firebird 3.0 .......................................................... 2  
   Summary of Features ......................................................... 2  
   Compatibility with Older Versions ..................................... 4  
3. Changes in the Firebird Engine ......................................... 5  
   Remodelled Architecture .................................................. 5  
   Server Modes ................................................................. 6  
   Providers ................................................................. 6  
   Connection String and Protocols ...................................... 9  
   Plug-Ins ................................................................. 12  
   External Engines .......................................................... 17  
   Optimizer Improvements .................................................. 20  
   Other Optimizations ......................................................... 20  
   Remote Interface/Network Protocol .................................... 20  
   Miscellaneous Improvements ........................................... 21  
   Connections Limit Raised ............................................... 21  
   Better Error Diagnosis .................................................... 21  
   ICU Version Upgraded ................................................... 21  
   Internal Debug Info Made Human-readable ..................... 21  
   A Silly Message is Replaced ............................................. 21  
   New Pseudocolumn RDB$RECORD_VERSION .................... 21  
   systemd init Scripts ....................................................... 22  
4. Changes to the Firebird API and ODS ................................. 23  
   ODS (On-Disk Structure) Changes ..................................... 23  
   New ODS Number .......................................................... 23  
   Implementation ID is Deprecated .................................... 23  
   Maximum Database Size .................................................. 24  
   Maximum Page Size ....................................................... 24  
   Maximum Number of Page Buffers in Cache .................... 24  
   Extension of Transaction ID Space Limit ......................... 24  
   Limits Raised for Attachment and Statement IDs ............... 24  
   System Tables .............................................................. 24  
   Application Programming Interfaces ................................. 26  
   Interfaces and the New Object-oriented API ...................... 27  
   Other New APIs ........................................................... 28  
   API Improvements ........................................................ 29  
5. Reserved Words and Changes .......................................... 36  
   New Keywords in Firebird 3.0 ............................................ 36  
6. Configuration Additions and Changes ................................... 37  
   Scope of Parameters ....................................................... 37  
   Macro Substitution ......................................................... 37  
   Includes ................................................................. 38  
   Wildcards ................................................................. 38  
   Expression of Parameter Values ....................................... 39  
   “Per-database” Configuration ........................................... 39  
   Format of Configuration Entries ........................................ 39
 Parameters Available ................................................................................................................. 39
 New Parameters .......................................................................................................................... 40
 SecurityDatabase ....................................................................................................................... 40
 AuthServer and AuthClient ........................................................................................................... 40
 WireCrypt .................................................................................................................................. 41
 UserManager ................................................................................................................................. 41
 TracePlugin ................................................................................................................................. 41
 WireCryptPlugin ............................................................................................................................ 41
 KeyHolderPlugin .......................................................................................................................... 42
 Providers ...................................................................................................................................... 42
 ServerMode ................................................................................................................................. 42
 RemoteAccess ............................................................................................................................. 42
 WireCompression ........................................................................................................................ 42
 IPv6V6Only .................................................................................................................................. 43

 Parameters Changed or Enhanced ............................................................................................. 44
 ExternalFileAccess ....................................................................................................................... 44

 Parameters Removed or Deprecated .......................................................................................... 44
 RootDirectory .............................................................................................................................. 44
 LegacyHash .................................................................................................................................. 44
 OldSetClauseSemantics ................................................................................................................ 45
 OldColumnNameing ...................................................................................................................... 45
 LockGrantOrder .......................................................................................................................... 45

Obsolete Windows priority settings .......................................................................................... 45

 7. Security ..................................................................................................................................... 46
 Location of User Lists .................................................................................................................. 46
 Creating an Alternative Security Database .................................................................................. 46
 Database Encryption ................................................................................................................... 47
 Secret Key ..................................................................................................................................... 47
 Encryption Tasks .......................................................................................................................... 47

 New Authentication Method in Firebird 3 .................................................................................... 48
 SSL/TLS Support .......................................................................................................................... 49
 Increased Password Length .......................................................................................................... 49
 The Authentication Plug-in ......................................................................................................... 49
 Multiple User Managers ............................................................................................................. 50

 "Over the wire" Connection Encryption ....................................................................................... 51
 The Secret Session Key ................................................................................................................. 51

 Mapping of Users to Objects ..................................................................................................... 51
 The Mapping Rule ....................................................................................................................... 52
 Syntax for MAPPING Objects ....................................................................................................... 53
 Legacy Mapping Rule .................................................................................................................... 55

 International Character Sets for User Accounts ......................................................................... 55
 Conditions for Non-ASCII Logins ................................................................................................. 56
 SQL Features for Managing Access ............................................................................................. 57
 SQL-driven User Management ..................................................................................................... 57
 SET ROLE .................................................................................................................................... 59
 GRANT/REVOKE Rights GRANTED BY Specified User .............................................................. 61
 REVOKE ALL ON ALL ............................................................................................................... 61
 User Privileges for Metadata Changes ....................................................................................... 62
 GRANT EXECUTE Privileges for UDFs ....................................................................................... 63
 Improvement for Recursive Stored Procedures .......................................................................... 63
 Privileges to Protect Other Metadata Objects .............................................................................. 63
 Pseudo-Tables with List of Users ............................................................................................... 64
13. Bugs Fixed ......................................................................................................................... 124
   Firebird 3.0.0 Release: Bug Fixes .......................................................................................... 124
      Core Engine .................................................................................................................. 124
      Server Crashes ............................................................................................................ 126
      Utilities ........................................................................................................................ 126
      Builds ............................................................................................................................ 127
   Firebird 3.0 Release Candidate 2: Bug Fixes and Minor Improvements ................................ 128
      Core Engine .................................................................................................................. 129
      Server Crashes ............................................................................................................ 132
      Database Encryption ..................................................................................................... 133
      API/Remote Interface ................................................................................................. 134
      Utilities ........................................................................................................................ 134
      Builds and Installers ..................................................................................................... 135
   Firebird 3.0 Release Candidate 1: Bug Fixes and Minor Improvements ................................ 136
      Core Engine .................................................................................................................. 137
      POSIX-Specific ............................................................................................................. 141
      Windows-Specific ........................................................................................................ 142
      Services API ................................................................................................................. 142
      Utilities ........................................................................................................................ 142
   Firebird 3.0 Second Beta Release: Bug Fixes ....................................................................... 143
      Core Engine .................................................................................................................. 144
      Server Crashes ............................................................................................................ 151
      API/Remote Interface ................................................................................................. 152
      Utilities ........................................................................................................................ 153
   Firebird 3.0 Second Beta Release: Improvements .................................................................. 154
   Firebird 3.0 First Beta Release .............................................................................................. 156
      Core Engine .................................................................................................................. 156
      Server Crashes ............................................................................................................ 162
      API/Remote Interface ................................................................................................. 163
      Utilities ........................................................................................................................ 163
   Firebird 3.0 Second Alpha Release ...................................................................................... 165
      Core Engine .................................................................................................................. 165
      Server Crashes ............................................................................................................ 168
      API/Remote Interface ................................................................................................. 168
      Security/User Management .......................................................................................... 169
      Procedural Language ................................................................................................... 169
      Data Definition Language ........................................................................................... 170
      Data Manipulation Language ....................................................................................... 170
      Utilities ........................................................................................................................ 171
      International Language Support .................................................................................. 173
      Installation Issues ........................................................................................................ 173
   Firebird 3.0 First Alpha Release ............................................................................................ 173
      Core Engine .................................................................................................................. 173
      API/Remote Interface ................................................................................................. 178
      Procedural Language ................................................................................................... 178
Data Definition Language ................................................................................................... 179
Data Manipulation Language & DSQL .............................................................................. 180
Command-line Utilities ...................................................................................................... 182
International Language Support ....................................................................................... 183
14. Firebird 3.0 Project Teams .......................................................................................... 184
Appendix A: Licence Notice .............................................................................................. 186
List of Tables

3.1. Matrix of Server Modes ................................................................. 6
6.1. Parameters available in databases.conf ........................................ 40
11.1. Arguments for *gbak* STATISTICS Output ................................. 110
14.1. Firebird Development Teams ................................................... 184
Chapter 1

General Notes

Thank you for choosing Firebird 3.0. We cordially invite you to test it hard against your expectations and engage with us in identifying and fixing any bugs you might encounter.

If you are upgrading from a previous major release version, be sure to study Chapter 12, *Compatibility Issues* before you attempt to do anything to your existing databases!

Bug Reporting

- If you think you have discovered a new bug in this release, please make a point of reading the instructions for bug reporting in the article [How to Report Bugs Effectively](http://example.com), at the Firebird Project website.

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

Follow these guidelines as you attempt to analyse your bug:

1. Write detailed bug reports, supplying the exact build number of your Firebird kit. Also provide details of the OS platform. Include reproducible test data in your report and post it to our [Tracker](http://example.com).

2. You are warmly encouraged to make yourself known as a field-tester of this pre-release by subscribing to the [field-testers' list](http://example.com) and posting the best possible bug description you can.

3. If you want to start a discussion thread about a bug or an implementation, please do so by subscribing to the [firebird-devel list](http://example.com). In that forum you might also see feedback about any tracker ticket you post regarding this Beta.

Documentation

You will find all of the README documents referred to in these notes—as well as many others not referred to—in the doc sub-directory of your Firebird 3.0 installation.

--The Firebird Project
Chapter 2

New In Firebird 3.0

The primary goals for Firebird 3 were to unify the server architecture and to improve support for SMP and multiple-core hardware platforms. Parallel objectives were to improve threading of engine processes and the options for sharing page cache across thread and connection boundaries.

Alongside these aims came new strategies to improve performance, query optimization, monitoring and scalability and to address the demand for more security options. A number of popular features were introduced into the SQL language, including the long-awaited support for the Boolean data type and the associated logical predications.

Summary of Features

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

Unification of the Firebird executable is complete

With the completion of true SMP support for Superserver, the Firebird core is now a unified library that supports a single ODS, loadable either as an embedded engine or by the “network listener” executable. Choice of server model is determined by settings for a new configuration parameter ServerMode, defining the locking and cache modes. It is specified at global level in firebird.conf.

By default, ServerMode = Super (alias ThreadedDedicated), i.e., SuperServer.

Note

The previous aliases.conf is replaced by databases.conf, now including not just aliases for databases but also (optionally) configuration parameters to enable configuration of databases and/or alternative security databases individually.

The changes are described in more detail in the chapter Changes in the Firebird Engine.

True SMP support for SuperServer

In Superserver mode, the engine now makes use of multiple CPUs and cores when spawning connections.

Tracker: CORE-775

Implemented by V. Khorsun

New, object-oriented C++ APIs

Object-oriented C++ APIs enable external code routines to plug in and run safely inside Firebird engine space, including (but not limited to):

- Encryption schemes for data
- User authentication schemes, including secure key exchange
• Eventually, plug-in support for stored procedures, triggers and functions written in Java, C++, Object-Pascal, etc.

“Per-Database” Configuration

Custom configuration at database level can now be achieved with formal entries in databases.conf (formerly aliases.conf).

Increased Limits in Several Areas

Transaction IDs, attachment IDs, statement IDs all have increased maximum values. Maximum number of page buffers in cache is increased for 64-bit servers. Maximum database size is increased. For details, see Chapter 4, Changes to the API and ODS.

Multiple Security Databases

Firebird now supports user access control via more than one security database on the server. Each database can be configured, using the parameter SecurityDatabase in databases.conf, to use a specific database other than the default security3.fdb. The user structures may even be defined within the user database itself.

Note

The flag MON$SEC_DATABASE was added to the monitoring table MON$DATABASE to assist in determining what type of security database is used—Default, Self or Other.

Several New SQL Commands to Manage Users and Access

Changes in architecture, stiffening of rules for security and data integrity, along with feature requests, have given rise to a raft of new SQL commands for managing users and their access to objects.

International Characters in User Credentials

Provided the new authentication provisions are used, the system can accept user names and passwords containing non-ASCII characters. See International Character Sets for User Accounts in the Security chapter.

New Data Type Support

A true BOOLEAN type (True/False/Unknown), complete with support for logical predicates, e.g.,

```
UPDATE ATABLE
SET MYBOOL = (COLUMN1 IS DISTINCT FROM COLUMN2)
```

For details, see BOOLEAN Type.

IDENTITY type, spawning unique identifiers for the defined column from an internal generator. For details, see IDENTITY-Style Column.

Manage NULL/NOT NULL Column/Domain Attribute

The NOT NULL attribute on a column or domain can now be managed using the ALTER TABLE or ALTER DOMAIN syntax, respectively. For details, see Manage Nullability in Domains and Columns.

Support for SQL Packages

For details, refer to Packages.

DDL Triggers

Now, triggers can be written to execute when database objects are modified or deleted. A typical use is to block unauthorised users from performing these tasks.

For details, refer to DDL Triggers.
New In Firebird 3.0

'Window' functions in DML
A whole new series of analytical functions to work with multiple subsets in DML. See Window (Analytical) Functions.

Statistical functions
A suite of statistical functions returning values for a variety of variance, standard deviation and linear regression formulae. See Statistical Functions.

Scrollable Cursors
The query engine now supports bi-directional (“scrollable”) cursors, enabling both forward and backward navigation in PSQL and in DSQL with support from the API. See Scrollable (Bi-directional) Cursor Support.

SQL:2008-Compliant OFFSET and FETCH Clauses
Support implemented for SQL-2008-compliant OFFSET and FETCH clauses as an alternative for {FIRST and SKIP} or {ROWS and TO} clauses. See SQL:2008-Compliant OFFSET and FETCH Clauses.

IPv6 Support
Firebird 3 can use IPv6 connections on both client and server sides. See the notes for the new configuration parameter IPv6V6Only.

Validation Whilst Database is On-line
On-line validation, first implemented in Firebird 2.5.4, has been ported forward to Firebird 3.0. See Perform Some Validation Services On-line.

Run-time Statistics in gbak Verbose Output
Verbose output from gbak can now include run-time statistics, reporting times elapsed, page reads and page writes. Also supported in the Services API.

Compatibility with Older Versions
A series of notes about compatibility with older Firebird versions is collated in Chapter 12, “Compatibility Issues”. Included there are instructions for initializing the security database, if it was not done by your installer kit, and for configuring the server for the legacy style of authentication.
In Firebird 3, the remodelling of the architecture that was begun in v.2.5 was completed with the implementation of full SMP support for the Superserver model. In the new scheme, it is possible to configure the execution model individually per database.

## Remodelled Architecture

Dmitry Yemanov

The remodelled architecture integrates the core engine for Classic/Superclassic, Superserver and embedded models in a common binary. The cache and lock behaviours that distinguish the execution models are now determined externally by the settings in the new configuration parameter `ServerMode`. The connection method is determined by the order and content of another parameter, `Providers` and the connection protocol that is deduced at run-time from the connection string supplied when a client requests an attachment.

The parameters for configuring the architecture are specified globally (in `firebird.conf`). `Providers` can be overridden specifically for a database (in `databases.conf`).

### Note

`databases.conf` is the old `aliases.conf` from previous versions, with a new name. In Firebird 3, the role of this file involves (potentially) much more than being just a lookup for database file paths. For more details about what can be configured at database level, refer to the chapter Configuration Additions and Changes.
**Server Modes**

Table 3.1. Matrix of Server Modes

<table>
<thead>
<tr>
<th>ServerMode</th>
<th>Synonym</th>
<th>Resource Model</th>
<th>Provider[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super</td>
<td>ThreadedDedicated</td>
<td>Database is opened exclusively by a single server process. User attachments are processed by threads launched from the common pool and all share a single database page cache inside the process. <em>This is the installation default.</em></td>
<td>Remote, Engine12(^1), Loopback</td>
</tr>
<tr>
<td>Superclassic</td>
<td>ThreadedShared</td>
<td>Databases are opened by a single server process, but access is not exclusive: an embedded process can open the same database concurrently. User attachments are processed by threads launched from the common pool, each having its own database page cache.</td>
<td>Remote, Engine12, Loopback</td>
</tr>
<tr>
<td>Classic</td>
<td>MultiProcess</td>
<td>A separate process is started for each attachment to server. A database may be opened by multiple Classic processes, including local processes for embedded access. Each process has its own database page cache.</td>
<td>Remote, Engine12, Loopback</td>
</tr>
</tbody>
</table>

\(^1\) **Only if exclusive access is available**

**Providers**

The providers are more or less what we traditionally thought of as the methods used to connect a client to a server, that is to say, across a network, host-locally, via the local loopback (“localhost”) or by a more direct local connection (the old libfbembed.so on POSIX, now implemented as the plug-in library libEngine1 2.so; on Windows, engine12.dll; on MacOSX, engine12.dylib).

- In *firebird.conf*, all are available by default, as follows:

  ```
  #Providers = Remote,Engine12,Loopback
  ```

- In *databases.conf*, one or more providers can be blocked by pasting the line from *firebird.conf*, uncommenting it, and deleting the unwanted provider[s].

**The Providers Architecture**

Alex Peshkov
Although a key feature of Firebird 3, the Providers architecture is not new. Providers existed historically in Firebird’s predecessors and, though well hidden, are present in all previous versions of Firebird. They were introduced originally to deal with a task that has been performed since then by “interface layers” such as ODBC, ADO, BDE and the like, to enable access to different database engines using a single external interface.

Subsequently, this Providers architecture (known then as Open Systems Relational Interface, OSRI) also showed itself as very efficient for supporting a mix of old and new database formats—different major on-disk structure versions—on a single server having mixed connections to local and remote databases.

The providers implemented in Firebird 3 make it possible to support all these modes (remote connections, databases with differing ODS, foreign engines) as well as chaining providers. Chaining is a term for a situation where a provider is using a callback to the standard API when performing an operation on a database.

**The Components**

The main element of the Providers architecture is the y-valve. On the initial attach or create database call y-valve scans the list of known providers and calls them one by one until one of them completes the requested operation successfully. For a connection that is already established, the appropriate provider is called at once with almost zero overhead.

Let’s take a look at some samples of y-valve operation when it selects the appropriate provider at the attach stage. These use the default configuration, which contains three providers:

- **Remote** (establish network connection)
- **Engine12** (main database engine)
- **Loopback** (force network connection to the local server for <database name> without an explicit network protocol being supplied).

The typical client configuration works this way: when one attaches to a database called RemoteHost:dbname (TCP/IP syntax) or \RemoteHost\dbname (NetBios) the Remote provider detects explicit network protocol syntax and, finding it first in the Provider list, redirects the call to RemoteHost. When <database name> does not contain a network protocol but just the database name, the Remote provider rejects it and the Engine12 provider comes to the fore and tries to open the named database file. If it succeeds, we get an embedded connection to the database.

**Note**

A special “embedded library” is no longer required. To make the embedded connection, the standard client loads the appropriate provider and becomes an embedded server.

**Failure Response**

But what happens if the engine returns an error on an attempt to attach to a database?

- If the database file to be attached to does not exist there is no interest at all.
- An embedded connection may fail if the user attaching to it does not have enough rights to open the database file. That would be the normal case if the database was not created by that user in embedded mode or if he was not explicitly given OS rights for embedded access to databases on that box.
Note

Setting access rights in such a manner is a requirement for correct Superserver operation.

• After a failure of Engine12 to access the database, the Loopback provider is attempted for an attach. It is not very different to Remote except that it tries to access the named database <dbname> on a server running a TCP/IP local loopback.

On Windows, XNET is tried first, then TCP/IP loopback (with localhost: prepended to <dbname>), then Named Pipes (NetBEUI) loopback (with \ prepended). The server may be started with XNET (or any other protocol) disabled, so we try all the options. On POSIX only TCP/IP protocol is supported, other options are not available.

If the attachment succeeds, a remote-like connection is established with the database even though it is located on the local machine.

Other Providers

Use of providers is not limited to the three standard ones. Firebird 3 does not support pre-ODS 12 databases. Removing support for old formats from the engine helps to simplify its code and gain a little speed. Taking into account that this speed gain sometimes takes place in performance-critical places, like searching a key in an index block, avoiding old code and related branches really does make Firebird fly faster.

Nevertheless, the Providers architecture does make it possible to access old databases when changing to a higher version of Firebird. A suitable provider may be considered for inclusion in a later sub-release.

Custom Providers

A strong feature of the Providers architecture is ability for the deployer to add his own providers to the server, the client, or both.

So what else might be wanted on a client, other than a remote connection? Recall Provider chaining that was mentioned earlier. Imagine a case where a database is accessed via very slow network connection, say something like 3G or, worse, GPRS. What comes to mind as a way to speed it up is to cache on the client some big tables that rarely change. Such systems were actually implemented but, to do it, one had to rename fbclient to something arbitrary and load it into its own library called fbclient, thus making it possible to use standard tools to access the database at the same time as caching required tables. It works but, as a solution, it is clearly not ideal.

With the Providers architecture, instead of renaming libraries, one just adds a local caching provider which can use any method to detect connections to it (something like a cache@ prefix at the beginning of the database name, or whatever else you choose).

In this example, when the database name cache@RemoteHost:dbname is used, the caching provider accepts the connection and invokes the y-valve once more with the traditional database name RemoteHost:dbname. When the user later performs any call to his database, the caching provider gets control of it before Remote does and, for a locally cached table, can forestall calls to the remote server.

Use of chaining allows a lot of other useful things to be implemented. An example might be MySQL-style replication at statement level without the need for triggers: just repeat the same calls for the replication host, perhaps when a transaction is committed. In this case, the chaining provider would be installed on the server, not the client, and no modification of the command line would be needed.
To avoid cycling when performing a callback to \texttt{y-valve} at attach time, such a provider can modify the list of providers using the \texttt{isc_dpb_config} parameter in the DPB. The same technique may be used at the client, too.

For details, see the Configuration Additions and Changes chapter.

The ability to access foreign database engines using providers should not be overlooked, either. It might seem strange to consider this, given the number of tools available for this sort of task. Think about the ability to access other Firebird databases using EXECUTE STATEMENT, that became available in Firebird 2.5. With a provider to ODBC or other common tool to access various data sources it is within reach to use EXECUTE STATEMENT to get direct access from procedures and triggers, to data from any database having a driver for the chosen access tool. It is even possible to have a provider to access some particular type of foreign database engine if there is some reason to want to avoid the ODBC layer.

Providers Q & A

Q. Interfaces and providers are probably very good, but I have an old task written using plain API functions and for a lot of reasons I can't rewrite it in the near future. Does it mean I will have problems migrating to Firebird 3?

• A. Definitely no problems. The old API is supported for backward compatibility in Firebird 3 and will be supported in future versions as long as people need it.

And what about performance when using the old API?

• A. The functional API is implemented as a very thin layer over interfaces. Code in most cases is trivial: convert passed handles to pointers to interfaces—hitherto referred to as “handle validation”—and invoke the appropriate function from the interface.

Functions that execute an SQL operation and fetch data from it are one place where coding is a little more complex, involving the SQLDA construct. The data moves related to the SQLDA have always created an overhead. The logic between the new and old APIs does not add significantly to that old overhead.

Connection String and Protocols

\textit{Connection string} refers to the local or remote path to the database to which a client requests an attachment (connection). The syntax of the connection string determines the transport protocol by which clients and the server communicate. The legacy syntaxes for the available protocols, supported by all Firebird versions, are as follows:

\textbf{For TCP/IP (aka INET) protocol:}

\begin{verbatim}
<host> [ / <port>] : <database file path or alias>
\end{verbatim}

\textbf{For named pipes (aka NetBEUI, aka WNET) protocol:}

\begin{verbatim}
\backslash \backslash <host> [ @ <port>] \ <database file path or alias>
\end{verbatim}
For local connections, simply:

<database file path or alias>

Local connection is implied if <host> is omitted. Depending on settings, platform and Firebird version, it could be performed via either the embedded engine, XNET (shared memory) protocol or TCP/IP localhost loopback.

Examples

Connect via TCP/IP using database name:

192.168.0.11:/db/mydb.fdb
192.168.0.11:C:\db\mydb.fdb
myserver:C:\db\mydb.fdb
localhost:/db/mydb.fdb

Connect via TCP/IP using database alias:

192.168.0.11:mydb
myserver:mydb
localhost:mydb

Connect via TCP/IP using non-default port 3051:

192.168.0.11/3051:C:\db\mydb.fdb
192.168.0.11/3051:mydb
myserver/3051:/db/mydb.fdb
localhost/3051:/db/mydb.fdb
myserver/3051:mydb
localhost/3051:mydb

Connect via TCP/IP using non-default service name:

192.168.0.11/fb_db:C:\db\mydb.fdb
192.168.0.11/fb_db:mydb
localhost/fb_db:/db/mydb.fdb
myserver/fb_db:/db/mydb.fdb
myserver/fb_db:mydb
localhost/fb_db:mydb

Connect via named pipes (Windows only):

\\myserver\C:\db\mydb.fdb
\\myserver@fb_db\C:\db\mydb.fdb

Local connection:

/db/mydb.fdb
C:\db\mydb.fdb
mydb
**URL-Style Connection Strings**

Firebird 3.0 introduces an additional, generalized, URL-style syntax for connection strings. The pattern is:

```
```

<protocol> ::= INET | WNET | XNET

INET resolves to TCP/IP, WNET to Named Pipes, while XNET surfaces the old “Windows local protocol” (shared memory).

**Examples**

**Connect via TCP/IP using database name:**

inet://192.168.0.11//db/mydb.fdb
inet://192.168.0.11/C:\db\mydb.fdb
inet://myserver/C:\db\mydb.fdb
inet://localhost/db/mydb.fdb

**Connect via TCP/IP using database alias:**

inet://192.168.0.11/mydb
inet://myserver/mydb
inet://localhost/mydb

**Connect via TCP/IP using non-default port 3051:**

inet://192.168.0.11:3051/C:\db\mydb.fdb
inet://192.168.0.11:3051/mydb
inet://myserver:3051//db/mydb.fdb
inet://localhost:3051//db/mydb.fdb
inet://myserver:3051/mydb
inet://localhost:3051/mydb

**Connect via TCP/IP using non-default service name:**

inet://192.168.0.11:fb_db/C:\db\mydb.fdb
inet://192.168.0.11:fb_db/mydb
inet://localhost:fb_db//db/mydb.fdb
inet://myserver:fb_db//db/mydb.fdb
inet://myserver:fb_db/db/mydb.fdb
inet://localhost:fb_db/db/mydb

**Connect via named pipes:**

wnet://myserver/C:\db\mydb.fdb
wnet://myserver:fb_db/C:\db\mydb.fdb

**Loopback connection via TCP/IP:**

inet:///db/mydb.fdb
Loopback connection via named pipes:

wnet://C:\db\mydb.fdb
wnet://mydb

Local connection via shared memory:

xnet://C:\db\mydb.fdb
xnet://mydb

Local (embedded by default) connection:

/db/mydb.fdb
C:\db\mydb.fdb
mydb

Local connection is implied if <host> is omitted. Depending on settings, platform and Firebird version, it could be performed via either the embedded engine, XNET (shared memory) protocol or TCP/IP localhost loopback.

On the server side, the provider configuration is in the default order Remote, Engine12, Loopback. If the Remote provider fails to match the connection string because the protocol or host parts are missing, then Engine12, the embedded engine, handles it as a hostless connection. To connect locally using a specific transport protocol, it is necessary to specify that protocol:

inet://<database file path or alias>
or
wnet://<database file path or alias>
or
xnet://<database file path or alias>

**Note**

WNET (named pipes) and XNET (shared memory) protocols are available only on Windows.

**Plug-Ins**

Alex Peshkov

From version 3 onward, Firebird's architecture supports plug-ins. For a number of predefined points in the Firebird code, a developer can write his own fragment of code for execution when needed.

A plug-in is not necessarily one written by a third party: Firebird has a number of intrinsic plug-ins. Even some core parts of Firebird are implemented as plug-ins.

**What is a Plug-In?**

The term “plug-in” is used to name related but different things:
• a dynamic library, containing code to be loaded as a plug-in (often called a plug-in module) and stored in the $FIREBIRD/plugins directory;

• code implementing a plug-in. That is slightly different from the library, since a single dynamic library may contain code for more than one plug-in;

• a plug-in's factory: an object created by that code (pure virtual C++ class), creating instances of the plug-in at Firebird's request;

• an instance of the plug-in, created by its factory.

**Plug-In Types**

Firebird's plug-in architecture makes it possible to create plug-ins of predefined types. Each version of Firebird will have a fixed set of supported plug-in types. To add a further type, the first requirement is to modify the Firebird code. Our plug-in architecture facilitates both adding new types of plug-ins and simplifying the coding of the plug-in along generic lines.

To be able to implement a plug-in, say, for encrypting a database on the disk, the Firebird code has to be prepared for it: it must have a point from which the plug-in is called.

The set of plug-in types implemented in Firebird 3 comprises:

*user authentication related:*
  - AuthServer (validates user's credentials on server when logins are used)
  - AuthClient (prepares credentials to be passed over the wire)
  - AuthUserManagement (maintains a list of users on a server in a format known to AuthServer)

*ExternalEngine*
  Controls the use of various engines, see External Engines.

*Trace*
  The Trace plug-in was introduced in Firebird 2.5, but the way it interacts with the engine was changed in Firebird 3 to accord with the new generic rules.

*Encryption*
  encrypting plug-ins are for
  - network (WireCrypt)
  - disk (DbCrypt)
  - a helper plug-in (KeyHolder), used to help maintain the secret key(s) for DbCrypt

*Provider*
  Firebird 3 supports providers as a plug-in type.

**Technical Details**

Plug-ins use a set of special Firebird interfaces. All plug-in-specific interfaces are reference counted, thus putting their lifetime under specific control. Interfaces are declared in the include file plug-in.h. A simple example for writing a plug-in module can be found in DbCrypt_example.
Features of a Plug-In

A short list of plug-in features:

- You can write a plug-in in any language that supports pure virtual interfaces. Interface declarations will need to be written for your language if they are missing.

- As with UDFs, you are free to add any reasonable code to your plug-in—with emphasis on reasonable. For example, prompting for user input at the server's console from a plug-in is hardly “reasonable”!

- Calling the Firebird API from your plug-in is OK, if needed. For example, the default authentication server and user manager use a Firebird database to store accounts.

- Firebird provides a set of interfaces to help with configuring your plug-ins. It is not obligatory to use them, since the plug-in code is generic and can employ any useful method for capturing configuration information. However, using the standard tools provides commonality with the established configuration style and should save the additional effort of rolling your own and documenting it separately.

Configuring Plug-ins

Configuration of plug-ins has two parts:

1. The engine has to be instructed what plug-ins it should load
2. The plug-ins themselves sometimes need some configuration.

The plug-ins to be loaded for each plug-in type are defined in the main configuration file, firebird.conf, usually with defaults. The ones defined in Firebird 3 are discussed in the chapter entitled “Configuration Additions and Changes”. In summary, the set that provides normal operation in the server, client and embedded cases consists of:

- AuthServer = Srp, Win_Sspi
- AuthClient = Srp, Win_Sspi, Legacy_Auth
- UserManager = Srp
- TracePlugin = fbtrace
- Providers = Remote,Engine12,Loopback
- WireCryptPlugin = Arc4

Note

If you want to add other plug-ins, they must be cited in firebird.conf. Apart from other considerations, this requirement acts as a security measure to avoid loading unknown code.
Taking the entry `TracePlugin = fbtrace` as an example, what does the value `fbtrace` signify? In a trivial case, it can indicate the name of a dynamic library but the precise answer is more complicated.

As mentioned earlier, a single plug-in module may implement more than one plug-in. In addition, a single plug-in may have more than one configuration at once, with a separate plug-in factory created for each configuration. Each of these three object contexts (module | implementation | factory) has its own name:

- The name of a module is the file name of a dynamic library
- The name of a plug-in implementation is the one given to it by the developer of the plug-in. It is hard-coded inside the module.
- The name of a factory is, by default, the same as the name of the plug-in implementation's name. It is the factory name which is actually used in `firebird.conf`.

In a typical trivial case, a module contains one plug-in that works with just one configuration and all three names are equal, and no more configuration is needed. An example would be `libEngine12.so | Engine12.dll | Engine12.dylib`, that contains the implementation of the embedded provider `Engine12`. Nothing other than the record `Providers = Engine12` is needed to load it.

For something more complex a file will help you to set up the plug-in factories precisely.

**plugins.conf**

The file `$(root)/plugins.conf` has two types of records: `config` and `plugin`.

the `plugin` record is a set of rules for loading and activating the plug-in. Its format is:

```
Plugin = PlugName ## this is the name to be referenced in firebird.conf
{
    Module = LibName ## name of dynamic library
    RegisterName = RegName ## name given to plug-in by its developer
    Config = ConfName ## name of config record to be used
    ConfigFile = ConfFile ## name of a file that contains plug-in's configuration
}
```

When plug-in `PlugName` is needed, Firebird loads the library `LibName` and locates the plug-in registered with the name `RegName`. The configuration values from the config record `ConfName` or the config file `ConfFile` are passed to the library.

**Note**

If both `ConfName` and `ConfFile` are given, then the config record will be used.

If both parameters are missing, the default `PlugName` is used; except that if the `ConfFile` is present and its name is the same as the module's dynamic library but with a `.conf` extension, it will be used.

The `ConfFile` is expected to use the format `Key=Value`, in line with other Firebird configuration files.

For the plug-in configuration record the same format is used:

```
Config = ConfName
{
    Key1 = Value1
```
A Sample Setup

Suppose you have a server for which some clients trust the wire encryption from one vendor and others prefer a
different one. They have different licences for the appropriate client components but both vendors use the name
“BestCrypt” for their products.

The situation would require renaming the libraries to, say, WC1 and WC2, since there cannot be two files in
the same directory with the same name. Now, the modules stop loading automatically because neither is called
“BestCrypt” any longer.

To fix the problem, plug-ins.conf should contain something like this:

```plaintext
Plugin = WC1
  {
    RegisterName = BestCrypt
  }
Plugin = WC2
  {
    RegisterName = BestCrypt
  }
```

The module names will be automatically set to WC1 and WC2 and found. You can add any configuration info
that the plug-ins need.

Remember to modify firebird.conf to enable both plug-ins for the WireCryptPlugin parameter:

```plaintext
WireCryptPlugin = WC1, WC2
```

The server will now select appropriate plug-in automatically to talk to the client.

Another sample is distributed with Firebird, in `$(root)/plugins.conf`, configuring one of the standard
plug-ins, UDR. Because it was written to a use non-default configuration, the module name and one configura-
tion parameter are supplied explicitly.

Plug-Ins Q & A

Q. There are plug-ins named Remote, Loopback, Arc4 in the default configuration, but no libraries with such
names. How do they work?

• A. They are “built-in” plug-ins, built into the fbclient library, and thus always present. Their existence is due
to the old ability to distribute the Firebird client for Windows as a single dll. The feature is retained for cases
where the standard set of plug-ins is used.

Q. What do the names of Srp and Arc4 plug-ins mean?

• A. Srp implements the Secure Remote Passwords protocol, the default way of authenticating users in Firebird
3. Its effective password length is 20 bytes, resistant to most attacks (including “man in the middle”) and
works without requiring any key exchange between client and server to work.
Arc4 means Alleged RC4 - an implementation of RC4 cypher. Its advantage is that it can generate a unique, cryptographically strong key on both client and server that is impossible to guess by capturing data transferred over the wire during password validation by SRP.

The key is used after the SRP handshake by Arc4, which makes wire encryption secure without need to exchange any keys between client and server explicitly.

Q. What do Win_Sspi and Legacy_Auth mean?

• A. Windows SSPI has been in use since Firebird 2.1 for Windows trusted user authentication. Legacy_Auth is a compatibility plug-in to enable connection by the Firebird 3 client to older servers. It is enabled by default in the client.

And Yes, it still transfers almost plain passwords over the wire, for compatibility.

On the server it works with security3.fdb just as with a security database from Firebird 2.5. It should be avoided except in situations where you understand well what you are sacrificing.

To use Legacy_Auth on the server you will need to avert network traffic encryption in firebird.conf by reducing the default Required setting for the WireCrypt parameter, either

WireCrypt = Enabled

or

WireCrypt = Disabled

Q. How can I find out what the standard Authentication and User Manager plug-ins are?

• They are listed in firebird.conf.

External Engines

Adriano dos Santos Fernandes

The UDR (User Defined Routines) engine adds a layer on top of the FirebirdExternal engine interface with the purpose of

• establishing a way to hook external modules into the server and make them available for use
• creating an API so that external modules can register their available routines
• making instances of routines “per attachment”, rather than dependent on the internal implementation details of the engine

External Names

An external name for the UDR engine is defined as

'&module name&!&routine name&!&misc info&'
Changes in the Firebird Engine

The `<module name>` is used to locate the library, `<routine name>` is used to locate the routine registered by the given module, and `<misc info>` is an optional user-defined string that can be passed to the routine to be read by the user.

**Module Availability**

Modules available to the UDR engine should be in a directory listed by way of the path attribute of the corresponding plugin_config tag. By default, a UDR module should be on `<fbroot>/plugins/udr`, in accordance with its path attribute in `<fbroot>/plugins/udr_engine.conf`.

The user library should include FirebirdUdr.h (or FirebirdUdrCpp.h) and link with the udr_engine library. Routines are easily defined and registered, using some macros, but nothing prevents you from doing things manually.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>A sample routine library is implemented in examples/udr, showing how to write functions, selectable procedures and triggers. It also shows how to interact with the current attachment through the legacy API.</td>
</tr>
</tbody>
</table>

**Scope**

The state of a UDR routine (i.e., its member variables) is shared among multiple invocations of the same routine until it is unloaded from the metadata cache. However, it should be noted that the instances are isolated “per session”.

**Character Set**

By default, UDR routines use the character set that was specified by the client.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>In future, routines will be able to modify the character set by overriding the <code>getCharSet</code> method. The chosen character set will be valid for communication with the old Firebird client library as well as the communications passed through the FirebirdExternal API.</td>
</tr>
</tbody>
</table>

**Enabling UDRs in the Database**

Enabling an external routine in the database involves a DDL command to “create” it. Of course, it was already created externally and (we hope) well tested.

**Syntax Pattern**

```sql
{ CREATE [ OR ALTER ] | RECREATE | ALTER } PROCEDURE <name>
[ ( <parameter list> ) ]
[ RETURNS ( <parameter list> ) ]
EXTERNAL NAME '<external name>' ENGINE <engine>

{ CREATE [ OR ALTER ] | RECREATE | ALTER } FUNCTION <name>
[ <parameter list> ]
RETURNS <data type>
EXTERNAL NAME '<external name>' ENGINE <engine>
```
Changes in the Firebird Engine

```sql
{ CREATE [ OR ALTER ] | RECREATE | ALTER } TRIGGER <name>

... EXTERNAL NAME '
<external name>' ENGINE <engine>

Examples

create procedure gen_rows (  
    start_n integer not null,
    end_n integer not null )  
returns (  
    n integer not null )  
) external name 'udrcpp_example!gen_rows'
    engine udr;

create function wait_event (  
    event_name varchar(31) character set ascii )  
returns integer  
) external name 'udrcpp_example!wait_event'
    engine udr;

create trigger persons_replicate
    after insert on persons
    external name 'udrcpp_example!replicate!ds1'
    engine udr;

How it Works

The external names are opaque strings to Firebird. They are recognized by specific external engines. External engines are declared in configuration files, possibly in the same file as a plug-in, as in the sample UDR library that is implemented in $(root)/plugins.

```text
external_engine = UDR {
    plugin_module = UDR_engine
}

plugin_module = UDR_engine {
    filename = $(this)/udr_engine
    plugin_config = UDR_config
}

plugin_config = UDR_config {
    path = $(this)/udr
}
```text

When Firebird wants to load an external routine (function, procedure or trigger) into its metadata cache, it gets the external engine through the plug-in external engine factory and asks it for the routine. The plug-in used is the one referenced by the attribute `plugin_module` of the external engine.

Note

Depending on the server architecture (Superserver, Classic, etc) and implementation details, Firebird may get external engine instances “per database” or “per connection”. Currently, it always gets instances “per database”.

Note
Changes in the Firebird Engine

Optimizer Improvements

Dmitry Yemanov

• See Tracker item CORE-4528.

Hash/merge joins for non-field (DBKEY or derived expression) equalities are now allowed.

• See Tracker item CORE-1482.

The optimizer now considers the ORDER BY optimization when making its decision about join order.

Other Optimizations

Vlad Khorsun

• See Tracker item CORE-4556.

Data pages are now allocated as a group of sequential ordered pages (extents).

• See Tracker item CORE-4445.

The main database file extends faster when physical backup state changes from stalled to merge.

• See Tracker item CORE-4443.

Linux systems that support “fast file growth” can now use it.

• See Tracker item CORE-4432.

Attachments no longer block others when the allocation table is read for the first time.

• See Tracker item CORE-4431.

Contention has been reduced for the allocation table lock while database is in stalled physical backup state.

Remote Interface/Network Protocol

Dmitry Yemanov

Tracker item CORE-2530.

Further improvements were made to Firebird's network protocol, providing a denser data stream and better prefetch logic. The following improvements were implemented:

1. The full length of a field whose value is NULL is no longer sent over the wire. (Tracker item CORE-2897). NULL flags (4 bytes per field) are replaced with a bitmap and only these flags are transmitted, in the bitmap. This improvement is available for the DSQL API only, so gbak does not benefit from this improvement, as it uses a lower level BLR API.
2. The prefetch (batch receive) algorithm is now aware of variable-length messages, so that VARCHARs and NULLs may reduce the transmitted message size, allowing more rows to be transmitted in each batch.

Acknowledgement

This work was sponsored by donations collected at the 9th Firebird Developers' Day conference in Brazil.

Miscellaneous Improvements

Miscellaneous engine improvements include.

**Connections Limit Raised**

Paul Beach

([CORE-4439]) :: Maximum connections (FD_SETSIZE) on Windows Superserver and Superclassic was raised from 1024 to 2048.

**Better Error Diagnosis**

Dmitry Yemanov

([CORE-3881]) :: The error reported for index/constraint violations has been extended to include the problematic key value.

**ICU Version Upgraded**

Adriano dos Santos Fernandes

([CORE-2224]) :: The ICU version was upgraded to v.52.1.

**Internal Debug Info Made Human-readable**

Vlad Khorsun

A new BLOB filter translates internal debug information into text.

**A Silly Message is Replaced**

Claudio Valderrama C.

A silly message sent by the parser when a reference to an undefined object was encountered was replaced with one that tells it like it really is.

**New Pseudocolumn RDB$RECORD_VERSION**

Adriano dos Santos Fernandes
A pseudocolumn named RDB$RECORD_VERSION returns the number of the transaction that created the current record version.

It is retrieved the same way as RDB$DB_KEY, i.e., select RDB$RECORD_VERSION from aTable where...

**systemd init Scripts**
Alex Peshkov

systemd init scripts are available in Firebird 3 POSIX installers. See Tracker ticket [CORE-4085](#).
Chapter 4

Changes to the Firebird API and ODS

ODS (On-Disk Structure) Changes

**New ODS Number**

Firebird 3.0 creates databases with an ODS (On-Disk Structure) version of 12. In the initial release, a database with an older ODS cannot be opened by Firebird 3.0. In order to work with a database with an older ODS it will be necessary to make a backup using `gbak` under the older server and restore it with `gbak` on Firebird 3.

**Note**

A legacy provider for databases with ODS 8 to 11.2 is planned for a future sub-release.

**Implementation ID is Deprecated**

Alex Peshkov

The Implementation ID in the ODS of a database is deprecated in favour of a new field in database headers describing hardware details that need to match in order for the database to be assumed to have been created by a compatible implementation.

The old Implementation ID is replaced with a 4-byte structure consisting of hardware ID, operating system ID, compiler ID and compatibility flags. The three ID fields are just for information: the ODS does not depend upon them directly and they are not checked when opening the database.

The compatibility flags are checked for a match between the database and the engine opening it. Currently we have only one flag, for endianness. As previously, Firebird will not open a database on little-endian that was created on big-endian, nor vice versa.

**Sample gstat Output**

```
# ./gstat -h employee
Database "/usr/home/firebird/trunk/gen/Debug/firebird/examples/empbuild/employee.fdb"
Database header page information:
..................
Implementation       HW=AMD/Intel/x64 little-endian OS=Linux CC=gcc
..................
```
The purpose is to make it easier to do ports of Firebird for new platforms.

**Maximum Database Size**

Maximum database size is increased to $2^{32}$ pages (previously $2^{31}$ pages). The new limit is 16TB|32TB|64TB, depending on the page size.

**Maximum Page Size**

The maximum page size remains 16 KB (16384 bytes).

**Maximum Number of Page Buffers in Cache**

The maximum number of pages that can be configured for the database cache depends on whether the database is running under 64-bit or 32-bit Firebird:

- 64-bit :: $2^{31} - 1$ (2,147,483,647) pages
- 32-bit :: 128,000 pages, i.e., unchanged from V.2.5

**Extension of Transaction ID Space Limit**

Dmitry Yemanov

Historically, transaction ID space was limited to $2^{31}$ transactions, counted from the time the database was created. After that point, the database becomes unavailable until backup and restore is performed to reset the transaction ID counter back to zero. Initially in Firebird 3.0, the transaction ID space was raised to $2^{32}$ transactions, doubling the database up-time without backup and restore.

This improvement request is about shifting that limit even further, with the introduction of 48-bit internal transaction IDs that are publicly (via the API and the MON$ tables) represented as 64-bit numbers. This makes the new limit roughly equal to $2.8 \times 10^{14}$ transactions. Later, it could be extended up to the $2^{63}$ limit.

The implemented solution has no additional storage overhead until the transaction counters grow beyond the $2^{32}$ boundary.

**Limits Raised for Attachment and Statement IDs**

Attachment IDs and statement IDs were changed to 64-bit numbers, both internally and externally via the API and the MON$ tables.

**System Tables**

**New System Tables**

RDB$AUTH_MAPPING Stores authentication and other security mappings
Changes to System Tables

**RDB$SYSTEM_FLAG**
Claudio Valderrama C.

RDB$SYSTEM_FLAG has been made NOT NULL in all tables.

**CORE-2787.**

**RDB$TYPES**
Dmitry Yemanov

Missing entries were added to RDB$TYPES. They describe the numeric values for these columns:

- RDB$PARAMETER_TYPE (table RDB$PROCEDURE_PARAMETERS)
- RDB$INDEX_INACTIVE (table RDB$INDICES)
- RDB$UNIQUE_FLAG (table RDB$INDICES)
- RDB$TRIGGER_INACTIVE (table RDB$TRIGGERS)
- RDB$GRANT_OPTION (table RDB$USER_PRIVILEGES)
- RDB$PAGE_TYPE (table RDB$PAGES)
- RDB$PRIVATE_FLAG (tables RDB$PROCEDURES and RDB$FUNCTIONS)
- RDB$LEGACY_FLAG (table RDB$FUNCTIONS)
- RDB$DETERMINISTIC_FLAG (table RDB$FUNCTIONS)

Monitoring Tables
Dmitry Yemanov

Changes to Client Address Reporting

Prior to Firebird 3.0, the network address of remote clients were reported in MON$ATTACHMENTS.MON$REMOTE_ADDRESS and RDB$GET_CONTEXT('SYSTEM', 'CLIENT_ADDRESS'). For TCP/IP protocol (a.k.a. INET), it contained a TCPv4 dot-separated address. For Named Pipes (a.k.a. WNET, NetBeui) protocol, it was always NULL. For shared memory (aka XNET) protocol, it contained the local host name.

Starting with Firebird 3.0, the network address of a remote client contains the TCP/IP port number of the remote client, separated with a slash:
Changes to the Firebird API and ODS

<IP address>/<port>

The port number is also retrieved via the new built-in context variable RDB$GET_CONTEXT('SYSTEM', 'CLIENT_PORT').

The host name is also reported now, in the new column MON$REMOTE_HOST.

Alert
The WNET (Named Pipes/Netbeui) protocol should be considered as deprecated. It is likely to be abandoned in a future version.

Per-table performance counters have been added to all of the monitoring tables. See Tracker CORE-4564.

MON$ATTACHMENTS

New information is now available:

- Operating system user name. See Tracker CORE-3779.
- Client host name. See Tracker CORE-2187.
- authentication method used for connection (MON$AUTH_METHOD). See Tracker CORE-4222.
- MON$REMOTE_ADDRESS now contains the <IP>/<port> string. See Tracker CORE-5028.

MON$DATABASE

- Database owner (MON$OWNER) added. See Tracker CORE-4218.
- Security database type (MON$SEC_DATABASE) flag added. Value will be one of Default/Self/Other. See Tracker CORE-4729.

MON$STATEMENTS

The PLAN is now included. See Tracker CORE-2303.

Application Programming Interfaces

A new public API replaces the legacy one in new applications, especially object-oriented ones. The interface part can be found in the header file Interfaces.h in the directory /include/firebird beneath the installation root directory.

Note
POSIX installations have a symlink pointing to /usr/include/firebird/Interfaces.h

The new public API can be also used inside user-defined routines (UDR, q.v.) for callbacks inside the engine, allowing a UDR to select or modify something in the database, for example.
The main difference between the new API and the legacy one is that UDRs can query and modify data in the same connection or transaction context as the user query that called that UDR. It is now possible to write external triggers and procedures, not just external functions (UDFs).

**Interfaces and the New Object-oriented API**

Alex Peshkov

Firebird needed a modernised API for a number of compelling reasons.

- High on the list was the limitation of the 16-bit integer pervading the legacy API, encompassing message size, SQL operator length, BLOB data portions, to name a few examples. While 16-bit was probably adequate when that old API came to life, in today's environments it is costly to work around.

  A trivial solution might be to add new functions that support 32-bit variables. The big downside is the obvious need to retain support for the old API by having pairs of functions with the same functionality but differing integer sizes. In fact, we did something like this to support 64-bit performance counters, for no better reason than being pressed to provide for it without having a more elegant way to implement it.

- Another important reason, less obvious, derives from the era when Firebird's predecessor, InterBase, did not support SQL. It used a non-standard query language, GDML, to manage databases. Data requests were transported between client and server using messages whose formats were defined at request compilation time in BLR (binary language representation). In SQL, the operator does not contain the description of the message format so the decision was taken to surround each message with a short BLR sequence describing its format.

  The ISC API also has the XSQLDA layer over BLR. The trap with the XSQLDA solution is that it encapsulates both the location of the data and their format, making it possible to change location or format (or both) between fetch calls. Hence, the need for the BLR wrapping in *every* fetch call—notwithstanding, this potential capability to change the data format between fetches was broken in the network layer before Firebird existed.

  But to support the XSQLDA layer that rides on top of the message-based API that lower level API also has support sending format BLR at every turn.

  This system involving calls processing data through multiple layers is hard to extend and wastes performance; the SQLDA is not simple to use; the desire to fix it was strong.

- Other reasons—numerous but perhaps less demanding—for changing the API included enhancing the status vector and optimizing dynamic library loading. Interfaces also make it so much easier and more comfortable to use the messages API.

**The Non-COM Choice**

The new interfaces are not compatible with COM, deliberately, and the reasons have to do with future performance enhancement.

At the centre of the Providers architecture in Firebird 3.0 is the *y-valve*, which is directed at dispatching API calls to the correct provider. Amongst the potential providers are older ones with potentially older interfaces. If we used COM, we would have to call the method `IUnknown` for each call (including record fetch), just to ensure that the provider really had some newer API method. Along with that comes the likelihood of future additions to the catalogue of API calls to optimize performance. A COM-based solution does not play well with that.

Firebird interfaces, unlike COM, support multiple versions. The interface version is determined by the total number of virtual functions it encompasses and is stored as a pointer-size integer at the beginning of the virtual
functions table. This makes it possible for very fast checking of the interface version, since it requires no virtual call. That is to say, the pointer check has no overhead, unlike COM.

The Hierarchy of Interfaces

A detailed discussion of all the functions presented by all the interfaces is outside the scope of this overview. The general schematic looks like this:

The base of the structure is IVersioned. It is the interface that enables a version upgrade. A lot of interfaces not requiring additional lifetime control are based directly on IVersioned. IMaster is one example already mentioned. Others include a number of callback interfaces whose lifetimes must match the lifetimes of the objects from which they were to be used for callback.

Two interfaces deal with lifetime control: IDisposable and IRefCounted. The latter is especially active in the creation of other interfaces: IPlugin is reference counted, as are many other interfaces that are used by plug-ins. These include the interfaces that describe database attachment, transaction management and SQL statements.

Not everything needs the extra overhead of a reference-counted interface. For example, IMaster, the main interface that calls functions available to the rest of the API, has unlimited lifetime by definition. For others, the API is defined strictly by the lifetime of a parent interface; the IStatus interface is non-threaded. For interfaces with limited lifetimes it is of benefit to have a simple way to destroy them, that is, a dispose() function.

Each plug-in has one and only one main interface—IPlugin—which is responsible for basic plug-in functionality. In fact, a lot of plugins have only that interface, although that is not a requirement.

Finally, there is IProvider, a kind of “main” plug-in in the Firebird API. IProvider is derived from IPlugin and must be implemented by every provider. If you want to write your own provider you must implement IProvider. It is implemented also by the y-valve: it is the y-valve implementation that is returned to the user when the getDispatcher() function from the master interface is called.

IProvider contains functions enabling creation of an attachment to a database (attach and create) or to the Services Manager.

Interfaces Q & A

Q. We access new API using IMaster. But how to get access to IMaster itself?

A. This is done using just the one new API function fb_get_master_interface(). It is exported by the fb-client library. Also IMaster is passed as a parameter to each plug-in during its registration in the system.

Q. The non-use of COM-based interfaces was said to be to avoid working with IUnknown methods and that this is done due to performance issues. Instead you have to check the interface version. Why is that faster than using IUnknown?

A. As was already mentioned we do not need to execute virtual calls when checking the interface version. Taking into account that each virtual call means a reset of the CPU cache, it is an important difference, especially for the very small calls like getting specific metadata properties from IMetadata.

Other New APIs

Other new APIs support various plug-ins by declaring the interfaces between the engine and the plug-in. Besides pluggable authentication and pluggable encryption, Firebird 3 supports “external engines”, bridges between the
engine and the execution environments that can run UDRs: native code, Java and others. By and large they are intended for use by third-party solution providers, rather than for client application development.

For creating custom plug-ins and bridges, the relevant interface (API) needs to be implemented in the plug-in code.

**API Improvements**

The following improvements to the API should be noted.

**Some SQL Size Limits Removed Using New API**

Dmitry Yemanov

If and only if the new API is being used.

- The size of the body of a stored procedure or a trigger can exceed the traditional limit of 32 KB. The theoretical limit provided by the new API is 4GB. At the moment, as a security measure, a hard-coded limit of 10MB is imposed. The same limit of 10MB also applies to any user-defined DSQL query.

- The total size of all input or output parameters for a stored procedure or a user-defined DSQL query is no longer limited to the traditional size of (64KB minus overhead). The theoretical limit provided by the new API is 4GB.

**Legacy API**

Improvements to the legacy API include.

**Scrollable Cursor Support**

Dmitry Yemanov

In PSQL, a scrollable cursor can be operated on directly to navigate flexibly from the current row to any another row either forwards or backwards. API support is available to make scrollable cursors available to DSQL applications.

**Scrollable Cursor Usage**

The result set must be opened with the flag `IStatement::CURSOR_TYPE_SCROLLABLE` explicitly specified.

**Fetch Methods**

The following fetch methods of the IResultSet interface are available:

```c
int fetchNext(IStatus* status, void* message);
// equivalent to FETCH NEXT FROM <cursor name>
```

Moves the cursor's current position to the next row and returns it. If the cursor is empty or already positioned at the last row, the condition `NO_DATA` is returned.
Changes to the Firebird API and ODS

int fetchPrior(IStatus* status, void* message);
// equivalent to FETCH PRIOR FROM <cursor name>

Moves the cursor's current position to the prior row and returns it. If the cursor is empty or already positioned
at the first row, the condition **NO_DATA** is returned.

int fetchFirst(IStatus* status, void* message);
// equivalent to FETCH FIRST FROM <cursor name>

Moves the cursor's current position to the first row and returns it. If the cursor is empty, the condition **NO_DATA**
is returned.

int fetchLast(IStatus* status, void* message);
// equivalent to FETCH LAST FROM <cursor name>

Moves the cursor's current position to the last row and returns it. If the cursor is empty, the condition **NO_DATA**
is returned.

int fetchAbsolute(IStatus* status, int position, void* message);
// equivalent to FETCH ABSOLUTE <position> FROM <cursor name>

Moves the cursor's current position to the specified <position> and returns the located row. If <position> is
beyond the cursor's boundaries, the condition **NO_DATA** is returned.

int fetchRelative(IStatus* status, int offset, void* message);
// equivalent to FETCH RELATIVE <offset> FROM <cursor name>

Moves the cursor's current position backward or forward by the specified <offset> and returns the located row.
If the calculated position is beyond the cursor's boundaries, the condition **NO_DATA** is returned.

**Notes**

1. When a scrolling option is omitted, NO SCROLL is implied (i.e., the cursor is opened as forward-only).
   This means that only the **fetchNext()** API call can be used. Other fetch methods will return an error.

2. Scrollable cursors are internally materialized as a temporary record set, thus consuming memory/disk
   resources, so this feature should be used only when really necessary.

**SPB Support for New Statistics Feature in gbak Output**

Vlad Khorsun

A new, much requested feature was added to **gbak** verbose output: optional run-time statistics. Read about it
here. The feature is fully supported in the Services API with a new item in the SPB (Services Parameter Block),

```
#define isc_spb_bkp_stat 15
```

along with its synonym
#define isc_spb_res_stat isc_spb_bkp_stat

**Usage**

isc_spb_bkp_stat, <len>, <string>
isc_spb_res_stat, <len>, <string>

where <len> (2 bytes) indicates the length of the following string parameter, and <string> (1-4 bytes) is a string consisting of one character per statistics item.

The `fbsvcmgr` utility also supports the new SPB tags.

**Better Error Reports for String Overflows**
Alex Peshkov

Include expected and actual string length in the error message for string overflows (SQLCODE -802).

**More Detail in “Wrong Page Type” Error Reports**
Alex Peshkov

More details in the error message "wrong page type", i.e., identifying expected and encountered page types by name instead of numerical type.

**New Item for isc_database_info() Call**
Vlad Khorsun

An option was added to the API function `isc_database_info()` to return the number of free pages in a database. See [CORE-1538](https://example.com).

**New Services Tag for Overriding LINGER**
Alex Peshkov

The Services API now includes the tag `isc_spb_prp_nolinger`, for example (in one line):

```
fbsvcmgr host:service_mgr user sysdba password xxx
   action_properties dbname employee prp_nolinger
```

For information regarding LINGER, see the write-up in the DDL chapter.

**New Connection Formats for Local Superserver Clients on Windows**

In previous Firebird versions, a serverless protocol known as “Windows Local” was available to local clients connecting to Superserver on a Windows platform, using the XNET subsystem. A typical connection string looked like this:

```
c:\Program Files\Firebird_2_5\examples\empbuild\employee.fdb
```
Changes to the Firebird API and ODS

Under the new unified server, that form of connection attempts to load an embedded server. It is no longer valid for a serverless client connection to Superserver. If you try, you will get a refusal message to the effect “File is in use by another process”. This is not a bug. Since Superserver clients share resources, another server (in this case, an embedded server) cannot attach a client to the same database that Superserver has any clients attached to.

However, all is not lost. The XNET subsystem can still do local client sessions for Superserver. You just need a more elaborate connection string now. You have a few choices:

- this one is the former “Windows local”, using the XNET subsystem and shared memory for a (nominally) serverless connection:

  \texttt{xnet://alias-or-path-to-database}

  So, for our connection to the employee database:

  \texttt{xnet://c:\Program Files\Firebird\examples\empbuild\employee.fdb}

  or using an alias:

  \texttt{xnet://employee}

- Connection to host/port via TCP:

  \texttt{inet://host:port/alias-or-path-to-database}

- Connection to host/port via Named Pipes (aka NetBEUI):

  \texttt{wnet://host:port/alias-or-path-to-database}

- Connection to localhost via TCP:

  \texttt{inet://alias-or-path-to-database}

- Connection to localhost via named pipes (aka NetBEUI):

  \texttt{wnet://alias-or-path-to-database}

**Perform Some Validation Services On-line**

Vlad Khorsun

This feature was ported forward from Firebird 2.5.4.

Database validation enables low-level checks of the consistency of on-disk structures and even to fix some minor corruptions. The recommended procedure for any valuable database is for the DBA to validate a database periodically to ensure it is healthy.
Exclusive access to the database is required: any kind of concurrent access is forbidden during validation. Sometimes, blocking user access could be a major hold-up, especially if the database is large and complex.

Online validation is a new feature that allows some consistency checks to be performed without exclusive access.

**What Online Validation Can Do**

- validate some (or all) user tables in a database.  
  *System tables are not validated.*
- validate some (or all) indices

Other ODS checks, such as Header\PIP\TIP\Generators pages, are not performed.

**Protection During Online Validation**

While a table (and/or its index) is undergoing validation, user attachments are allowed to read this table. Any attempt to change data (INSERT\UPDATE\DELETE) will wait until validation finishes or, depending on the lock timeout of the user transaction, will return a lock timeout error.

Any kind of garbage collection on the table or its indexes is disabled whilst it is undergoing validation:

- background and cooperative garbage collection will just skip this table
- sweep will be terminated with an error

When online validation starts to check a table, it acquires a couple of locks to prevent concurrent modifications of its data:

- a relation lock in PR (protected read) mode
- (NEW) a garbage collection lock in PW (protected write) mode

Both locks are acquired using a user-specified lock timeout. An error is reported for any lock request that fails and that table is skipped.

Once the locks are acquired, the table and its indexes are validated in the same way as a full validation does it. The locks are released when it completes and the whole procedure is repeated for the next table.

**The New Services API action: isc_action_svc_validate**

Online validation is implemented as a Firebird service and is accessed through the Services API. Thus, it cannot be run from the gfix utility.

The call involves the following elements:

**Action:**
- isc_action_svc_validate

**Parameters:**
- isc_spb_dbname :  
  database file name, string, mandatory
isc_spb_val_tab_incl, isc_spb_val_tab_excl,  
isc_spb_val_idx_incl, isc_spb_val_idx_excl :  
patterns for tables\indices names, string, optional

isc_spb_val_lock_timeout :  
lock timeout, integer, optional

Output:  
text messages with progress of online validation process

Using isc_action_svc_validate Interactively

The fbsvcmgr utility has full support for the new service. The syntax is:

fbsvcmgr [host:]service_mgr [user <...>] [password <...>]  
action_validate dbname <filename>  
[val_tab_incl <pattern>]  
[val_tab_excl <pattern>]  
[val_idx_incl <pattern>]  
[val_idx_excl <pattern>]  
[val_lock_timeout <number>]

where

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>val_tab_incl</td>
<td>pattern for table names to include in validation run</td>
</tr>
<tr>
<td>val_tab_excl</td>
<td>pattern for table names to exclude from validation run</td>
</tr>
<tr>
<td>val_idx_incl</td>
<td>pattern for index names to include in validation run, by</td>
</tr>
<tr>
<td></td>
<td>default %, i.e. all indexes</td>
</tr>
<tr>
<td>val_idx_excl</td>
<td>pattern for index names to exclude from validation run</td>
</tr>
<tr>
<td>val_lock_timeout</td>
<td>lock timeout, used to acquire locks for table to validate, in</td>
</tr>
<tr>
<td></td>
<td>seconds, default is 10 secs. 0 is no-wait, -1 is infinite wait</td>
</tr>
</tbody>
</table>

Usage Notes

- Patterns are regular expressions, processed by the same rules as SIMILAR TO expressions.
- All patterns are case-sensitive, regardless of database dialect.
- If the pattern for tables is omitted then all user tables will be validated.
- If the pattern for indexes is omitted then all indexes of the appointed tables will be validated.
- System tables are not validated.
- To specify a list of tables or indexes:
  1. Separate names with the pipe character ‘\|’
  2. Do not add spaces: TAB1 \| TAB2 is wrong
  3. Enclose the whole list in double quotes to avoid confusing the command interpreter

Examples
1. Validate all tables in database 'c:\db.fdb' with names starting with 'A'. Indexes are not validated. Lock wait is not performed.

```
fbsvcmgr.exe service_mgr user SYSDBA password masterkey
    action_validate dbname c:\db.fdb
    val_tab_incl A%
    val_idx_excl %
    val_lock_timeout 0
```

2. Validate tables TAB1 and TAB2 and all their indexes. Lock wait timeout is 10 seconds (the default):

```
fbsvcmgr.exe service_mgr user SYSDBA password masterkey
    action_validate dbname c:\db.fdb
    val_tab_incl "TAB1|TAB2"
```

3. Default behavior of val_XXX options: validate all user tables and their indexes in database 'c:\db.fdb', lock wait is the default 10 seconds:

```
fbsvcmgr.exe service_mgr user SYSDBA password masterkey
    action_validate dbname c:\db.fdb
```

**Code Improvement**
Alex Peshkov

*(CORE-4387)* The functions `IStatement::execute()` and `IAttachment::execute()` now return an error pointer to the old transaction interface.
Chapter 5

Reserved Words and Changes

New Keywords in Firebird 3.0

Reserved

Items marked with asterisks (*) were previously non-reserved.

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Non-reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>REGR_AVGX</td>
</tr>
<tr>
<td>CORR</td>
<td>REGR_AVGY</td>
</tr>
<tr>
<td>COVAR_POP</td>
<td>REGR_COUNT</td>
</tr>
<tr>
<td>COVAR_SAMP</td>
<td>REGR_INTERCEPT</td>
</tr>
<tr>
<td>DELETING *</td>
<td>REGR_R2</td>
</tr>
<tr>
<td>DETERMINISTIC</td>
<td>REGR_SLOPE</td>
</tr>
<tr>
<td>FALSE</td>
<td>REGR_SXX</td>
</tr>
<tr>
<td>INSERTING *</td>
<td>REGR_SXY</td>
</tr>
<tr>
<td>OFFSET</td>
<td>REGR_SYY</td>
</tr>
<tr>
<td>OVER</td>
<td>RETURN</td>
</tr>
<tr>
<td>RDB$RECORD_VERSION</td>
<td>ROW</td>
</tr>
<tr>
<td></td>
<td>FIRST_VALUE</td>
</tr>
<tr>
<td>ABSOLUTE</td>
<td>IDENTITY</td>
</tr>
<tr>
<td>ACOSH</td>
<td>INCREMENT</td>
</tr>
<tr>
<td>ASINH</td>
<td>LAST_VALUE</td>
</tr>
<tr>
<td>ATANH</td>
<td>LAG</td>
</tr>
<tr>
<td>BODY</td>
<td>LEAD</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>LINGER</td>
</tr>
<tr>
<td>DDL</td>
<td>NAME</td>
</tr>
<tr>
<td>DECRYPT</td>
<td>NTH_VALUE</td>
</tr>
<tr>
<td>DENSE_RANK</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>ENCRYPT</td>
<td>PARTITION</td>
</tr>
<tr>
<td>ENGINE</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 6

Configuration
Additions and Changes

The file aliases.conf is renamed to databases.conf. An old aliases.conf from a previous version can simply be renamed and the new engine will just continue to use it as before. However, databases.conf can now include some configuration information for individual databases.

Scope of Parameters

Some parameters are marked as configurable per-database or per-connection.

• Per-database configuration is done in databases.conf.

• Per-connection configuration is primarily for client tool use and is done using the DPB parameter isc_dpb_config or, for Services, the SPB parameter isc_spb_config.

• In the case of Embedded, the DPB can be used to tune per-database entries on first attaching to a database.

Macro Substitution

A number of predefined macros (syntax $(name)) is available for use in the configuration files to substitute for a directory name:

$(root)
Root directory of Firebird instance

$(install)
Directory where Firebird is installed. $(root) and $(install) are initially the same. $(root) can be overridden by setting or altering the environment variable FIREBIRD, in which case it becomes different from $(install).

$(this)
Directory where current configuration file is located

$(dir_conf)
Directory where firebird.conf and databases.conf are located

$(dir_secdb)
Directory where the default security database is located

$(dir_plugins)
Directory where plugins are located
$(dir_udf)
  Directory where UDFs are located by default

$(dir_sample)
  Directory where samples are located

$(dir_sampledb)
  Directory where sample DB (employee.fdb) is located

$(dir_intl)
  Directory where international modules are located

$(dir_msg)
  Directory where the messages file (firebird.msg) is located. $(dir_msg) usually should be the same as $(root) but can be overridden by the environment variable FIREBIRD_MSG.

Tip
You can observe the usage of some of the macros in databases.conf.

Notes
In our pre-built binaries, $(dir_conf) and $(dir_secdb) would normally be the same as $(root) and $(install).

$(dir_plugins), $(dir_udf), $(dir_sample), $(dir_sampledb) and $(dir_intl) are predefined sub-directories inside $(root).

The build conventions are not “rules” that could be expected to apply in every distribution of Firebird. Distro-specific Linux packages, for example, each prefer to fit the Firebird components into standard layouts that comply with their own conventions. As an illustration, user binaries, such as isql might be located in /usr/bin, server binaries in /usr/sbin, configuration files in /etc/firebird.d and so on. Obviously, $(root) would then make no sense, even if the $(dir_something) macros still pointed to actual directories.

Includes

One configuration file can be included in another by using an “include” directive, e.g.,

include some_file.conf

A relative path is treated as relative to the enclosing configuration file. So, if our example above is inside /opt/config/master.conf then our include refers to the file /opt/config/some_file.conf.

Wildcards

The standard wildcards * and ? may be used in an include directive, to include all matching files in undefined order. For example,

include $(dir_plugins)/config/*.conf
Expression of Parameter Values

Previously, byte values were specified by default as integer, representing the number of bytes. However, now you can optionally specify them in Kilobytes, Megabytes or Gigabytes, as appropriate, by adding K, M or G (case-insensitive). For example, 24M is read as 25165824 (24 * 1024 * 1024).

Boolean values are expressed as non-zero (true)/zero (false) by default, but you may now use the quoted strings 'y', 'yes' or 'true' instead of a non-zero digit.

“Per-database” Configuration

Custom configuration at database level is achieved with formal entries in databases.conf.

Format of Configuration Entries

In aliases.conf the format for specifying a database alias was

```
aliasname = /absolute/path/to/database_file
```

If you are not adding any database-specific configuration directives for an alias, the format is just as it was before, e.g.,

```
emp = c:\Program Files\examples\empbuild\employee.fdb
** or **
emp = /opt/firebird/examples/empbuild/employee.fdb
** or **
emp = $(dir_sampleDb)/employee.fdb
```

A slightly more complex format is used for cases where certain non-global parameters are to be targeted at an individual databases. The entry for the database is defined by the alias declaration, as previously. The database-specific directives are listed below it, within curly brackets.

```
# Directives for MYBIGDB
MYBIGDB = opt/databases/mybigdb.fdb
{
    LockMemSize = 32M # We know that MYBIGDB needs a lot of locks
    LockHashSlots = 19927 # and a hash table large enough for them
}
```

Parameters Available

The following parameters can be copy/pasted to databases.conf and used as overrides for specific databases.
Table 6.1. Parameters available in databases.conf

<table>
<thead>
<tr>
<th>Engine-related</th>
<th>DatabaseGrowthIncrement</th>
<th>DeadlockTimeout</th>
<th>DefaultDbCachePages</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventMemSize</td>
<td>FileSystemCacheThreshold</td>
<td>ExternalFileAccess</td>
<td></td>
</tr>
<tr>
<td>GCPolicy</td>
<td>LockAcquireSpins</td>
<td>LockHashSlots</td>
<td></td>
</tr>
<tr>
<td>LockMemSize</td>
<td>MaxUnflushedWrites</td>
<td>MaxUnflushedWriteTime</td>
<td></td>
</tr>
<tr>
<td>SecurityDatabase</td>
<td>UserManager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WireCompression</td>
<td>WireCrypt</td>
<td>WireCryptPlugin</td>
<td></td>
</tr>
</tbody>
</table>

**Client-related**

Some parameters can be configured at the client connection via the DPB/SPB, as an alternative to configuring them in databases.conf. Please refer back to Scope of Parameters at the beginning of this chapter to understand these differences.

<table>
<thead>
<tr>
<th>AuthClient</th>
<th>Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionTimeout</td>
<td>DummyPacketInterval</td>
</tr>
<tr>
<td>RemoteAuxPort</td>
<td>RemotePipeName</td>
</tr>
<tr>
<td>RemoteServicePort</td>
<td>TCPNoNagle</td>
</tr>
</tbody>
</table>

**New Parameters**

New parameters added to firebird.conf are:

**SecurityDatabase**

Defines the name and location of the security database that stores login user names and passwords used by the server to validate remote connections. By default, in firebird.conf, it is $(root)/security3.fdb. It can be overridden for a specific database by a configuration in databases.conf.

**AuthServer and AuthClient**

Two parameters that determine what authentication methods can be used by the network server and the client redirector. The enabled methods are listed as string symbols separated by commas, semicolons or spaces.

- Secure remote passwords (Srp), using the plug-in is the default, using the OS-appropriate plug-in (libSrp.s0 | Srp.dll | Srp.dylib)
• On Windows, the Security Support Provider Interface (Sspi) is used when no login credentials are supplied.

• Client applications can use legacy authentication (Legacy_Auth) to talk to old servers.

For AuthServer, Srp and Win_Sspi are listed; for AuthClient, Srp, Win_Sspi and Legacy_Auth.

To disable a method, erase the comment marker (#) and remove the unwanted method from the list.

Both parameters can be used in databases.conf. They can both be used in the DPB or the SPB for a connection-specific configuration.

**WireCrypt**

Sets whether the network connection should be encrypted. It has three possible values: Required | Enabled | Disabled. The default is set such that encryption is Required for connections coming in to the server and Enabled for connections outgoing to a server.

To access a server using an older client library and, thus, no encryption, WireCrypt in the server configuration file should be set to Enabled or Disabled to avert the default Required.

The rules are simple: if one side has WireCrypt = Required and the other sets the parameter to Disabled, side with WireCrypt=Required rejects the connection and it is not established.

A missing WireCrypt plug-in or encryption key in cases where the channel must be encrypted also thwarts a connection.

In all other cases, connection is established without encryption if at least one side has WireCrypt = Disabled. In other cases, the encrypted connection is established.

**UserManager**

Sets the plug-in that will operate on the security database. It can be a list with blanks, commas or semicolons as separators: the first plug-in from the list is used.

The default plug-in is Srp (libSrp.s0 | Srp.dll | Srp.dylib).

The UserManager parameter can be used in databases.conf for a database-specific override.

**TracePlugin**

Specifies the plug-in used by Firebird's Trace facility to send trace data to the client app or audit data to the log file.

The default plug-in is fbtrace (libfbtrace.s0 | fbtrace.dll | fbtrace.dylib).

**WireCryptPlugin**

A wire-crypt plug-in is used to encrypt and decrypt data transferred over the network.
The installation default **Arc4** implies use of an **Alleged RC4** plug-in. The configured plug-in, which requires a key generated by the configured authentication plug-in, can be overridden in the API for a specific connection via the DPB or the SPB.

**Tip**
For information about configuring plug-ins, see Configuring Plug-ins in the Engine chapter.

**KeyHolderPlugin**
This parameter would represent some form of temporary storage for database encryption keys. Nothing is implemented as a default plug-in but a sample Linux plug-in named libCryptKeyHolder_example.so can be found in /plugins/.

**Providers**
List of allowed transports for accessing databases, discussed in the Engine chapter.

**ServerMode**
Determines the execution mode of the server (“server model”). Discussed in the Engine chapter.

**RemoteAccess**
Parameter in firebird.conf and databases.conf provides an efficient, configurable replacement for hard-coded rules limiting access to security3.fdb. It can also be used to configure limited remote access to any other database, including non-default security databases.

By default RemoteAccess is enabled for all databases except the security database. If you intend using more than one dedicated security database, then disabling remote access to it (or them) via databases.conf is recommended.

For stricter security, server-wide, you can set RemoteAccess to false in firebird.conf and use entries in database.conf to re-enable it for specific databases.

*RemoteAccess* is a Boolean. It can be expressed with either true/false, 1/0 or Yes/No.

**WireCompression**
Alex Peshkov
Parameter in firebird.conf or databases.conf, enabling or disabling compression of data over the wire at global or individual database level.

The default setting is disabled (= False). Settings and environment must be correct at both server and client for WireCompression to take effect:

- To enable it at the server side, in firebird.conf and/or databases.conf, change the setting to True
• To activate Wirecompression from the client side, pass the appropriate tag in the config item of the DPB or SPB call:

isc_dbp_config/isc_sbp_config <string-length> "WireCompression=true"

• Both server and client versions must be Firebird 3 or greater (protocol >=13)

See Tracker item CORE-733.

IPv6V6Only
Michael Kubecek

Parameter in firebird.conf only. (TCP ports are created before any connection is established.)

Firebird 3 supports IPv6 connections, on both client and server sides.

Server

By default, the Firebird server listens on the zero IPv6 address (::) and accepts all incoming connections, whether IPv4 or IPv6, and IPv6V6Only is set to false (=0). If it is set to true, the server, still listening implicitly or explicitly on the zero IPv6 address, will accept only IPv6 connections.

Note

A different listening address, either IPv4 or IPv6, can be set using the RemoteBindAddress parameter. If an IPv4 address or a non-zero IPv6 address is used, the IPv6V6Only directive has no effect.

On POSIX platforms, in Classic mode, the parameters RemoteBindAddress, RemoteServicePort and RemoteServiceName are ignored by fbserver, since the listening socket is set up by (x)inetd. The listening address and/or port need to be set in the (x)inetd.

IPv6V6Only is a Boolean. It can be expressed with either true/false, 1/0 or Yes/No.

Client

The standard text form of an IPv6 address uses the colon character to separate the four groups of digits. In the connection string, the IPv6 address must be enclosed in square brackets, to resolve the ambiguity with the use of the colon as the separator between the host IP address and the database path. For example:

connect '[2014:1234::5]:test';
connect '[2014:1234::5]/3049:/srv/firebird/test.fdb';

Notes

For consistency, square brackets can be optionally used around an IPv4 address or a domain name.

If a domain name is used in connection string, all addresses (IPv4 and IPv6) are tried in the order returned by resolver until a connection is established. If all attempts fail, the client fails to connect.
Parameters Changed or Enhanced

The following parameters have been changed or enhanced:

ExternalFileAccess

Entries in the “Restrict” list of the ExternalFileAccess parameter can be used to mangle file names with relative paths.

Entries in the “Restrict” list were already used to mangle file names with no path component. For example, with

```
ExternalFileAccess = /opt/extern
```

and the following sequence of commands:

```
SQL> create table qq external file 'zz' (x int);
SQL> insert into qq values(1);
SQL> commit;
```

the file /opt/extern/zz will be created.

But if something like this is submitted,

```
create table qq external file 'dir/zz' (x int);
```

the result is an error about denied access to file /opt/firebird/bin/dir/zz.

The improvement avoids this gap by mangling the file name in accord with the value of the parameter and, if necessary, creating the missing path components, such as ‘dir’ in the example above.

Parameters Removed or Deprecated

The following parameters have been removed or deprecated:

RootDirectory

In older version, this parameter provided a superfluous option for recording the file system path to Firebird’s “root” files (firebird.conf, the security database and so on).

LegacyHash

This parameter used to make it possible to use the old security.fdb from Firebird 1.X installations after it had been subjected to an upgrade script and thence to enable or disable use of the obsolete DES hash encrypting algorithm. It is no longer supported.
**OldSetClauseSemantics**

This parameter enabled temporary support for an implementation fault in certain sequences of SET clauses in versions of Firebird prior to v.2.5. It is no longer available.

**OldColumnNaming**

This parameter temporarily enabled legacy code support for an old InterBase/Firebird 1.0 bug that generated unnamed columns for computed output which was not explicitly aliased in the SELECT specification. It is no longer available.

**LockGrantOrder**

This parameter used to allow the option to have Firebird's Lock Manager emulate InterBase v3.3 lock allocation behaviour, whereby locks would be granted in no particular order, as soon as soon as they were available, rather than by the normal order (first-come, first-served). The legacy option is no longer supported.

**Obsolete Windows priority settings**

UsePriorityScheduler, PrioritySwitchDelay and PriorityBoost, which were marginally relevant to obsolete processors on obsolete Windows versions, are no longer supported.
Security improvements in Firebird 3 include:

## Location of User Lists

Alex Peshkov

**CORE-685**

Firebird now supports an unlimited number of security databases. Any database may act as a security database and can be a security database for itself.

Use `databases.conf` to configure a non-default security database. This example configures `/mnt/storage/private.security.fdb` as the security database for the first and second databases:

```plaintext
first = /mnt/storage/first.fdb
  { SecurityDatabase = /mnt/storage/private.security.fdb }
second = /mnt/storage/second.fdb
  { SecurityDatabase = /mnt/storage/private.security.fdb }
```

Here we use third database as its own security database:

```plaintext
third = /mnt/storage/third.fdb
  { SecurityDatabase = third }
```

**Note**

The value of the `SecurityDatabase` parameter can be a database alias or the actual database path.

---

### Creating an Alternative Security Database

To start using a separate, non-default security database, the first step is to create it, unless it already exists. An embedded `isql` connection is used:

```plaintext
> isql -user sysdba
```
Security

SQL> create database '/mnt/storage/private.security.fdb';

Now connect to any database which will be served by the security database you are currently preparing, in order to create its SYSDBA user:

    SQL> connect first;
    SQL> create user sysdba password 'sysdba-in-private-security-password';
    SQL> commit;
    SQL> exit;

Database Encryption

Alex Peshkov

CORE-657

With Firebird 3 comes the ability to encrypt data stored in database. Not all of the database file is encrypted: just data, index and blob pages.

To make it possible to encrypt a database you need to obtain or write a database crypt plug-in.

Note

The sample crypt plug-in in examples/dbcrypt does not perform real encryption, it is merely a sample of how to go about it.

Secret Key

The main problem with database encryption is how to store the secret key. Firebird provides a helper to transfer that key from the client but that does not imply that storing the key on a client is the best way: it is no more than a possible alternative. A very bad option is to keep the key on the same disk as the database.

Encryption Tasks

To separate encryption and key access efficiently, a database crypt plug-in is split into two parts: encryption itself and the secret key holder. This may be an efficient approach for third-party plug-ins when you want to use some good encryption algorithm but you have your own secret way to store a key.

Encrypting a Database

Once you have decided on a crypt plug-in and a key, you can encrypt the database with:

    ALTER DATABASE ENCRYPT WITH <PLUGIN_NAME> [ KEY <key-name> ]

The optional KEY argument allows the key name to be passed in the command, if the plug-in requires it.
Security

Encryption will start right after this statement commits and will be performed in background. Normal database activity is not disturbed during encryption.

**Monitoring Encryption**

During encryption, progress can be monitored using the field MON$CRYPT_PAGE in the pseudo-table MON$DATABASE. For example:

```sql
select MON$CRYPT_PAGE * 100 / MON$PAGES
from MON$DATABASE
```

The example query will output the percentage of encryption completed so far.

You can also watch the database header page using repeated calls to `gstat -e`. If the database has been encrypted, `gstat -h` can also provide limited information about encryption state.

**Decrypting a Database**

To decrypt the database do:

```
ALTER DATABASE DECRYPT
```

For Linux, an example plug-in named `libDbCrypt_example.so` can be found in the `/plugins/` sub-directory.

---

**New Authentication Method in Firebird 3**

Alex Peshkov

All of the code related to authentication is plug-in-enabled. Though Firebird performs the generic work, like extracting authentication data from a network message or putting it into such messages as appropriate, all the activity related to calculating hashes, storing data in databases or elsewhere, using specific prime numbers and so on is done by plug-ins.

Firebird 3 has new method of user authentication implemented as a default plugin: secure remote password (SRP) protocol. Quoting from [Wikipedia](https):

> “The SRP protocol creates a large private key shared between the two parties in a manner similar to Diffie-Hellman key exchange, then verifies to both parties that the two keys are identical and that both sides have the user’s password. In cases where encrypted communications as well as authentication are required, the SRP protocol is more secure than the alternative SSH protocol and faster than using Diffie-Hellman key exchange with signed messages. It is also independent of third parties, unlike Kerberos.”

SSH needs key pre-exchange between server and client when placing a public key on the server to make it work. SRP does not need that. All a client needs are login and password. All exchange happens when the connection is established.

Moreover, SRP is resistant to “man-in-the-middle” attacks.
Use of the new authentication method is not compatible with old security databases and passwords from them. However, an upgrade procedure is available to migrate users from a Firebird 2.x security2.fdb database. For instructions, see Upgrading a v.2.x Security Database in the Compatibility chapter.

Use of an old security database can be supported with the Legacy_Auth authentication plug-in, but this kills the security benefits of Firebird 3.

The Firebird 3 client is built to make it possible to talk to old servers with the default configuration.

### SSL/TLS Support

**CORE-3251**

So, the answer to the question “Does Firebird use SSL/TLS for password validation?” is “yes and no”. The “No” answer comes because, by default, SSL is not used. That is due to a minor licensing incompatibility between Firebird and OpenSSL, the most popular SSL implementation.

The “Yes” applies because anyone is free to write an authentication plug-in that uses SSL and TLS.

### Increased Password Length

**CORE-1898**

Implementation of SRP in our plugin has increased the password length from 8 bytes to 20 or more. Because of the use of SHA1 for hashes, maximum security is provided for passwords up to 20 characters in length. Longer passwords can be used without restriction but there is a remote possibility of hash collisions between passwords that differ beyond the 20th byte. Just be aware of the possibility that any password longer than 20 characters password could share the same hash with some shorter password so, theoretically, they could be attacked using brute force.

A custom SRP plug-in could be built quite easily, using a hash algorithm that would guarantee unique hashes for longer passwords.

### Tip

The increased length limit means the default SYSDBA password in Windows and MacOS installations is the full 'masterkey' string (9 chars), no longer ‘masterke’ (8 chars) as in older versions!

Support for the LegacyHash and Authentication parameters in firebird.conf has been dropped. Authentication is overtaken by an AuthServer parameter in firebird.conf or elsewhere.

### The Authentication Plug-in

The Authentication plug-in comprises three parts:

- **Client**—prepares data at the client to be sent to server on client
- **Server**—validates password for correctness
• User Manager—adds, modifies and deletes users on the server. It is not needed if some external authentication method, such as Windows trusted authentication, is used.

All three parts are actually separate plug-ins which should be configured separately in firebird.conf. Let's look at an example of configuring a server to accept connections from old clients. The default setting are:

```plaintext
AuthServer = Srp, Win_Sspi
UserManager = Srp
```

To enable access from old clients, AuthServer needs to be changed:

```plaintext
AuthServer = Srp, Win_Sspi, Legacy_Auth
```

If we also want to manage the list of users in the old format we must add:

```plaintext
UserManager = Legacy_UserManager
```

### Multiple User Managers

Multiple user managers can be enumerated in firebird.conf. The first member of the list of user managers is the default. Selecting from `sec$users` would produce something like the following:

```sql
SQL> select SEC$USER_NAME, SEC$PLUGIN from sec$users;
```

<table>
<thead>
<tr>
<th>SEC$USER_NAME</th>
<th>SEC$PLUGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDBA</td>
<td>Legacy_UserManager</td>
</tr>
<tr>
<td>SYSDBA</td>
<td>Srp</td>
</tr>
<tr>
<td>QA_USER1</td>
<td>Srp</td>
</tr>
<tr>
<td>QA_USER2</td>
<td>Srp</td>
</tr>
<tr>
<td>QA_USER3</td>
<td>Srp</td>
</tr>
<tr>
<td>QA_USER4</td>
<td>Srp</td>
</tr>
<tr>
<td>QA_USER5</td>
<td>Srp</td>
</tr>
<tr>
<td>GUEST</td>
<td>Srp</td>
</tr>
<tr>
<td>SHUT1</td>
<td>Srp</td>
</tr>
<tr>
<td>SHUT2</td>
<td>Srp</td>
</tr>
<tr>
<td>QATEST</td>
<td>Srp</td>
</tr>
</tbody>
</table>

There might well be two users named SYSDBA in such a list, because each user manager has its own SYSDBA.

### Notes

- All user management commands can have the `USING PLUGIN` clause, whose purpose is to enable selection of a particular `UserManager` plug-in from the list in firebird.conf.
- The default user manager in firebird.conf is `Srp`. If you need to manage legacy logins, set it to `Legacy_UserManager`, e.g., `UserManager = Legacy_UserManager,Srp`.
- `UserManager` can be configured at database level, in `databases.conf`. 
"Over the wire" Connection Encryption

Alex Peshkov

CORE-672 ...

All network traffic in Firebird 3 may be optionally encrypted. As with authentication, plug-ins are used for encrypting and decrypting network traffic.

The default plug-in is **arc4** (Alleged RC4). It is eminently possible to write your own crypt plug-in to encrypt data travelling over the wire. Whatever you use for your plug-in, it is necessary to use the Firebird 3 version of the fbclient library.

The Secret Session Key

The challenge with use of a symmetric cypher is where to get a key for it. Firebird assumes that such a key, also called a **secret session key**, is produced by the authentication plug-in at the connection establishment phase. SRP meets this requirement just fine by producing a cryptographically strong session key.

**Tip**

If you want to use encryption with an authentication plug-in that does not provide the session key and agree to use some pre-defined key, say, one stored at the client side as a file and on the server in the security database for that specific client, then make that plug-in inform Firebird that it does have a session key.

Specifications for the Key

Specifications for the key's size, its format, how it is calculated and verified, etc., are not generalised. The key’s format and other details are specific to the wire encryption/decryption plug-in.

In particular, RC4 uses a symmetric key which can have any length, while the key produced by SRP has a length of 20 bytes. That key is a SHA-1 hash on SRP's session key and some other SRP-related things, such as user name.

Exporting a Key from an Authentication Plug-in

To export a key from your authentication plug-in, use the ServerBlock or the ClientBlock interface. One of these is always passed to the server/client part of an authentication plug-in. Both have a "newKey" method that returns a pointer to the CryptKey interface. That interface in turn has the methods **setSymmetric** and **setAsymmetric** for storing the symmetric or asymmetric key in the interface, i.e., exporting that key.

Mapping of Users to Objects

Alex Peshkov
Firebird 3 introduces new SQL privileges to map access between users and groups and security objects and between databases. See Tracker item CORE-1900.

With Firebird now supporting multiple security databases, some new problems arise that could not occur with a single, global security database. Clusters of databases using the same security database were efficiently separated. Mappings provide the means to achieve the same efficiency when multiple databases are using their own security databases. Some cases require control for limited interaction between such clusters. For example:

- when EXECUTE STATEMENT ON EXTERNAL DATA SOURCE requires some data exchange between clusters
- when server-wide SYSDBA access to databases is needed from other clusters, using services.
- comparable problems that have existed on Firebird 2.1 and 2.5 for Windows, due to support for Trusted User authentication: two separate lists of users—one in the security database and another in Windows, with cases where it was necessary to relate them. An example is the demand for a ROLE granted to a Windows group to be assigned automatically to members of that group.

The single solution for all such cases is mapping the login information assigned to a user when it connects to a Firebird server to internal security objects in a database—CURRENT_USER and CURRENT_ROLE.

**The Mapping Rule**

The mapping rule consists of four pieces of information:

1. mapping scope—whether the mapping is local to the current database or whether its effect is to be global, affecting all databases in the cluster, including security databases
2. mapping name—an SQL identifier, since mappings are objects in a database, like any other
3. the object FROM which the mapping maps. It consists of four items:
   - The authentication source
     - plug-in name or
     - the product of a mapping in another database or
     - use of server-wide authentication or
     - any method
   - The name of the database where authentication succeeded
   - The name of the object from which mapping is performed
   - The type of that name—user name | role | OS group—depending upon the plug-in that added that name during authentication

   Any item is accepted but only type is required.
4. the object TO which the mapping maps. It consists of two items:
   - The name of the object TO which mapping is performed
   - The type, for which only USER or ROLE is valid
**Syntax for MAPPING Objects**

Mappings are defined using the following set of DDL statements:

```
{CREATE | ALTER | CREATE OR ALTER} [GLOBAL] MAPPING name
USING {
    PLUGIN name [IN database] | ANY PLUGIN [IN database | SERVERWIDE] |
    MAPPING [IN database] | '*' [IN database]}
FROM {ANY type | type name}
TO {USER | ROLE} [name]
--
DROP [GLOBAL] MAPPING name
```

**Description**

- Any mapping may be tagged as GLOBAL.

Global mapping works best if a Firebird 3 or higher version database is used as the security database. If you plan to use another database for this purpose—using your own provider, for example—then you should create a table in it named RDB$MAP, with the same structure as RDB$MAP in a Firebird 3 database and with SYSDBA-only write access.

**Beware!**

If global and local mappings of the same name exist then know and make it known that they are different objects!

- The CREATE, ALTER and CREATE OR ALTER statements use the same set of options. The name (identifier) of a mapping is used to identify it, as in other DDL command sets.

- The USING clause has a highly complicated set of options:
  - an explicit plug-in name means it will work only for that plug-in
  - it can use any available plug-in; although not if the source is the product of a previous mapping
  - it can be made to work only with server-wide plug-ins
  - it can be made to work only with previous mapping results
  - it can be left to use any method, using the asterisk (*) argument
  - it can be provided with the name of the database that originated the mapping for the FROM object

**Note**

This argument is not valid for mapping server-wide authentication.

- The FROM clause takes a mandatory argument, the *type* of the object named.
  - When mapping names from plug-ins, *type* is defined by the plug-in.
  - When mapping the product of a previous mapping, *type* can be only USER or ROLE.
If an explicit name is provided, it will be taken into account by this mapping. Use the ANY keyword to work with any name of the given type.

- In the TO clause, the USER or ROLE to which the mapping is made must be specified. NAME is optional: if it is not supplied, the name from the originating mapping is used.

**Examples**

The examples use the CREATE syntax. Usage of ALTER is exactly the same and the usage of DROP should be obvious.

1. Enable use of Windows trusted authentication in all databases that use the current security database:

   ```sql
   CREATE GLOBAL MAPPING TRUSTED_AUTH
   USING PLUGIN WIN_SSPI
   FROM ANY USER
   TO USER;
   ```

2. Enable SYSDBA-like access for windows admins in current database:

   ```sql
   CREATE MAPPING WIN_ADMINS
   USING PLUGIN WIN_SSPI
   FROM Predefined_Group
   DOMAIN_ANY_RID_ADMINS
   TO ROLE RDB$ADMIN;
   ```

   **Note**
   
   The group DOMAIN_ANY_RID_ADMINS does not exist in Windows, but such a name would be added by the win_sspi plug-in to provide exact backwards compatibility.

3. Enable a particular user from another database to access the current database with another name:

   ```sql
   CREATE MAPPING FROM_RT
   USING PLUGIN SRP IN "rt"
   FROM USER U1 TO USER U2;
   ```

   **Important**
   
   Database names or aliases will need to be enclosed in double quotes on operating systems that have case-sensitive file names.

4. Enable the server's SYSDBA (from the main security database) to access the current database. (Assume that the database is using a non-default security database):

   ```sql
   CREATE MAPPING DEF_SYSDBA
   USING PLUGIN SRP IN "security.db"
   FROM USER SYSDBA
   TO USER;
   ```
5. Ensure users who logged in using the legacy authentication plug-in do not have too many privileges:

```sql
CREATE MAPPING LEGACY_2_GUEST
    USING PLUGIN legacy_auth
    FROM ANY USER
    TO USER GUEST;
```

**Legacy Mapping Rule**

Previous versions of Firebird have one hard-coded global default rule: users authenticated in the security database are always mapped into any database one-to-one. It is a safe rule: it makes no sense for a security database not to trust itself!

For backward compatibility this rule is retained in Firebird 3.

Generic mapping is used to set the rule defining the user name under which the user accesses a database when performing a request from a database using one security database to a database using a different one, or when server-wide authentication, such as `win_sspi`, is used. The rule comes into action whenever the Firebird engine is processing the authentication block associated with a request to connect to a database.

**Mapping Trusted Users to CURRENT_USER and CURRENT_ROLE**

Each record in an authentication block contains the name of the plug-in that added it, the type of record (user name, OS group, role, etc.), the name of an object of the specified type (user SYSDBA, role PUBLIC, group DOMAIN_ADMINS) and the name of the security database in which authentication took place. Under server-wide authentication, the security database name could be NULL. The mapping rule assesses all these parameters to define the value to assign to CURRENT_USER and CURRENT_ROLE in the resulting attachment.

In Firebird 3, an explicit mapping must exist in systems with server-wide “trusted user” authentication enabled, including Win_Sspi authentication on Windows, in order for the system user's user name to be assigned to the context variables CURRENT_USER and CURRENT_ROLE.

**International Character Sets for User Accounts**

Alex Peshkov

For creating user names and passwords, the new authentication plug-ins bring a degree of flexibility with regard to character sets. To make use of international credentials support, it is necessary to use SRP or any other set of authentication plug-ins that supports the new model.

**Advice**

Use of the legacy authentication plug-in is not recommended, except where it is necessary for connecting a client to a server version older than Firebird 3. Legacy authentication has never supported international characters in user names and passwords. This restriction is unchanged for Firebird 3 legacy authentication.

When writing authentication plug-ins it is unnecessary to be concerned about character sets, since all data exchange with the new plug-ins is done using UTF8. Just be prepared to handle user names and passwords that contain characters beyond the range of the 7-bit ASCII character subset.
**Conditions for Non-ASCII Logins**

The simple rule to follow for using non-ASCII user names and passwords is that the character set and code page selected in the client for SQL input must match those in which the credentials are stored on the server and, obviously, the credentials must be supplied in the specified character set. On POSIX terminals and most GUI clients this condition is satisfied. However, there are three fairly common situations where problems could arise:

1. connecting via a Windows text console with the default OEM code page selected
2. omitting the SET NAMES command in *isql*
3. running scripts

Examining these situations in more detail:

1. All operating systems except Windows use same the same character set (code page, locale), by default, for all programs. In Windows, for historical reasons, GUI applications use an ANSI code page, but applications started on the command-line console emulator use the corresponding OEM code page. As an example, the Russian version of Windows is uses code page 1251 (ANSI Russian) for GUI applications and code page 866 (OEM Russian) for command-line applications.

   The Firebird client always uses the ANSI code page. In order to provide the fully functional behaviour of a console application, the console must be switched to the ANSI codepage (chcp 1251 for our Russian example).

   The Windows weirdness does not stop there. Parameters passed to a Windows application, but not to 16-bit DOS, are always passed in the ANSI encoding, even if it was started from a command-line console running in the default OEM mode. This might give the illusion that Firebird utilities run correctly out-of-the-box on an OEM console. However, they run correctly only as far as the international information supplied in a command line, e.g.,

   ```
   C:> isql server:database -user <intl-login> -pas <intl-password>
   ```

   This works. However, as soon as you try to use a CONNECT or CREATE DATABASE command inside *isql*, or to enter the password from the OEM terminal, you will encounter problems. This is inconvenient and we apologise for that. The OEM console issues are in the plan to be fixed in a future version.

2. Currently, command-line parameters and the CONNECT and CREATE DATABASE commands in *isql* are not affected by SET NAMES or the `-Charset` parameter. All the other commands, particularly CREATE/ALTER/DROP USER, are affected by them and thus, the character set must be specified. This is very important because, in future versions, the plan is to make all usage of international credentials depend on a character set that is explicitly specified.

   Take careful note of one very confusing use case, an attempt to set a non-ASCII password for the current user:

   ```
   SQL> alter current user set password '<intl-password>'
   ```

   This command will succeed, even if the character set has not been set correctly. However, a subsequent attempt to log in with the modified password will fail.
3. In scripts, the first requirement is to set the correct character set for the SQL server that is to run the script, using a `SET NAMES` statement. For Firebird 3, it is essential to pay attention to the locale and code page settings in the environment in which the script is to execute. They must match the character set that is set in the script.

Setting the locale correctly affects particularly the credentials passed to the server when attaching to it. The rest of the script should proceed successfully even without configuring the environment properly.

In summary, if you plan to use international character credentials in the script, you **DO** need to attend to the international settings in your environment.

**Reminder**

User names are SQL identifiers and thus conform to the same rules, i.e., enclosed in double quotes when containing international characters or when case-sensitivity is required.

---

**SQL Features for Managing Access**

Changes in architecture, stiffening of rules for security and data integrity, along with a bucket list of feature requests, have given rise in this release to a number of new SQL commands for managing users and access to objects.

**SQL-driven User Management**

Alex Peshkov

The SQL set of DDL commands for managing user accounts has been enhanced in Firebird 3, thus improving the capability to manage (add, modify or delete) users in a security database from a regular database attachment.

**Syntax Forms**

```
CREATE USER username [ options_list ] [ TAGS ( tag [, tag [, tag ...] ] ) ]
ALTER USER username [ SET ] [ options_list ] [ TAGS ( tag [, tag [, tag ...] ] ) ]
ALTER CURRENT USER [ SET ] [ options_list ] [ TAGS ( tag [, tag [, tag ...] ] ) ]
CREATE OR ALTER USER username [ SET ] [ options ] [ TAGS ( tag [, tag [, tag ...] ] ) ]
DROP USER username [ USING PLUGIN plugin_name ]
```

**OPTIONS** is a (possibly empty) list with the following options:

```
PASSWORD 'password'
FIRSTNAME 'string value'
MIDDLENAME 'string value'
LASTNAME 'string value'
ACTIVE
INACTIVE
USING PLUGIN plugin_name
```

Each TAG may have one of two forms:
TAGNAME = 'string value'

or the DROP TAGNAME tag form to remove a user-defined attribute entirely:

DROP TAGNAME

**Note**
The tagname side of the name/value pair can be any valid SQL identifier.

**Older Methods Deprecated**

From Firebird 3.0, multiple security databases are supported. This capability is not supported by either the gsec utility or the Services API. Use of both of these methods is deprecated.

**Usage Details**

The CREATE USER, CREATE OR ALTER USER and DROP USER clauses are available only for SYSDBA or another user granted the RDB$ADMIN role in the security database (and logged in under that role, of course.)

The PASSWORD clause is **required** when creating a new user.

An ordinary user can ALTER his own password, real name attributes and tags. Any attempt to modify another user will fail, as will an attempt to make “self” inactive or active.

If you want to modify “self”, you can use the simplified form **ALTER CURRENT USER**.

At least one of PASSWORD, FIRSTNAME, MIDDLENAME, LASTNAME, ACTIVE, INACTIVE or TAGS must be present in an ALTER USER or CREATE OR ALTER USER statement.

It is not a requirement to use any of the clauses FIRSTNAME, MIDDLENAME and LASTNAME. Any of them may be left empty or used to store short information about the user.

The INACTIVE clause is used to disable the user's login capability without dropping it. The ACTIVE clause restores that login ability.

TAGS is a list of end-user defined attributes. The length of the string value should not exceed 255 bytes.

Setting a list of tags for the user retains previously set tags if they are not mentioned in the current list.

**Note**
A UID or GID that was entered by the deprecated gsec utility is treated as a tag in the SQL interface.

**Examples**

Generic:
CREATE USER superhero PASSWORD 'test';
ALTER USER superhero SET FIRSTNAME 'Clark' LASTNAME 'Kent';
CREATE OR ALTER USER superhero SET PASSWORD 'IdQfA';
DROP USER superhero;
ALTER CURRENT USER SET PASSWORD 'SomethingLongEnough';

Working with tags:

ALTER USER superhero SET TAGS (a='a', b='b');

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
</tr>
</tbody>
</table>

ALTER USER superhero SET TAGS (b='x', c='d');

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>B</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>d</td>
</tr>
</tbody>
</table>

ALTER USER superhero SET TAGS (drop a, c='sample');

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>sample</td>
</tr>
</tbody>
</table>

Displaying the list of users:

SELECT CAST(U.SEC$USER_NAME AS CHAR(20)) LOGIN,
       CAST(A.SEC$KEY AS CHAR(10)) TAG,
       CAST(A.SEC$VALUE AS CHAR(20)) "VALUE",
       SEC$PLUGIN "PLUGIN"
FROM SEC$USERS U LEFT JOIN SEC$USER_ATTRIBUTES A
  ON U.SEC$USER_NAME = A.SEC$USER_NAME;

<table>
<thead>
<tr>
<th>LOGIN</th>
<th>TAG</th>
<th>VALUE</th>
<th>PLUGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDBA</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>Srp</td>
</tr>
<tr>
<td>SUPERHERO</td>
<td>B</td>
<td>x</td>
<td>Srp</td>
</tr>
<tr>
<td>SUPERHERO</td>
<td>C</td>
<td>sample</td>
<td>Srp</td>
</tr>
<tr>
<td>SYSDBA</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>Legacy_UserManager</td>
</tr>
</tbody>
</table>

*Note*

Output depends upon the user management plug-in. If the legacy plug-in is used, bear in mind that some options are not supported and will simply be ignored.

**SET ROLE**

Alex Peshkov

See Tracker item CORE-1377.
The SQL2008-compliant operator SET ROLE allows the CURRENT_ROLE context variable to be set to one that has been granted to the CURRENT_USER or to a user assigned to the database attachment as trusted (SET TRUSTED ROLE).

Syntax Pattern for SET ROLE

Enable CURRENT_USER access to a role that has been previously granted:

```
SET ROLE <rolename>
```

Example of SET ROLE Usage

```
SET ROLE manager;
select current_role from rdb$database;
```

Displays:

```
ROLE
-----------------
MANAGER
```

**SET TRUSTED ROLE**

The idea of a separate SET TRUSTED ROLE command is that, when the trusted user attaches to a database without providing any role info, SET TRUSTED ROLE makes a trusted role (if one exists) the CURRENT_ROLE without any additional activity, such as setting it in the DPB.

A trusted role is not a specific type of role but may be any role that was created using CREATE ROLE, or a predefined system role such as RDB$ADMIN. It becomes a trusted role for an attachment when the security objects mapping subsystem finds a match between the authentication result passed from the plug-in and a local or global mapping for the current database. The role may be one that is not even granted explicitly to that trusted user.

Notes

- A trusted role is not assigned to the attachment by default. It is possible to change this behaviour using an appropriate authentication plug-in and a CREATE/ALTER MAPPING command.
- Whilst the CURRENT_ROLE can be changed using SET ROLE, it is not always possible to revert using the same command, because it performs an access rights check.

Syntax Pattern

Enable access to a trusted role, if the CURRENT_USER is logged in under Trusted User authentication and the role is available:

```
SET TRUSTED ROLE
```
An example of the use of a trusted role is assigning the system role RDB$ADMIN to a Windows administrator when Windows trusted authentication is in use.

**GRANT/REVOKE Rights GRANTED BY Specified User**

Alex Peshkov

Previously, the grantor or revoker of SQL privileges was always the current user. This change makes it so that a different grantor or revoker can be specified in GRANT and REVOKE commands.

**Syntax Pattern**

```sql
grant <right> to <object> [ { granted by | as } [ user ] <username> ]
revoke <right> from <object> [ { granted by | as } [ user ] <username> ]
```

The **GRANTED BY** clause form is recommended by the SQL standard. The alternative form using **AS** is supported by Informix and possibly some other servers and is included for better compatibility.

**Example (working as SYSDBA)**

```sql
create role r1;
grant r1 to user1 with admin option;
grant r1 to public granted by user1;

-- (in isql)
show grant;
/* Grant permissions for this database */
GRANT R1 TO PUBLIC GRANTED BY USER1
GRANT R1 TO USER1 WITH ADMIN OPTION
```

**REVOKE ALL ON ALL**

When a user is removed from the security database or another authentication source, this new command is useful for revoking its access to all objects in the database.

**Syntax Pattern**

```sql
REVOKE ALL ON ALL FROM [USER] username
REVOKE ALL ON ALL FROM [ROLE] rolename
```

**Example**

```
# gsec -del guest
# isql employee
fbs bin # ./isql employee
Database: employee
SQL> REVOKE ALL ON ALL FROM USER guest;
SQL>
```
User Privileges for Metadata Changes

Dmitry Yemanov
with Roman Simakov

In Firebird 3, the system tables are read-only. This SQL syntax provides the means to assign metadata write privileges to specified users or roles for specified objects. See Tracker item CORE-735.

Note

Some people have been applying the nickname “DDL privileges” to this feature. Don't confuse it with “DDL triggers”! A more useful nickname would be “Metadata privileges”.

Syntax Patterns

Granting metadata privileges:

```
GRANT CREATE <object-type>
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
GRANT ALTER ANY <object-type>
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
GRANT DROP ANY <object-type>
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
```

Revoking metadata privileges:

```
REVOKE [GRANT OPTION FOR] CREATE <object-type>
  FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] ALTER ANY <object-type>
  FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] DROP ANY <object-type>
  FROM [USER | ROLE] <user-name> | <role-name>;
```

Special form for database access:

```
GRANT CREATE DATABASE TO [USER | ROLE] <user-name> | <role-name>;
GRANT ALTER DATABASE
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
GRANT DROP DATABASE
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
REVOKE CREATE DATABASE FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] ALTER DATABASE
  FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] DROP DATABASE
  FROM [USER | ROLE] <user-name> | <role-name>;
```

Notes on Usage

- `<object-type>` can be any of the following:
  
  CHARACTER SET  COLLATION  DOMAIN  EXCEPTION
• If the ANY option is used, the user will be able to perform any operation on any object
• If the ANY option is absent, the user will be able to perform operations on the object only if he owns it
• If the ANY option was acquired via a GRANT operation then, to revoke it, the REVOKE operation must accord with that GRANT operation

Example

GRANT CREATE TABLE TO Joe;
GRANT ALTER ANY TABLE TO Joe;
REVOKE CREATE TABLE FROM Joe;

**GRANT EXECUTE Privileges for UDFs**
Dmitry Yemanov

**CORE-2554:** EXECUTE permission is now supported for UDFs (both legacy and PSQL based ones).

**Syntax Pattern**

```
GRANT EXECUTE ON FUNCTION <name> TO <grantee list>
[<grant option> <granted by clause>]
--
REVOKE EXECUTE ON FUNCTION <name> FROM <grantee list>
[<granted by clause>]
```

Note

The initial EXECUTE permission is granted to the function owner (user who created or declared the function).

**Improvement for Recursive Stored Procedures**
Alex Peshkov

A recursive stored procedure no longer requires the EXECUTE privilege to call itself. See Tracker item **CORE-3242**.

**Privileges to Protect Other Metadata Objects**

New SQL-2008 compliant USAGE permission is introduced to protect metadata objects other than tables, views, procedures and functions.
Syntax Pattern

GRANT USAGE ON <object type> <name> TO <grantee list>
[<grant option> <granted by clause>]
--
REVOKE USAGE ON <object type> <name> FROM <grantee list>
[<granted by clause>]
--
<object type> ::= {DOMAIN | EXCEPTION | GENERATOR | SEQUENCE | CHARACTER SET | COLLATION)

Notes

The initial USAGE permission is granted to the object owner (user who created the object).

In Firebird 3.0.0, only USAGE permissions for exceptions (CORE-2884) and generators/sequences (gen_id, next value for: CORE-2553) are enforced. Granting privileges for character sets, collations and domains is disabled, making these object types unavailable for any type of GRANT or REVOKE commands. Access to them is not subject to any form of enforcement, although this could change in future releases if it is deemed appropriate.

Pseudo-Tables with List of Users

CORE-2639.

To access lists of users and attributes, query the virtual tables SEC$USERS and SEC$USER_ATTRIBUTES.

Important

This feature depends highly on the user management plug-in. Take into an account that some options are ignored when using the legacy user management plug-in.

The pseudo-tables are much like the MONS family tables used for monitoring the server. The table is created on demand when you run the statement

SELECT * FROM SEC$USERS

or

SELECT * FROM SEC$USER_ATTRIBUTES

The output lists the users (or their attributes) in the security database that is configured for the current database and available for management to the current user. SEC$USERS includes a field indicating whether a user has the RDB$ADMIN role in the security database.
Chapter 8

Data Definition Language (DDL)

Quick Links

• BOOLEAN Data Type
• IDENTITY-Style Column
• Manage Nullability in Domains and Columns
• Modify Generators (Sequences)
• Alter Default Character Set
• BLOB Expressions in Computed Columns
• “Linger” Database Closure for Superserver
• New option in DROP SHADOW to Preserve the Shadow File
• New SQL for Managing Users and Access Privileges

DDL Enhancements

The following enhancements have been added to the SQL data definition language lexicon:

New Data Types

A fully-fledged Boolean type is introduced in this release, along with a surfaced emulation of the Microsoft-style “identity” column.

BOOLEAN Data Type
Adriano dos Santos Fernandes

The SQL-2008 compliant BOOLEAN data type (8 bits) comprises the distinct truth values TRUE and FALSE. Unless prohibited by a NOT NULL constraint, the BOOLEAN data type also supports the truth value UNKNOWN as the null value. The specification does not make a distinction between the NULL value of this data type and the truth value UNKNOWN that is the result of an SQL predicate, search condition, or boolean value expression: they may be used interchangeably to mean exactly the same thing.

As with many programming languages, the SQL BOOLEAN values can be tested with implicit truth values. For example, field1 OR field2 and NOT field1 are valid expressions.
**The IS Operator**

Predications use the operator IS [NOT] for matching. For example, field1 IS FALSE, or field1 IS NOT TRUE.

### Note

Equivalence operators ("=", "!=", "<>" and so on) are valid in all comparisons.

### Examples

```
CREATE TABLE TBOOL (ID INT, BVAL BOOLEAN);
COMMIT;

INSERT INTO TBOOL VALUES (1, TRUE);
INSERT INTO TBOOL VALUES (2, 2 = 4);
INSERT INTO TBOOL VALUES (3, NULL = 1);
COMMIT;

SELECT * FROM TBOOL
  ID   BVAL
  ======= ========
    1   <true>
    2   <false>
    3   <null>

-- Test for TRUE value
SELECT * FROM TBOOL WHERE BVAL
  ID   BVAL
  ======= ========
    1   <true>

-- Test for FALSE value
SELECT * FROM TBOOL WHERE BVAL IS FALSE
  ID   BVAL
  ======= ========
    2   <false>

-- Test for UNKNOWN value
SELECT * FROM TBOOL WHERE BVAL IS UNKNOWN
  ID   BVAL
  ======= ========
    3   <null>

-- Boolean values in SELECT list
SELECT ID, BVAL, BVAL AND ID < 2
  FROM TBOOL
  ID   BVAL
  ======= ======= ========
    1   <true>   <true>
    2   <false>  <false>
    3   <null>   <false>

-- PSQL Declaration with start value
DECLARE VARIABLE VAR1 BOOLEAN = TRUE;

-- Valid syntax, but as with a comparison
-- with NULL, will never return any record
```
SELECT * FROM TBOOL WHERE BVAL = UNKNOWN
SELECT * FROM TBOOL WHERE BVAL <> UNKNOWN

Notes
- Represented in the API with the FB_BOOLEAN type and FB_TRUE and FB_FALSE constants.
- The value TRUE is greater than the value FALSE.
- Although BOOLEAN is not implicitly convertible to any other datatype, it can be explicitly converted to and from string with CAST.

Keywords INSERTING, UPDATING and DELETING

To avoid ambiguities when used in Boolean expressions, the previously non-reserved keywords INSERTING, UPDATING and DELETING, which return True|False when tested in PSQL, have been made reserved words in all contexts. If you have used any of these words as identifiers for database objects, columns, variables or parameters, it will be necessary to redefine them, either with new names or by enclosing their identifiers in double quotes.

Identity Column Type
Adriano dos Santos Fernandes

An identity column is a column associated with an internal sequence generator. Its value is set automatically when the column is omitted in an INSERT statement.

Syntax Patterns

```
<column definition> ::= 
    <name> <type> GENERATED BY DEFAULT AS IDENTITY [ (START WITH <value>) ]<constraints>
```

When defining a column, the optional START WITH clause allows the generator to be initialised to a value other than zero. See Tracker ticket CORE-4199.

```
<alter column definition> ::= 
    <name> RESTART [ WITH <value> ]
```

A column definition can be altered to modify the starting value of the generator. RESTART alone resets the generator to zero; the optional WITH <value> clause allows the restarted generator to start at a value other than zero. See Tracker ticket CORE-4206.

Rules
- The data type of an identity column must be an exact number type with zero scale. Allowed types are thus SMALLINT, INTEGER, BIGINT, NUMERIC(x,0) and DECIMAL(x,0).
- An identity column cannot have DEFAULT or COMPUTED value.
**Notes**

- An identity column cannot be altered to become a regular column. The reverse is also true.
- Identity columns are implicitly NOT NULL (non-nullable).
- Uniqueness is not enforced automatically. A UNIQUE or PRIMARY KEY constraint is required to guarantee uniqueness.
- The use of other methods of generating key values for IDENTITY columns, e.g., by trigger-generator code or by allowing users to change or add them, is discouraged to avoid unexpected key violations.

**Example**

```sql
create table objects (
    id integer generated by default as identity primary key,
    name varchar(15)
);

insert into objects (name) values ('Table');
insert into objects (name) values ('Book');
insert into objects (id, name) values (10, 'Computer');

select * from objects;
```

```
ID  NAME
1   Table
2   Book
10  Computer
```

**Implementation Details**

Two new columns have been inserted in RDB$RELATION_FIELDS to support identity columns: RDB$GENERATOR_NAME and RDB$IDENTITY_TYPE.

- RDB$GENERATOR_NAME stores the automatically created generator for the column. In RDB$GENERATORS, the value of RDB$SYSTEM_FLAG of that generator will be 6.

- Currently, RDB$IDENTITY_TYPE will currently always store the value 1 (by default) for identity columns and NULL for non-identity columns. In the future this column will store the value 0, too (for ALWAYS) when Firebird implements support for this type of identity column.

**Manage Nullability in Domains and Columns**

A. dos Santos Fernandes

ALTER syntax is now available to change the nullability of a table column or a domain

**Syntax Pattern**

```
ALTER TABLE <table name> ALTER <field name> { DROP | SET } [NOT] NULL
ALTER DOMAIN <domain name> { DROP | SET } [NOT] NULL
```
**Notes**

The success of a change in a table column from NULL to NOT NULL is subject to a full data validation on the table, so ensure that the column has no nulls before attempting the change.

A change in a domain subjects all the tables using the domain to validation.

An explicit NOT NULL on a column that depends on a domain prevails over the domain. In this situation, the changing of the domain to make it nullable does not propagate to the column.

---

**Modify Generators (Sequences)**

More statement options have been added for modifying generators (sequences). Where previously in SQL the only option was `ALTER SEQUENCE <sequence name> RESTART WITH <value>`, now a full lexicon is provided and `GENERATOR` and `SEQUENCE` are synonyms for the full range of commands.

`RESTART` can now be used on its own to restart the sequence at its previous start or restart value. A new column `RDB$INITIAL_VALUE` is added to the system table `RDB$GENERATORS` to store that value.

**Syntax Forms**

```sql
{ CREATE | RECREATE } { SEQUENCE | GENERATOR } <sequence name> [ START WITH <value> ]
CREATE OR ALTER { SEQUENCE | GENERATOR } <sequence name> { RESTART | START WITH <value> }
ALTER { SEQUENCE | GENERATOR } <sequence name> RESTART [ WITH <value> ]
```

---

**Alter the Default Character Set**

A. dos Santos Fernandes

```sql
ALTER DATABASE
  ...
  SET DEFAULT CHARACTER SET <new_charset>
```

The alteration does not change any existing data. The new default character set is used only in subsequent DDL commands and will assume the default collation of the new character set.

---

**BLOB Expressions in Computed Columns**

Adriano dos Santos Fernandes

A substring from a BLOB column can now be used to define a computed column.

**For Example**

```sql
ALTER TABLE ATABLE
  ADD ABLOB
  COMPUTED BY (SUBSTRING(BLOB_FIELD FROM 1 FOR 20))
```
“Linger” Database Closure for Superserver

Sometimes it is desirable to have the Superserver engine keep the database open for a period after the last attachment is closed, i.e., to have it “linger” a while. It can help to improve performance at low cost, under conditions where the database is opened and closed frequently, by keeping resources “warm” for next time it is reopened.

Firebird 3.0 introduces an enhancement to `ALTER DATABASE` to manage this optional LINGER capability for databases running under Superserver.

**Syntax Form**

```
ALTER DATABASE SET LINGER TO {seconds};
ALTER DATABASE DROP LINGER;
```

**Usage**

To set linger for the database do:

```
ALTER DATABASE SET LINGER TO 30;  -- sets linger interval to 30 seconds
```

Either of the following forms will clear the linger setting and return the database to the normal condition (no linger):

```
ALTER DATABASE DROP LINGER;
ALTER DATABASE SET LINGER TO 0;
```

**Note**

Dropping LINGER is not an ideal solution for the occasional need to turn it off for some once-only condition where the server needs a forced shutdown. The `gfix` utility now has the `-NoLinger` switch, which will close the specified database immediately the last attachment is gone, regardless of the LINGER setting in the database. The LINGER setting is retained and works normally the next time.

The same one-off override is also available through the Services API, using the tag `isc_spb_prp_nolinger`, e.g. (in one line):

```
fbsvcmgr host:service_mgr user sysdba password xxx
   action_properties dbname employee prp_nolinger
```

See also Tracker ticket CORE-4263 for some discussion of the development of this feature.

**Option to Preserve Shadow File**

Alex Peshkov

The DROP SHADOW command has a new option to preserve the shadow file in the filesystem:


See Tracker ticket [CORE-4955](#).

### New SQL for Managing Users and Access Privileges

A number of new features and enhancements have been added to the DDL lexicon for managing users and their access to objects in databases. They are described in detail in Chapter 7, Security.

#### SQL-driven User Management

The SQL set of DDL commands for managing user accounts has been enhanced in Firebird 3, thus improving the capability to manage (add, modify or delete) users in a security database from a regular database attachment.

```sql
DROP SHADOW shadow_num
[DELET | PRESERVE) FILE]
```

#### SET ROLE and SET TRUSTED ROLE

The SQL2008-compliant operator SET ROLE allows the CURRENT_ROLE context variable to be set to one that has been granted to the CURRENT_USER or to a user assigned to the database attachment as trusted (SET TRUSTED ROLE).

#### GRANTED BY Clause for Privileges

Previously, the grantor or revoker of SQL privileges was always the current user. The GRANTED BY clause makes it so that a different grantor or revoker can be specified in GRANT and REVOKE commands.

#### REVOKE ALL ON ALL

When a user is removed from the security database or another authentication source, this new command is useful for revoking its access to all objects in the database.

#### GRANT/REVOKE Metadata Privileges

In Firebird 3, the system tables are read-only. This SQL syntax provides the means to assign metadata write privileges to specified users or roles for specified objects.

#### EXECUTE Privileges for UDFs

EXECUTE permission is now supported for UDFs (both legacy and PSQL based ones).

---

The command-line and shell utility `gsec` is deprecated from this release forward. It will continue to work with `security3.fdb` but it will not work with alternative security databases.

---

`gsec` is deprecated!

The command-line and shell utility `gsec` is deprecated from this release forward. It will continue to work with `security3.fdb` but it will not work with alternative security databases.
**GRANT/REVOKE USAGE**

New SQL-2008 compliant USAGE permission is introduced to protect metadata objects other than tables, views, procedures and functions.
In this chapter are the additions and improvements that have been added to the SQL data manipulation language subset in Firebird 3.0.

Quick Links

- Supplemental SQL 2008 Features for MERGE
- Window (Analytical) Functions
- Advanced PLAN Output
- SUBSTRING With Regular Expressions
- Inverse Hyperbolic Trig Functions
- Statistical Functions
- Enhancements to DATEADD() Internal Function
- TRIM() BLOB Arguments Lose 32 KB limit
- Alternatives for Embedding Quotes in String Literals
- SQL:2008-Compliant OFFSET and FETCH Clauses
- Prohibit Edgy Mixing of Implicit/Explicit Joins
- Support for Left-side Parameters in WHERE Clause
- RETURNING Clause Can be Aliased
- RETURNING Clause from Positioned Updates and Deletes
- Cursor Stability
- Improvements for Global Temporary Tables
- Improvements for DML Strings
- COUNT() Now Returns BIGINT
- SIMILAR TO Performance Improvement
- OR'ed Parameter in WHERE Clause
- A Little Dialect 1 Accommodation
- Embedded SQL (ESQL) Enhancements

Supplemental SQL 2008 Features for MERGE

Adriano dos Santos Fernandes

In summary, support for MERGE was supplemented with the introduction of these features:

- Addition of the DELETE extension (CORE-2005)
- Enabling the use of multiple WHEN MATCHED | NOT MATCHED clauses (CORE-3639) and ability to apply conditions to WHEN MATCHED | NOT MATCHED
• Addition of the RETURNING ... INTO ... clause (CORE-3020)

The purpose of MERGE is to read data from the source and INSERT or UPDATE in the target table according to a condition. It is available in DSQL and PSQL.

**Syntax Pattern**

```mermaid
textblock
<merge statement> ::=  
MERGE  
INTO <table or view> [ [AS] <correlation name> ]  
USING <table or view or derived table> [ [AS] <correlation name> ]  
ON <condition>  
<merge when>...  
<returning clause>  

<merge when> ::=  
<merge when matched> |  
<merge when not matched>  

<merge when matched> ::=  
WHEN MATCHED [ AND <condition> ] THEN  
{ UPDATE SET <assignment list> | DELETE }  

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

**Rules**

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

**Example**

MERGE INTO customers c  
USING  
(SELECT * FROM customers_delta WHERE id > 10) cd  
ON (c.id = cd.id)  
WHEN MATCHED THEN  
UPDATE SET name = cd.name  
WHEN NOT MATCHED THEN  
INSERT (id, name)  
VALUES (cd.id, cd.name)

**Notes**

A right join is made between the INTO (left-side) and USING tables using the condition. UPDATE is called when a record exists in the left table (INTO), otherwise INSERT is called.

As soon as it is determined whether or not the source matches a record in the target, the set formed from the corresponding (WHEN MATCHED / WHEN NOT MATCHED) clauses is evaluated in the order specified, to check their optional conditions. The first clause whose condition evaluates to true is the one which will be executed, and the subsequent ones will be ignored.

If no record is returned in the join, INSERT is not called.
Window (Analytical) Functions

Adriano dos Santos Fernandes

According to the SQL specification, window functions (also known as analytical functions) are a kind of aggregation, but one that does not “filter” the result set of a query. The rows of aggregated data are mixed with the query result set.

The window functions are used with the OVER clause. They may appear only in the SELECT list or the ORDER BY clause of a query.

Besides the OVER clause, Firebird window functions may be partitioned and ordered.

Syntax Pattern

<window function> ::= <window function name>([<expr> [, <expr> ...]]) OVER ( [PARTITION BY <expr> [, <expr> ...]] [ORDER BY <expr> [<direction>]] [nulls placement>] [, <expr> [<direction>] [nulls placement>] ...)
<direction> ::= {ASC | DESC}
>nulls placement> ::= NULLS {FIRST | LAST}

Aggregate Functions Used as Window Functions

All aggregate functions may be used as window functions, adding the OVER clause.

Imagine a table EMPLOYEE with columns ID, NAME and SALARY, and the need to show each employee with his respective salary and the percentage of his salary over the payroll.

A normal query could achieve this, as follows:

```sql
select id, department, salary, salary / (select sum(salary) from employee) portion from employee order by id;
```

Results

<table>
<thead>
<tr>
<th>id</th>
<th>department</th>
<th>salary</th>
<th>portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R &amp; D</td>
<td>10.00</td>
<td>0.2040</td>
</tr>
<tr>
<td>2</td>
<td>SALES</td>
<td>12.00</td>
<td>0.2448</td>
</tr>
</tbody>
</table>
The query is repetitive and lengthy to run, especially if EMPLOYEE happened to be a complex view.

The same query could be specified in a much faster and more elegant way using a window function:

```sql
select id, department, salary, salary / sum(salary) OVER () AS portion
from employee
order by id;
```

Here, `sum(salary) over ()` is computed with the sum of all SALARY from the query (the employee table).

**Partitioning**

Like aggregate functions, that may operate alone or in relation to a group, window functions may also operate on a group, which is called a "partition".

**Syntax Pattern**

```
<window function>(...) OVER (PARTITION BY <expr> [, <expr> ...])
```

Aggregation over a group could produce more than one row, so the result set generated by a partition is joined with the main query using the same expression list as the partition.

Continuing the employee example, instead of getting the portion of each employee's salary over the all-employees total, we would like to get the portion based on just the employees in the same department:

```sql
select id, department, salary, salary / sum(salary) OVER (PARTITION BY department) AS portion
from employee
order by id;
```

**Results**

<table>
<thead>
<tr>
<th>id</th>
<th>department</th>
<th>salary</th>
<th>portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R &amp; D</td>
<td>10.00</td>
<td>0.3448</td>
</tr>
<tr>
<td>2</td>
<td>SALES</td>
<td>12.00</td>
<td>0.6000</td>
</tr>
<tr>
<td>3</td>
<td>SALES</td>
<td>8.00</td>
<td>0.4000</td>
</tr>
<tr>
<td>4</td>
<td>R &amp; D</td>
<td>9.00</td>
<td>0.3103</td>
</tr>
<tr>
<td>5</td>
<td>R &amp; D</td>
<td>10.00</td>
<td>0.3448</td>
</tr>
</tbody>
</table>


## Ordering

The ORDER BY sub-clause can be used with or without partitions and, with the standard aggregate functions, make them return the partial aggregations as the records are being processed.

### Example

```sql
select id, salary, sum(salary) over (order by salary) cumul_salary
from employee
order by salary;
```

The result set produced:

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>cumul_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>8.00</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>17.00</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>37.00</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>37.00</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>49.00</td>
</tr>
</tbody>
</table>

Then cumul_salary returns the partial/accumulated (or running) aggregation (of the SUM function). It may appear strange that 37.00 is repeated for the ids 1 and 5, but that is how it should work. The ORDER BY keys are grouped together and the aggregation is computed once (but summing the two 10.00). To avoid this, you can add the ID field to the end of the ORDER BY clause.

It's possible to use multiple windows with different orders, and ORDER BY parts like ASC/DESC and NULLS FIRST/LAST.

With a partition, ORDER BY works the same way, but at each partition boundary the aggregation is reset.

All aggregation functions, other than LIST(), are usable with ORDER BY.

### Exclusive window functions

Beyond aggregate functions are the exclusive window functions, currently divided into ranking and navigational categories. Both sets can be used with or without partition and ordering, although the usage does not make much sense without ordering.

#### Ranking Functions

The rank functions compute the ordinal rank of a row within the window partition. In this category are the functions DENSE_RANK, RANK and ROW_NUMBER.

**Syntax**

```
<ranking window function> ::=  
```
The ranking functions can be used to create different type of incremental counters. Consider \texttt{SUM(1) OVER (ORDER BY SALARY)} as an example of what they can do, each of them in a different way. Following is an example query, also comparing with the \texttt{SUM} behavior.

\begin{verbatim}
select
  id, salary,
  dense_rank() over (order by salary),
  rank() over (order by salary),
  row_number() over (order by salary),
  sum(1) over (order by salary)
from employee
order by salary;
\end{verbatim}

The result set:

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>dense_rank</th>
<th>rank</th>
<th>row_number</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

The difference between \texttt{DENSE_RANK} and \texttt{RANK} is that there is a gap related to duplicate rows (relative to the window ordering) only in \texttt{RANK}. \texttt{DENSE_RANK} continues assigning sequential numbers after the duplicate salary. On the other hand, \texttt{ROW_NUMBER} always assigns sequential numbers, even when there are duplicate values.

\textbf{Navigational Functions}

The navigational functions get the simple (non-aggregated) value of an expression from another row of the query, within the same partition.

\textbf{Syntax}

\begin{verbatim}
< navigational window function > ::=  
  FIRST_VALUE(< expr >) |  
  LAST_VALUE(< expr >) |  
  NTH_VALUE(< expr >, < offset >) [ FROM FIRST | FROM LAST ] |  
  LAG(< expr > [ [, < offset > [, < default > ] ] ] ) |  
  LEAD(< expr > [ [, < offset > [, < default > ] ] ] )
\end{verbatim}

\textbf{Important to Note}

\texttt{FIRST_VALUE}, \texttt{LAST_VALUE} and \texttt{NTH_VALUE} also operate on a window frame. Currently, Firebird always frames from the first to the current row of the partition, not to the last. This is likely to produce strange results for \texttt{NTH_VALUE} and especially \texttt{LAST_VALUE}.
Example

```sql
select
   id,
   salary,
   first_value(salary) over (order by salary),
   last_value(salary) over (order by salary),
   nth_value(salary, 2) over (order by salary),
   lag(salary) over (order by salary),
   lead(salary) over (order by salary)
from employee
order by salary;
```

The result set:

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>first_value</th>
<th>last_value</th>
<th>nth_value</th>
<th>lag</th>
<th>lead</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>9.00</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>8.00</td>
<td>9.00</td>
<td>9.00</td>
<td>8.00</td>
<td>10.00</td>
</tr>
<tr>
<td>1</td>
<td>10.00</td>
<td>8.00</td>
<td>10.00</td>
<td>9.00</td>
<td>9.00</td>
<td>10.00</td>
</tr>
<tr>
<td>5</td>
<td>10.00</td>
<td>8.00</td>
<td>10.00</td>
<td>9.00</td>
<td>10.00</td>
<td>12.00</td>
</tr>
<tr>
<td>2</td>
<td>12.00</td>
<td>8.00</td>
<td>12.00</td>
<td>9.00</td>
<td>10.00</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

FIRST_VALUE and LAST_VALUE get, respectively, the first and last value of the ordered partition.

NTH_VALUE gets the n-th value, starting from the first (default) or the last record, from the ordered partition. An offset of 1 from first would be equivalent to FIRST_VALUE; an offset of 1 from last is equivalent to LAST_VALUE.

LAG looks for a preceding row, and LEAD for a following row. LAG and LEAD get their values within a distance respective to the current row and the offset (which defaults to 1) passed.

In a case where the offset points outside the partition, the default parameter (which defaults to NULL) is returned.

Advanced Plan Output

Dmitry Yemanov

PLAN output can now be output in a more structured and comprehensible form, e.g.

```sql
SELECT statement
   -> First [10]
   -> Sort [SUM, O_ORDERDATE]
   -> Aggregate
      -> Sort [L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY]
      -> Inner Loop Join
      -> Filter
         -> Table #ORDERS# Access By ID
         -> Bitmap
         -> Index #ORDERS_ORDERDATE# Range Scan
      -> Filter
         -> Table #CUSTOMER# Access By ID
         -> Bitmap
         -> Index #CUSTOMER_PK# Unique Scan
```
Advanced PLAN Output in isql

New syntax SET EXPLAIN [ON | OFF ] has been added to the isql utility to surface this option. For details, refer to SET EXPLAIN Extensions for Viewing Detailed Plans in the Utilities chapter.

Internal Functions

Additions and enhancements to the internal functions set are:

SUBSTRING with Regular Expressions
Adriano dos Santos Fernandes

A substring search can now use a regular expression.

Search Pattern

```
SUBSTRING(<string> [NOT] SIMILAR TO <pattern> ESCAPE <char>)
```

Discussion: Tracker CORE-2006

For more information about the use of SIMILAR TO expressions, refer to README.similar_to.txt in the / doc/ subdirectory of your Firebird installation.

Tip

The regex used is the SQL one. A guide is available in the DML chapter of the v.2.5 release notes and also at the Firebird web site.

Inverse Hyperbolic Trigonometric Functions
Claudio Valderrama C.

The six inverse hyperbolic trigonometric functions have been implemented internally. They are:

**ACOSH**

Returns the hyperbolic arc cosine of a number (expressed in radians). Format: `ACOSH( <number> )`

**ASINH**

Returns the hyperbolic arc sine of a number (expressed in radians). Format: `ASINH( <number> )`

**ATANH**

Returns the hyperbolic arc tangent of a number (expressed in radians). Format: `ATANH( <number> )`
**Data Manipulation Language (DML)**

**COSH**

Returns the hyperbolic cosine of an angle (expressed in radians). Format: \( \text{COSH}(\ <\text{number}\>\ ) \)

**SINH**

Returns the hyperbolic sine of an angle (expressed in radians). Format: \( \text{SINH}(\ <\text{number}\>\ ) \)

**TANH**

Returns the hyperbolic tangent of an angle (expressed in radians). Format: \( \text{TANH}(\ <\text{number}\>\ ) \)

**Statistical Functions**

Hajime Nakagami  
Adriano dos Santos Fernandes

A suite of SQL-standards-compliant statistical functions has been implemented.

**Aggregate Statistical Functions**


**Syntax**

\[
<\text{single param statistical function}> ::= <\text{single param statistical function name}>(<\text{expr}>)
\]

\[
<\text{single param statistical function name}> ::= \{ \text{VAR\_POP} | \text{VAR\_SAMP} | \text{STDDEV\_POP} | \text{STDDEV\_SAMP} \}
\]

\[
<\text{dual param statistical function}> ::= <\text{dual param statistical function name}>(<\text{expr}_1>, <\text{expr}_2>)
\]

\[
<\text{dual param statistical function name}> ::= \{ \text{COVAR\_POP} | \text{COVAR\_SAMP} | \text{CORR} \}
\]

**Semantics**

- NULL is returned from \( \text{VAR\_SAMP} \), \( \text{STDDEV\_SAMP} \) or \( \text{COVAR\_SAMP} \) if the result count is 0 or 1
- NULL is returned from \( \text{VAR\_POP} \), \( \text{STDDEV\_POP} \), \( \text{COVAR\_POP} \) or \( \text{CORR} \) if the result count is 0

**Syntax**

```
SELECT STDDEV\_SAMP(salary) FROM employees;
```

**SD/Variance Function Descriptions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{VAR_SAMP} )</td>
<td>( \text{VAR_SAMP}(\ &lt;\text{expr}&gt;\ ) )</td>
<td>Returns the Sample Variance, equivalent to ( (\text{SUM}(\text{expr}^2) - \text{SUM}(\text{expr})^2 / \text{COUNT}(\text{expr})) / (\text{COUNT}(\text{expr}) - 1) )</td>
</tr>
</tbody>
</table>

81
### Data Manipulation Language (DML)

#### Function Format Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>VAR_POP</code></td>
<td><code>VAR_POP(&lt;expr&gt;)</code></td>
<td>Returns the the Population Variance, equivalent to</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>(SUM(&lt;expr&gt; ^ 2) - SUM(&lt;expr&gt;) ^ 2 / COUNT(&lt;expr&gt;)) / COUNT(&lt;expr&gt;)</code></td>
</tr>
<tr>
<td><code>STDDEV_SAMP</code></td>
<td><code>STDDEV_SAMP(&lt;expr&gt;)</code></td>
<td>Returns the Sample Standard Deviation, equivalent to</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SQRT(VAR_SAMP(&lt;expr&gt;))</code></td>
</tr>
<tr>
<td><code>STDDEV_POP</code></td>
<td><code>STDDEV_POP(&lt;expr&gt;)</code></td>
<td>Returns the Population Standard Deviation, equivalent to</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SQRT(VAR_POP(&lt;expr&gt;))</code></td>
</tr>
<tr>
<td><code>COVAR_SAMP</code></td>
<td><code>COVAR_SAMP(&lt;expr1&gt;, &lt;expr2&gt;)</code></td>
<td>Returns the Sample Population, equivalent to</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>(SUM(&lt;expr1&gt; * &lt;expr2&gt;) - SUM(&lt;expr1&gt;) * SUM(&lt;expr2&gt;) / COUNT(*)) / (COUNT(*) - 1)</code></td>
</tr>
<tr>
<td><code>COVAR_POP</code></td>
<td><code>COVAR_POP(&lt;expr1&gt;, &lt;expr2&gt;)</code></td>
<td>Returns the Population Covariance, equivalent to</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>(SUM(&lt;expr1&gt; * &lt;expr2&gt;) - SUM(&lt;expr1&gt;) * SUM(&lt;expr2&gt;) / COUNT(*)) / COUNT(*)</code></td>
</tr>
<tr>
<td><code>CORR</code></td>
<td><code>CORR(&lt;expr1&gt;, &lt;expr2&gt;)</code></td>
<td>Returns the Coefficient of Correlation, equivalent to</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>COVAR_POP(&lt;expr1&gt;, &lt;expr2&gt;) / (STDDEV_POP(&lt;expr2&gt;) * STDDEV_POP(&lt;expr1&gt;))</code></td>
</tr>
</tbody>
</table>

#### Linear Regression Functions

The suite of REGR_* functions analyses the relationships between two sets of numeric data, considering only sets that are not NULL in either expression. See Tracker ticket `CORE-4722`.

**Syntax**

```
<regr function> ::= <function name>(<expr1>, <expr2>)
<function name> := { REGR_AVGX | REGR_AVGY | REGR_COUNT | REGR_INTERCEPT | REGR_R2 | REGR_SLOPE | REGR_SXX | REGR_SXY | REGR_SYY }
```

**Formulae**

The formulae use the following variables:
**Data Manipulation Language (DML)**

Y: `<expr1>` (<`expr1`> IS NOT NULL AND `expr2` IS NOT NULL)
X: `<expr2>` (<`expr1`> IS NOT NULL AND `expr2` IS NOT NULL)
N: COUNT of recordset unless `<expr1>` IS NULL OR `<expr2>` IS NULL

**Important**

Y and X are DOUBLE PRECISION. N is SMALLINT, INTEGER or BIGINT.

All functions eliminate expression pairs where either expression in the pair is NULL. If no rows remain, the functions (except REGR_COUNT()) return NULL.

### Linear Regression Function Descriptions

<table>
<thead>
<tr>
<th>Function</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>REGR_AVGX</code></td>
<td><code>REGR_AVGX(Y, X)</code></td>
<td>Returns the average of the independent expression (Y) in the expression pair. The return value is of type DOUBLE PRECISION.</td>
</tr>
<tr>
<td><strong>Formula:</strong></td>
<td><code>REGR_AVGX(Y, X) = SUM(X) / N</code></td>
<td></td>
</tr>
<tr>
<td><code>REGR_AVGY</code></td>
<td><code>REGR_AVGY(Y, X)</code></td>
<td>Returns the average of the dependent expression (X) in the expression pair. The return value is of type DOUBLE PRECISION.</td>
</tr>
<tr>
<td><strong>Formula:</strong></td>
<td><code>REGR_AVGY(Y, X) = SUM(Y) / N</code></td>
<td></td>
</tr>
<tr>
<td><code>REGR_COUNT</code></td>
<td><code>REGR_COUNT(Y, X)</code></td>
<td>Returns the number of expression pairs (Y and X). The return value is of type SMALLINT, INTEGER or BIGINT. If no rows remain after elimination of pairs where either expression is NULL, the function returns 0.</td>
</tr>
<tr>
<td><strong>Formula:</strong></td>
<td><code>REGR_COUNT(Y, X) = N</code></td>
<td></td>
</tr>
<tr>
<td><code>REGR_INTERCEPT</code></td>
<td><code>REGR_INTERCEPT(Y, X)</code></td>
<td>Returns the y-intercept of the regression line determined by a set of expression pairs (Y and X).</td>
</tr>
<tr>
<td><strong>Formula:</strong></td>
<td><code>REGR_INTERCEPT(Y, X) = REGR_AVGY(Y, X) - REGR_SLOPE(Y, X) * REGR_AVGX(Y, X)</code></td>
<td></td>
</tr>
<tr>
<td><code>REGR_R2</code></td>
<td><code>REGR_R2(Y, X)</code></td>
<td>Returns the square of the correlation coefficient of a set of expression pairs (Y and X).</td>
</tr>
<tr>
<td><strong>Formula:</strong></td>
<td><code>REGR_R2(Y, X) = POWER(CORR(Y, X),2)</code></td>
<td></td>
</tr>
<tr>
<td><code>REGR_SLOPE</code></td>
<td><code>REGR_SLOPE(Y, X)</code></td>
<td>Returns the slope of the regression line, determined by a set of expression pairs (Y and X).</td>
</tr>
</tbody>
</table>
### Data Manipulation Language (DML)

<table>
<thead>
<tr>
<th>Function</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REGR_SXX</strong></td>
<td><strong>REGR_SXX(Y, X)</strong></td>
<td>Returns the sum of squares of the independent expression (Y) in an expression pair (Y and X). Formula: ( \text{REGR}_\text{SXX}(Y, X) = N \times \text{VAR}_\text{POP}(X) )</td>
</tr>
<tr>
<td><strong>REGR_SXY</strong></td>
<td><strong>REGR_SXY(Y, X)</strong></td>
<td>Returns the sum of products of the independent expression multiplied by the dependent expression in an expression pair (Y and X). Formula: ( \text{REGR}_\text{SXY}(Y, X) = N \times \text{COVAR}_\text{POP}(Y, X) )</td>
</tr>
<tr>
<td><strong>REGR_SYY</strong></td>
<td><strong>REGR_SYY(Y, X)</strong></td>
<td>Returns the sum of squares of the dependent expression in an expression pair (Y and X). Formula: ( \text{REGR}_\text{SYY}(Y, X) = N \times \text{VAR}_\text{POP}(Y) )</td>
</tr>
</tbody>
</table>

---

**TRIM() BLOB Arguments Lose 32 KB limit**
Adriano dos Santos Fernandes

In prior versions, TRIM(substring from string) allowed BLOBs for both arguments, but the first argument had to be smaller than 32 KB. Now both arguments can take BLOBs of up to 4 GB.

**String Literal Limit Adjustments**
Adriano dos Santos Fernandes

The internal length of a string can, at some levels, be almost 64 KB. Tests demonstrated that it is safe to accept a string literal of up to that size for writing to a text BLOB. Accordingly,

1. The (32KB - 3) “safety limit” on literal string length for writing to text BLOBs has been raised to 65,533 bytes (64KB - 3);
2. A limit, in characters, is calculated in run-time for strings that are in multi-byte character sets, to avoid overrunning the bytes limit. For example, for a UTF8 string (max. 4 bytes/character) the run-time limit is likely to be about \((\text{floor} \left( \frac{65533}{4} \right)) = 16383\) characters.

See Tracker ticket **CORE-4881**.

**Enhancements to DATEADD() Internal Function**
Adriano dos Santos Fernandes
For the internal function DATEADD()

- The function now supports a fractional value for MILLISECOND. See Tracker item CORE-4457.
- the data type of input <amount> arguments has changed from INTEGER to BIGINT. See Tracker item CORE-4310.

### DML Improvements

A collection of useful DML improvements is released with Firebird 3.

### Alternatives for Embedding Quotes in String Literals

Adriano dos Santos Fernandes

It is now possible to use a character, or character pair, other than the doubled (escaped) apostrophe, to embed a quoted string inside another string. The keyword `q` or `Q` preceding a quoted string informs the parser that certain left-right pairs or pairs of identical characters within the string are the delimiters of the embedded string literal.

**Syntax**

```plaintext
<alternate string literal> ::= { q | Q } <quote> <alternate start char> [ { <char> }... ] <alternate end char> <quote>
```

**Rules**

When `<alternate start char>` is ‘(’, ‘{’, '[' or '<', `<alternate end char>` is paired up with its respective “partner”, viz. ‘)’, ‘}’, ‘]’ and ‘>’. In other cases, `<alternate end char>` is the same as `<alternate start char>`.

Inside the string, i.e., `<char>` items, single (not escaped) quotes could be used. Each quote will be part of the result string.

**Examples**

```sql
select q'{abc{def}ghi}' from rdb$database; -- result: abc{def}ghi
select q'!That's a string!' from rdb$database; -- result: That's a string
```

### SQL:2008-Compliant OFFSET and FETCH Clauses

Mark Rotteveel

New SQL:2008 compliant OFFSET and FETCH clauses provide a standard equivalent for FIRST and SKIP, and an alternative for ROWS...TO, when fetching sets from ordered output.

- The OFFSET clause specifies the number of rows to skip
- The FETCH clause specifies the number of rows to fetch.
As with SKIP and FIRST, OFFSET and FETCH clauses can be applied independently, in both top-level and nested query expressions. They are available in PSQL and DSQL.

**Syntax Pattern**

```sql
SELECT ... [ ORDER BY <expr_list> ]
[ OFFSET <simple_value_expr> { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ <simple_value_expr> ] { ROW | ROWS } ONLY ]
```

<simple_value_expr> is a (numeric) literal, a DSQL parameter (?) or a PSQL named parameter (:namedparameter) that resolves to an integer data type.

**Examples**

```sql
-- 1:
SELECT * FROM T1 ORDER BY COL1
  OFFSET 10 ROWS;
-- 2:
SELECT * FROM T1 ORDER BY COL1
  FETCH FIRST 10 ROWS ONLY;
-- 3:
SELECT * FROM ( 
  SELECT * FROM T1 ORDER BY COL1 DESC
    OFFSET 1 ROW
    FETCH NEXT 10 ROWS ONLY
  ) a 
  ORDER BY a.COL1
  FETCH FIRST ROW ONLY;
```

**Notes**

1. The FIRST/SKIP and ROWS clauses are non-standard alternatives.
2. The OFFSET and/or FETCH clauses cannot be mixed with clauses from the FIRST/SKIP or ROWS alternatives in the same query expression.
3. Expressions and column references are not allowed within either the OFFSET or the FETCH clause.
4. Unlike the ROWS clause, OFFSET and FETCH are available only in SELECT statements.
5. The “percentage FETCH” defined in the SQL standard is not supported.
6. “FETCH ... WITH TIES” defined in the SQL standard is not supported.

---

**Prohibit Edgy Mixing of Implicit/Explicit Joins**

Dmitry Yemanov

While mixing of implicit and explicit join syntaxes is not recommended at all, the parser would allows them, nevertheless. Certain “mixes” actually cause the optimizer to produce unexpected results, including “No record for fetch” errors. The same edgy styles are prohibited by other SQL engines and now they are prohibited in Firebird.

To visit some discussion on the subject, see the Tracker ticket [CORE-2812](http://Tracker-Link).
Support for Left-side Parameters in WHERE Clause
Adriano dos Santos Fernandes

The following style of subquery, with the parameter in the left side of a WHERE...IN (SELECT...) condition, would fail with the error “The data type of the parameter is unknown”.

This style is now accepted. For example:

```sql
SELECT <columns> FROM table_1 t1
WHERE <conditions on table_1>
AND (? IN (SELECT some_col FROM table_2 t2 WHERE t1.id = t2.ref_id))
```

Important
Better SQL coding practice would be to use EXISTS in these cases; however, developers were stumbling over this problem when using generated SQL from Hibernate, which used the undesirable style.

Enhancements to the RETURNING Clause
Adriano dos Santos Fernandes

Two enhancements were added to the RETURNING clause:

RETURNING Clause Value Can be Aliased

When using the RETURNING clause to return a value to the client, the value can now be passed under an alias.

Example Without and With Aliases

```sql
UPDATE T1 SET F2 = F2 * 10
RETURNING OLD.F2, NEW.F2; -- without aliases

UPDATE T1 SET F2 = F2 * 10
RETURNING OLD.F2 OLD_F2, NEW.F2 AS NEW_F2; -- with aliases
```

Note
The keyword AS is optional.

RETURNING Clause from Positioned Updates and Deletes

Support has been added for a RETURNING clause in positioned (WHERE CURRENT OF) UPDATE and DELETE statements.

Example
UPDATE T1 SET F2 = F2 * 10 WHERE CURRENT OF C
RETURNING NEW.F2;

**Cursor Stability**
Vlad Khorsun

Until this release, Firebird suffered from an infamous bug whereby a data modification operation could loop infinitely and, depending on the operation, delete all the rows in a table, continue updating the same rows ad infinitum or insert rows until the host machine ran out of resources. All DML statements were affected (INSERT, UPDATE, DELETE, MERGE). It occurred because the engine used an implicit cursor for the operations.

To ensure stability, rows to be inserted, updated or deleted had to be marked in some way in order to avoid multiple visits. Another workaround was to force the query to have a SORT in its plan, in order to materialize the cursor.

From Firebird 3, engine uses the Undo log to check whether a row was already inserted or modified by the current cursor.

**Important**
This stabilisation does NOT work with SUSPEND loops in PSQL.

**An Improvement for GTTs**
Vlad Khorsun

Global temporary tables (GTTs) are now writable even in read-only transactions. The effect is as follows.-

*Read-only transaction in read-write database*
- Writable in both ON COMMIT PRESERVE ROWS and ON COMMIT DELETE ROWS

*Read-only transaction in read-only database*
- Writable in ON COMMIT DELETE ROWS only

Also

- Rollback for GTT ON COMMIT DELETE ROWS is faster
- Rows do not need to be backed out on rollback
- Garbage collection in GTT is not delayed by active transactions of other connections

**Note**
The same refinements were also backported to Firebird 2.5.1.

**An Improvement for DML Strings**
Adriano dos Santos Fernandes
Strings in DML queries are now transformed or validated to avoid passing malformed strings to the engine internals, for example, to the MON$STATEMENTS.MONSQL_TEXT column.

The solution adopted depends on the character set of the attachment.-

• NONE—non-ASCII characters are transformed to question marks

• Others—the string is checked for malformed characters

**COUNT() Now Returns BIGINT**

The COUNT() aggregator now returns its result as BIGINT instead of INTEGER.

**Optimizations**

Optimizations made for this release included:

**SIMILAR TO**
Adriano dos Santos Fernandes

The performance of SIMILAR TO was improved.

**OR'ed Parameter in WHERE Clause**
Dmitry Yemanov

Performance for `(table.field = :param or :param = -1)` in the WHERE clause was enhanced.

**Better Choices for Navigation**
Dmitry Yemanov

Previously, when an ORDER plan was in a SELECT structure, the optimizer would choose the first index candidate that matched the ORDER BY or GROUP BY clause. This “first come” approach is not the best when multiple index choices are available. The Firebird 3 engine surveys all of the available choices and picks the most suitable index.

See Tracker ticket [CORE-4285](#).

**Plainer Execution Path for UNION Queries**
Dmitry Yemanov

Previously, the execution path for UNION queries was hierarchical, often causing redundant reads. This optimization replaces the hierarchical execution path with a plainer one that improves performance.

See Tracker ticket [CORE-4165](#).

**Index Walk for Compound Index**
Dmitry Yemanov
The optimizer now allows an index walk (ORDER plan) when a suitable compound index (A, B) is available for a query condition of the style \texttt{WHERE A = ? ORDER BY B}.

See Tracker ticket \texttt{CORE-1846}.

\textbf{Performance Improvement for SET STATISTICS INDEX}

Vlad Khorsun

BTR\_selectivity() would walk the whole leaf level of given index b-tree to calculate index selectivity. Throughout the process, the only rescheduling would happen at a disk I/O operation. The effect was to impose long waits for AST requests from concurrent attachments, such as page lock requests, monitoring, cancellation, etc. An improvement in Firebird 3 seems to solve that problem.

See Tracker ticket \texttt{CORE-1846}.

\textbf{Dialect 1 Interface}

Adriano dos Santos Fernandes

Selection of SQL\_INT64, SQL\_DATE and SQL\_TIME in dialect 1 was enabled.

See Tracker \texttt{CORE-3972}.

\textbf{Embedded SQL (ESQL) Enhancements}

Dmitry Yemanov

Two enhancements were included in the Embedded SQL subset in this release:

\textit{Support for UPDATE OR INSERT statement}

See Tracker ticket \texttt{CORE-4438}

\textit{Support for the RETURNING clause}

See Tracker ticket \texttt{CORE-4437}
Advancements in procedural SQL (PSQL) include:

Quick Links

- PSQL Stored Functions
- PSQL Subroutines
- Packages
- DDL Triggers
- Exceptions with Parameters
- CONTINUE in Looping Logic
- PSQL Cursor Stabilization
- PSQL Cursors as Variables
- SQLSTATE Now Valid in Exception Trap
- Some Size Limits Removed Using New API

PSQL Stored Functions

Dmitry Yemanov

It is now possible to write a scalar function in PSQL and call it just like an internal function.

Syntax for the DDL

```
{CREATE [OR ALTER] | ALTER | RECREATE} FUNCTION <name>
[(param1 [, ...])]
RETURNS <type>
AS
BEGIN
    ...
END
```

Tip

The CREATE statement is the *declaration syntax* for PSQL functions, parallel to DECLARE for legacy UDFs.

Example

```
CREATE FUNCTION F(X INT) RETURNS INT
AS
BEGIN
    RETURN X+1;
END;
```
SELECT F(5) FROM RDB$DATABASE;

---

**PSQL Sub-routines**

Adriano dos Santos Fernandes

The header of a PSQL module (stored procedure, stored function, trigger, executable block) can now accept sub-procedure and sub-function blocks in the header declarations for use within the body of the module.

**Syntax for Declaring a Sub-procedure**

```sql
DECLARE PROCEDURE <name> [(param1 [, ...])]
[RETURNS (param1 [, ...])] AS
...
```

**Syntax for declaring a Sub-function**

```sql
DECLARE FUNCTION <name> [(param1 [, ...])]
RETURNS <type> AS
...
```

**Examples**

```sql
SET TERM ^;
--
-- Sub-function in EXECUTE BLOCK
-- EXECUTE BLOCK RETURNS (N INT)
AS
  DECLARE FUNCTION F(X INT) RETURNS INT AS
  BEGIN
    RETURN X+1;
  END BEGIN
N = F(5);
SUSPEND;
END ^
--
-- Sub-function inside a stored function
-- CREATE OR ALTER FUNCTION FUNC1 (n1 INTEGER, n2 INTEGER)
  RETURNS INTEGER AS
  DECLARE FUNCTION SUBFUNC (n1 INTEGER, n2 INTEGER) RETURNS INTEGER AS
  BEGIN
    RETURN n1 + n2;
  END BEGIN
  RETURN SUBFUNC(n1, n2);
```
A package is a group of procedures and functions managed as one entity. The notion of “packaging” the code components of a database operation addresses several objectives:

**Modularisation**

The idea is to separate blocks of interdependent code into logical modules, as programming languages do.

In programming it is well recognised that grouping code in various ways, in namespaces, units or classes, for example, is a good thing. With standard procedures and functions in the database this is not possible. Although they can be grouped in different script files, two problems remain:

1. The grouping is not represented in the database metadata.
2. Scripted routines all participate in a flat namespace and are callable by everyone (we are not referring to security permissions here).

**To facilitate dependency tracking**

We want a mechanism to facilitate dependency tracking between a collection of related internal routines, as well as between this collection and other routines, both packaged and unpackaged.

Firebird packages come in two parts: a *header* (keyword PACKAGE) and a *body* (keyword PACKAGE BODY). This division is very similar to a Delphi unit, the header corresponding to the interface part and the body corresponding to the implementation part.

The header is created first (CREATE PACKAGE) and the body (CREATE PACKAGE BODY) follows.

Whenever a packaged routine determines that it uses a certain database object, a dependency on that object is registered in Firebird system tables. Thereafter, to drop, or maybe alter that object, you first need to remove what depends on it. As it is a package body that depends on it, that package body can just be dropped, even if some other database object depends on this package. When the body is dropped, the header remains, allowing you to recreate its body once the changes related to the removed object are done.

**To facilitate permission management**

It is good practice in general to create routines to require privileged use and to use roles or users to enable the privileged use. As Firebird runs routines with the caller privileges, it is necessary also to grant resource usage to each routine when these resources would not be directly accessible to the caller. Usage of each routine needs to be granted to users and/or roles.

Packaged routines do not have individual privileges. The privileges act on the package. Privileges granted to packages are valid for all package body routines, including private ones, but are stored for the package header.
For example:

GRANT SELECT ON TABLE secret TO PACKAGE pk_secret;
GRANT EXECUTE ON PACKAGE pk_secret TO ROLE role_secret;

To enable “private scope”
This objective was to introduce private scope to routines; that is, to make them available only for internal usage within the defining package.

All programming languages have the notion of routine scope, which is not possible without some form of grouping. Firebird packages also work like Delphi units in this regard. If a routine is not declared in the package header (interface) and is implemented in the body (implementation), it becomes a private routine. A private routine can only be called from inside its package.

Signatures
For each routine that is assigned to a package, elements of a digital signature (the set of [routine name, parameters and return type]) are stored in the system tables.

The signature of a procedure or routine can be queried, as follows:

SELECT...
-- sample query to come

Packaging Syntax

```
<package_header> ::=  
  { CREATE [OR ALTER] | ALTER | RECREATE } PACKAGE <name>
  AS  
  BEGIN  
    [ <package_item> ... ]  
  END

<package_item> ::=  
  <function_decl> ; |  
  <procedure_decl> ;

<function_decl> ::=  
  FUNCTION <name> [( <parameters> )] RETURNS <type>

<procedure_decl> ::=  
  PROCEDURE <name> [( <parameters> ) [RETURNS ( <parameters> )]]

<package_body> ::=  
  { CREATE | RECREATE } PACKAGE BODY <name>
  AS  
  BEGIN  
    [ <package_item> ... ]  
    [ <package_body_item> ... ] 
  END
```
<package_body_item> ::= 
  <function_impl> | 
  <procedure_impl>

<function_impl> ::= 
  FUNCTION <name> [(<parameters>)] RETURNS <type> 
  AS 
  BEGIN 
  ... 
  END |
  FUNCTION <name> [(<parameters>)] RETURNS <type> 
  EXTERNAL NAME '<name>' ENGINE <engine>

<precedure_impl> ::= 
  PROCEDURE <name> [(<parameters>)] [RETURNS (<parameters>)] 
  AS 
  BEGIN 
  ... 
  END |
  PROCEDURE <name> [(<parameters>)] [RETURNS (<parameters>)] 
  EXTERNAL NAME '<name>' ENGINE <engine>

<drop_package_header> ::= 
  DROP PACKAGE <name>

<drop_package_body> ::= 
  DROP PACKAGE BODY <name>

Syntax rules

- All routines declared in the header and at the start of the body should be implemented in the body with the same signature, i.e., you cannot declare the routine in different ways in the header and in the body.

- Default values for procedure parameters cannot be redefined in <package_item> and <package_body_item>. They can be in <package_body_item> only for private procedures that are not declared.

Notes

- DROP PACKAGE drops the package body before dropping its header.

- The source of package bodies is retained after ALTER/RECREATE PACKAGE. The column RDB$PACKAGES.RDB$VALID_BODY_FLAG indicates the state of the package body. See Tracker item CORE-4487.

- UDF declarations (DECLARE EXTERNAL FUNCTION) are currently not supported inside packages.

- Syntax is available for a description (COMMENT ON) for package procedures and functions and their parameters. See Tracker item CORE-4484.

Simple Packaging Example

SET TERM ^;
-- package header, declarations only
CREATE OR ALTER PACKAGE TEST
AS
BEGIN
  PROCEDURE P1(I INT) RETURNS (O INT); -- public procedure
END

-- package body, implementation
RECREATE PACKAGE BODY TEST
AS
BEGIN
  FUNCTION F1(I INT) RETURNS INT; -- private function
  PROCEDURE P1(I INT) RETURNS (O INT)
  AS
  BEGIN
  END
  FUNCTION F1(I INT) RETURNS INT
  AS
  BEGIN
    RETURN 0;
  END
END ^

Note
More examples can be found in the Firebird installation, in ./examples/package/.

DDL triggers

A. dos Santos Fernandes

Acknowledgement
This feature was sponsored with donations gathered at the fifth Brazilian Firebird Developers' Day, 2008

The purpose of a “DDL trigger” is to enable restrictions to be placed on users who attempt to create, alter or drop a DDL object.

Syntax Pattern

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

<ddl event> ::= ANY DDL STATEMENT
  | <ddl event item> {{OR <ddl event item>}}...

<ddl event item> ::= CREATE TABLE
Semantics

1. BEFORE triggers are fired before changes to the system tables. AFTER triggers are fired after system table changes.

**Important Rule**
The event type [BEFORE | AFTER] of a DDL trigger cannot be changed.

2. When a DDL statement fires a trigger that raises an exception (BEFORE or AFTER, intentionally or unintentionally) the statement will not be committed. That is, exceptions can be used to ensure that a DDL operation will fail if the conditions are not precisely as intended.

3. DDL trigger actions are executed only when committing the transaction in which the affected DDL command runs. Never overlook the fact that what is possible to do in an AFTER trigger is exactly what is possible to do after a DDL command without autocommit. You cannot, for example, create a table in the trigger and use it there.
4. With “CREATE OR ALTER” statements, a trigger is fired one time at the CREATE event or the ALTER event, according to the previous existence of the object. With RECREATE statements, a trigger is fired for the DROP event if the object exists, and for the CREATE event.

5. ALTER and DROP events are generally not fired when the object name does not exist. For the exception, see point 6.

6. The exception to rule 5 is that BEFORE ALTER/DROP USER triggers fire even when the user name does not exist. This is because, underneath, these commands perform DML on the security database and the verification is not done before the command on it is run. This is likely to be different with embedded users, so do not write code that depends on this.

7. If some exception is raised after the DDL command starts its execution and before AFTER triggers are fired, AFTER triggers will not be fired.

8. Packaged procedures and triggers do not fire individual {CREATE | ALTER | DROP} {PROCEDURE | FUNCTION} triggers.

**Permissions**

The following users can create, alter or drop DDL triggers and access the trigger-related switches in the Firebird utilities:

- the database owner
- SYSDBA
- a user logged in under the RDB$ADMIN role
- a user having the ALTER DATABASE metadata privilege

**Support in Utilities**

A DDL trigger is a type of database trigger, so the parameters -nodbtriggers (GBAK and ISQL) and -T (NBACKUP) apply to them. Remember that only users with the appropriate metadata privileges can use these switches.

**DDL_TRIGGER Context Namespace**

The introduction of DDL triggers brings with it the new DDL_TRIGGER namespace for use with RDB $GET_CONTEXT. Its usage is valid only when a DDL trigger is running. Its use is valid in stored procedures and functions called by DDL triggers.

The DDL_TRIGGER context works like a stack. Before a DDL trigger is fired, the values relative to the executed command are pushed onto this stack. After the trigger finishes, the values are popped. So in the case of cascade DDL statements, when an user DDL command fires a DDL trigger and this trigger executes another DDL command with EXECUTE STATEMENT, the values of the DDL_TRIGGER namespace are the ones relative to the command that fired the last DDL trigger on the call stack.
**Elements of DDL_TRIGGER Context**

- **EVENT_TYPE**: event type (CREATE, ALTER, DROP)
- **OBJECT_TYPE**: object type (TABLE, VIEW, etc)
- **DDL_EVENT**: event name (<ddl event item>), where <ddl_event_item> is EVENT_TYPE || ' ' || OBJECT_TYPE
- **OBJECT_NAME**: metadata object name
- **OLD_OBJECT_NAME**: for tracking the renaming of a domain (see note)
- **NEW_OBJECT_NAME**: for tracking the renaming of a domain (see note)
- **SQL_TEXT**: sql statement text

**Note**

ALTER DOMAIN <old name> TO <new name> sets OLD_OBJECT_NAME and NEW_OBJECT_NAME in both BEFORE and AFTER triggers. For this command, OBJECT_NAME will have the old object name in BEFORE triggers and the new object name in AFTER triggers.

**Examples Using DDL Triggers**

Here is how you might use a DDL trigger to enforce a consistent naming scheme, in this case, stored procedure names should begin with the prefix “SP_”:

```sql
set auto on;
create exception e_invalid_sp_name 'Invalid SP name (should start with SP_)';
set term !;
create trigger trig_ddl_sp before CREATE PROCEDURE
as
begin
  if (rdb$get_context('DDL_TRIGGER', 'OBJECT_NAME') not starting 'SP_') then
    exception e_invalid_sp_name;
  end!
-- Test
create procedure sp_test
as
begin
end!
create procedure test
as
begin
end!
-- The last command raises this exception and procedure TEST is not created
-- Statement failed, SQLSTATE = 42000
-- exception 1
-- -E_INVALID_SP_NAME
```
Implement custom DDL security, in this case restricting the running of DDL commands to certain users:

create exception e_access_denied 'Access denied';
set term !;
create trigger trig_ddl before any ddl statement as begin
  if (current_user <> 'SUPER_USER') then exception e_access_denied;
end!
-- Test
create procedure sp_test as begin end!
-- The last command raises this exception and procedure SP_TEST is not created
-- Statement failed, SQLSTATE = 42000
-- exception 1
-- -E_ACCESS_DENIED
-- -Access denied
-- -At trigger 'TRIG_DDL' line: 4, col: 5
set term ;!

Use a trigger to log DDL actions and attempts:

create sequence ddl_seq;
create table ddl_log ( id bigint not null primary key, moment timestamp not null, user_name varchar(31) not null, event_type varchar(25) not null, object_type varchar(25) not null, ddl_event varchar(25) not null, object_name varchar(31) not null, sql_text blob sub_type text not null, ok char(1) not null );
set term !;
create trigger trig_ddl_log_before before any ddl statement as declare id type of column ddl_log.id; begin
  -- We do the changes in an AUTONOMOUS TRANSACTION, so if an exception happens
  -- and the command didn't run, the log will survive.
in autonomous transaction do
begin
    insert into ddl_log (id, moment, user_name, event_type, object_type,
                        ddl_event, object_name, sql_text, ok)
    values (next value for ddl_seq, current_timestamp, current_user,
            rdb$get_context('DDL_TRIGGER', 'EVENT_TYPE'),
            rdb$get_context('DDL_TRIGGER', 'OBJECT_TYPE'),
            rdb$get_context('DDL_TRIGGER', 'DDL_EVENT'),
            rdb$get_context('DDL_TRIGGER', 'OBJECT_NAME'),
            rdb$get_context('DDL_TRIGGER', 'SQL_TEXT'),
            'N')
    returning id into id;
    rdb$set_context('USER_SESSION', 'trig_ddl_log_id', id);
end
end!

-- Note: the above trigger will fire for this DDL command. It's good idea to
-- use -nodbtriggers when working with them!
create trigger trig_ddl_log_after after any ddl statement
as
begin
    -- Here we need an AUTONOMOUS TRANSACTION because the original transaction
    -- will not see the record inserted on the BEFORE trigger autonomous
    -- transaction if user transaction is not READ COMMITTED.
    in autonomous transaction do
        update ddl_log set ok = 'Y'
        where id = rdb$get_context('USER_SESSION', 'trig_ddl_log_id');
    end!
    commit!
    set term ;!

-- Delete the record about trig_ddl_log_after creation.
delete from ddl_log;
commit;

-- Test

-- This will be logged one time
-- (as T1 did not exist, RECREATE acts as CREATE) with OK = Y.
recreate table t1 {
    n1 integer,
    n2 integer
};

-- This will fail as T1 already exists, so OK will be N.
create table t1 {
    n1 integer,
    n2 integer
};

-- T2 does not exist. There will be no log.
drop table t2;

-- This will be logged twice
-- (as T1 exists, RECREATE acts as DROP and CREATE) with OK = Y.
recreate table t1 {
    n integer
};
commit;

select id, ddl_event, object_name, sql_text, ok
from ddl_log order by id;

<table>
<thead>
<tr>
<th>ID</th>
<th>DDL_EVENT</th>
<th>OBJECT_NAME</th>
<th>SQL_TEXT</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CREATE TABLE</td>
<td>T1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CREATE TABLE</td>
<td>T1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DROP TABLE</td>
<td>T1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CREATE TABLE</td>
<td>T1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQL_TEXT:
recreate table t1 (n1 integer, n2 integer)

SQL_TEXT:
create table t1 (n1 integer, n2 integer)

SQL_TEXT:
recreate table t1 (n integer)

SQL_TEXT:
recreate table t1 (n integer)

--- Scrollable (Bi-directional) Cursor Support ---

Dmitry Yemanov

Instead of just fetching rows sequentially in a forward direction, “scrollability” allows flexible navigation through an open cursor set both backwards and forwards. Rows next to, prior to and relative to the current cursor row can be targeted. In PSQL, a scrollable cursor can be operated on directly. API support is available to enable DSQL applications to fetch rows in a similar manner.

--- Cursor Syntax for PSQL ---

To declare a cursor:

DECLARE <name> SCROLL CURSOR FOR ( <select expression> )
To fetch forward:

```
FETCH <cursor name> [INTO <var name> [, <var name> ...]];
```

To fetch in any direction:

```
FETCH {NEXT | PRIOR | FIRST | LAST | ABSOLUTE <n> | RELATIVE <n>}
_FROM <cursor name> [INTO <var name> [, <var name> ...]];
```

See also Scrollable Cursor Support for DSQL in the chapter entitled [Changes to the Firebird API and ODS]. The section entitled Scrollable Cursor Usage explains a little more about the usage of the various FETCH options.

### Notes

1. When a scrolling option is omitted, NO SCROLL is implied (i.e., the cursor is opened as forward-only). This means that only FETCH [NEXT FROM] commands can be used. Other commands will return an error.

2. Scrollable cursors are internally materialized as a temporary record set, thus consuming memory/disk resources, so this feature should be used only when really necessary.

### Exceptions with parameters

Adriano dos Santos Fernandes

An exception can now be defined with a message containing slots for parameters which are filled and passed when raising the exception, using the syntax pattern

```
EXCEPTION <name> USING ( <value list> )
```

#### Examples

```
create exception e_invalid_val 'Invalid value @1 for the field @2';

...
if (val < 1000) then
  thing = val;
else
  exception e_invalid_val using (val, 'thing');
end

CREATE EXCEPTION EX_BAD_SP_NAME
'label of procedures must start with ''@1'' : ''@2'''

CREATE TRIGGER TRG_SP_CREATE BEFORE CREATE PROCEDURE
AS
```
DECLARE SP_NAME VARCHAR(255);
BEGIN
    SP_NAME = RDB$GET_CONTEXT('DDL_TRIGGER', 'OBJECT_NAME');
    IF (SP_NAME NOT STARTING 'SP_')
        THEN EXCEPTION EX_BAD_SP_NAME USING ('SP_', SP_NAME);
END;

Notes
The status vector is generated using this code combination: isc_except, <exception number>,
isc_formatted_exception, <formatted exception message>, <exception parameters>
Since a new error code (isc_formatted_exception) is used, the client must be v.3.0, or at least use the fire-
bird.msg file from v.3.0, in order to translate the status vector to a string.
Considering, in left-to-right order, each parameter passed in the exception-raising statement as “the Nth”, with
N starting at 1:
• If an Nth parameter is not passed, the text is not substituted.
• If NULL is passed, it is replaced by the string ‘*** null ***’.
• If more parameters are passed than are defined in the exception message, the surplus ones are ignored.
• The total length of the message, including the values of the parameters, is still limited to 1053 bytes.

CONTINUE in Looping Logic
Adriano dos Santos Fernandes
CONTINUE is a complementary command to BREAK/LEAVE, allowing flow of control to break (leave) and
start of the next iteration of a FOR/WHILE loop.
Syntax

    CONTINUE [<label>];

Example

FOR SELECT A, D FROM ATABLE INTO :achar, :ddate
    DO BEGIN
        IF (ddate < current_data - 30) THEN
            CONTINUE;
        ELSE
            /* do stuff */
            ...
    END

PSQL Cursor Stabilization
Vlad Khorsun
PSQL cursors without SUSPEND inside are now stable:

FOR SELECT ID FROM T WHERE VAL IS NULL INTO :ID
    DO BEGIN
        UPDATE T SET VAL = 1
        WHERE ID = :ID;
    END

Previously, this block would loop interminably. Now, the loop will not select the value if it was set within the loop.

Note
This could change the behaviour of legacy code.

If there is a SUSPEND inside the block, the old instability remains: this query, for example, still produces the infinite loop:

FOR SELECT ID FROM T INTO :ID
    DO BEGIN
        INSERT INTO T (ID) VALUES (:ID);
        SUSPEND;
    END

Extension of Colon Prefix Usage
Adriano dos Santos Fernandes

Hitherto, the colon (:) prefix has been used in PSQL to mark a reference to a variable in DML statements. Its use has been extended in Firebird 3 for two unrelated purposes:

1. to allow OLD/NEW fields in cursors to be read or assigned to and to assign them to variables.
2. to make variable assignment in both DML and PSQL statements in modules and blocks more flexible and, where needed, to resolve ambiguity between field names and variable names

PSQL Cursors as Variables

Referencing cursors as record variables is now supported in PSQL. Either explicit (DECLARE AS CURSOR) or implicit (FOR SELECT) PSQL cursors make their current record available via the cursor name, thus making the INTO clause optional.

In FOR SELECT loops, it requires the AS CURSOR clause to be specified. For example:

execute block as
begin
    for
        select id, x from t1 as cursor c1
    do begin

for select id, x from t2 where x = :cl.x as cursor c2 do
begin
    /* . . . */
end
end
end

Note
Notice the extension of the use of a colon (:) as a prefix to the referenced cursor field.

Another example

for
    select rdb$relation_id as id, rdb$relation_name as name
    from rdb$relations
    where rdb$view_blr is null
    as cursor tables
do begin
    out_id = tables.id;
    out_name = tables.name;
    suspend;
end

To avoid ambiguity, the colon prefix could be used:

    out_id = :tables.id;
    out_name = :tables.name;
/* or */
    :out_id = :tables.id;
    :out_name = :tables.name;

“tables” here is a cursor name and acts similarly to OLD/NEW in triggers.

Colon Prefix as a Variable Marker

It is now valid to apply the colon prefix to a variable on either side of an assignment statement in PSQL-only constructs.

Previously, these were valid assignments:

    var1 = :var2;
/* or */
    new.fld = :var;

whereas, these were invalid:

    :var1 = :var2;
/* or */
    :new.fld = :var;
The extension (CORE-4434) fixes this inconsistency.

This is now valid syntax:

```sql
create trigger t1 before insert on t1
as
  declare v integer;
begin
  :v = :old.n;
  :new.n = :v;
end
```

In fact, using this example, there is no difference between

```sql
:v = :old.n;
/* and */
  v = :old.n;
```

Here, it is just “syntactic sugar” but, in other cases, it provides the means to resolve ambiguity between field name references and variable names.

**Example of possible ambiguity**

```sql
for
  select rdb$relation_id as id, rdb$relation_name as name
  from rdb$relations
  where rdb$view_blr is null
  as cursor tables
do begin
  out_id = :table.id;
  select tables.name from tables where tables.id = :tables.id into :out_name;
  suspend;
end
```

Inside the nested SELECT, “tables” is both a table name and a cursor name here, so the colon is used to resolve the ambiguity.

---

**SQLSTATE in Exception Handlers**

Dmitry Yemanov

An SQLSTATE code becomes a valid condition for trapping an exception with a WHEN statement.

---

**PSQL Body Size Limit Raised**

If and only if the new API is being used, the size of the body of a stored procedure or a trigger can exceed the traditional limit of 32 KB. Currently, as a security measure, a hard-coded limit of 10MB is imposed, although the
theoretical limit is 4GB. The traditional size of all input or output parameters no longer limited to the traditional size of (64KB minus overhead), either.

For details, see Some SQL Size Limits Removed Using New API in the API chapter.
Chapter 11

Monitoring & Command-line Utilities

No new monitoring features or other utilities are released with Firebird 3.0. Existing features have undergone a few improvements.

Monitoring

Dmitry Yemanov

Several changes have been made to the set of virtual tables storing the monitoring information. These are listed in the System Tables section of Chapter 4, Changes to the Firebird API and ODS.

Remote Client Recognition

From this version forward, the port number of the remote client address for TCPv4 and TCPv6 protocols is included in MON$ATTACHMENTS.MON$REMOTE_ADDRESS. A new column in that table also reports the host name of the remote client. For details, see Changes to Client Address Reporting in the same chapter.

Tracing

Latest improvements to the Trace functions include:

- Trace output now supports showing the explained plan. See Tracker CORE-4451 (V. Khorsun).
- Tracing execution of stored functions. See Tracker CORE-4345 (V. Khorsun).
- Trace no longer ignores the fact that users from different security databases are actually different users. It now “knows” in which security database a particular SYSDBA was authenticated or where a privileged user's elevated privileges are established. It is no longer possible, for example, for a user named SYSDBA to access security3.fdb if it is not currently authenticated for that access. See See Tracker CORE-4851 (A. Peshkov).

gbak

New “Skip Data” Backup Option
Alex Peshkov

gbak has a new option switch-- -skip_d(ata) to ignore the data from specific tables during a backup.
The switch -skip_d(ata) accepts a regular expression as its argument. For example, to skip two tables in the employee database (aliased here as 'employee'):

```
gbak -skip_d '(sales|customer)' employee e1.fbk
```

**Tip**
The regex used is the SQL one—the same one that is used for Firebird's SIMILAR TO searches. A guide is available in the DML chapter of the v.2.5 release notes and also at the Firebird web site.

---

**Long Names for Log Files**
Alex Peshkov

This improvement allows the *gbak* log to take an extra-long name without encountering the message “Attempt to store 256 bytes in a clumplet”.

---

**Run-time Statistics in Verbose Output**
Vlad Khorsun

*gbak* can now show some runtime statistics in its verbose output. A new command-line switch `-STATISTICS` has been added to specify which statistics items should be produced. Currently, four items are implemented:

**Table 11.1. Arguments for *gbak* STATISTICS Output**

<table>
<thead>
<tr>
<th>Item</th>
<th>Argument</th>
<th>Data Reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time</td>
<td>T</td>
<td>Time elapsed since the <em>gbak</em> process started, in seconds and milliseconds</td>
</tr>
<tr>
<td>Time delta</td>
<td>D</td>
<td>Time elapsed since the previous line of output, in seconds and milliseconds</td>
</tr>
<tr>
<td>Page reads</td>
<td>R</td>
<td>Number of page reads since the previous line of output, integer</td>
</tr>
<tr>
<td>Page writes</td>
<td>W</td>
<td>Number of page writes since the previous line of output, integer</td>
</tr>
</tbody>
</table>

At least one item is mandatory for the STATISTICS switch. The arguments are case-insensitive and they can be in any order. For example, “TDRW” and “WdrT” are equivalent.

The STATISTICS switch will have no effect if the `-v[er]` switch is not specified.

Verbose output with STATISTICS includes two special lines:

- a line with headers for the specified statistics, printed before the other statistics lines:

  `gbak: time delta reads writes`
Monitoring & Command-line Utilities

• a line with total statistics summaries for the specified items, printed after the end of the main process:

\[ \text{gbak: 46.684 0.002 171 82442 total statistics} \]

**Note**

The feature is fully supported in the Services API with a new item in the SPB (Services Parameter Block). The \texttt{fbsvcmgr} utility also supports the SPB implementation.

**Examples**

1. Show total time since \texttt{gbak} start:

\[ \text{gbak \textasciitilde v \textasciitilde STATISTICS T \textasciitilde b employee emp_bkp.fbk \textasciitilde y log21.log} \]

2. Show delta time and page reads:

\[ ... \textasciitilde STAT DR ... \]

3. Show all statistics items:

\[ ... \textasciitilde sta TDRW ... \]

4. Sample of \texttt{gbak} verbose output:

\[ \text{firebird>gbak \textasciitilde v \textasciitilde stat tdrw \textasciitilde r a.fbk a.fdb} \]
\[ \text{gbak: opened file a.fbk} \]
\[ \text{gbak: time delta reads writes} \]
\[ \text{gbak: 0.173 0.173 0 0 transportable backup -- data in XDR format} \]
\[ \text{gbak: 0.175 0.002 0 0 backup file is compressed} \]
\[ \text{gbak: 0.177 0.001 0 0 backup version is 10} \]
\[ \text{gbak: 0.270 0.092 0 650 created database a.fdb, page_size 8192 bytes} \]
\[ \text{gbak: 0.273 0.002 0 2 started transaction} \]
\[ ... \]
\[ \text{gbak: 18.661 0.002 0 0 restoring data for table TEST1} \]
\[ \text{gbak: 18.698 0.036 0 0 10000 records restored} \]
\[ ... \]
\[ \text{gbak: 25.177 0.036 0 1770000 records restored} \]
\[ \text{gbak: 25.220 0.042 0 1633 1780000 records restored} \]
\[ ... \]
\[ \text{gbak: 38.702 0.002 0 0 restoring privilege for user SYSDBA} \]
\[ \text{gbak: 38.707 0.004 22 0 creating indexes} \]
\[ \text{gbak: 45.015 6.308 82 38394 activating and creating deferred index T2_VAL} \]
\[ ... \]
\[ \text{gbak: 46.682 0.008 4 13 finishing, closing, and going home} \]
\[ \text{gbak: 46.684 0.002 171 82442 total statistics} \]
\[ \text{gbak: adjusting the ONLINE and FORCED WRITES flags} \]

**See also:** Tracker ticket \texttt{CORE-1999}
**gsec**

The `gsec` utility is deprecated from Firebird 3 forward. This means you are encouraged to use the new SQL features for managing access described in Chapter 7, Security, in preference to existing equivalents provided by `gsec`.

**Important**

`gsec` will continue to work with `security3.fdb`. However, it does not work with alternative security databases.

---

**isql**

**SET EXPLAIN Extensions for Viewing Detailed Plans**

Dmitry Yemanov

A new SET option is added: SET EXPLAIN [ON | OFF]. It extends the SET PLAN option to report the explained plan instead of the standard one.

If SET PLAN is omitted, then SET EXPLAIN turns the plan output on. SET PLANONLY works as in previous versions.

**Usage options**

- SET PLAN = simple plan + query execution
- SET PLANONLY = simple plan, no query execution
- SET PLAN + SET EXPLAIN = explained plan + query execution
- SET PLAN + SET EXPLAIN + SET PLANONLY = explained plan, no query execution
- SET EXPLAIN = explained plan + query execution
- SET EXPLAIN + SET PLANONLY = explained plan, no query execution

---

**Metadata Extract**

Claudio Valderrama C.

The metadata extract tool (-[e][x][tract] switch) was improved to create a script that takes the dependency order of objects properly into account.

A label has also been added, reflecting the deterministic flag for stored functions. (A. dos Santos Fernandes)

---

**Path to INPUT Files**

Adriano dos Santos Fernandes
The INPUT command will now use a relative path based on the directory of the last-opened, unclosed file in the chain to locate the next file.

**Command Buffer Size Increase**  
Adriano dos Santos Fernandes

The size of the *isql* command buffer has increased from 64 KB to 10 MB to match the new engine limits. See Tracker ticket [CORE-4148](#).

**Deterministic Label for Stored Functions**  
Adriano dos Santos Fernandes

A label was added in the SHOW FUNCTION command reflecting the *deterministic* flag for stored functions. See Tracker item [CORE-4940](#).

---

**fb_lock_print**

**Input Arguments**  
Dmitry Yemanov

*fb_lock_print* now accepts 32-bit integers as the input arguments for seconds and intervals. Previously they were limited to SMALLINT.

**Useability Improvements**  
Vlad Khorsun

A few other small improvements:

1. More detailed usage help is available from the command line (-help).
2. Events history and list of owners are no longer output by default: they may be requested explicitly if required. Header-only is the new default.
3. New -o[wners] switch to print only owners (locks) with pending requests

---

**gfix**

**-NoLinger Switch**  
Alex Peshkov

*gfix* has a new switch **-NoLinger** to provide a one-off override to the LINGER setting of a database.
For information regarding LINGER, see the write-up in the DDL chapter.

**Improvements to Validation Messages**
Vlad Khorsun

- Critical validation messages are now split from minor ones
- The table name is now returned in the text of validation contraint error messages, to help identify the error context

**Other Tweaks**

Some implementation annoyances were cleared up in several utilities.

**All Command-line Utilities**

**Resolution of Database Path**
Alex Peshkov

All utilities resolve database paths in `databases.conf` when they need to access a database file directly. But not all of them would follow the same rules when expanding a database name. Now, they do.

**Help and Version Information**
Claudio Valderrama C.

All command-line utilities except `gpre` and `qli` now present help and version information in a unified and coherent way.

No info yet at [CORE-2540](#).

**Hard-coded Messages Replaced**
Claudio Valderrama C.

Hard-coded messages were replaced with the regular parameterised-style ones in `tracemanager` and `nbackup`.

**Arbitrary Switch Syntax Clean-up**
Claudio Valderrama C.

Switch options in `qli` and `nbackup` were made to check the correctness (or not) of the abbreviated switch options presented.
Chapter 12

Compatibility Issues

In this section are features and modifications that might affect the way you have installed and used Firebird in earlier releases.

Where Are the Tools?

On Windows, you will find all of the executable programs, including the command-line and shell tools, in the Firebird installation (root) folder. In previous versions they were in a folder beneath the root folder, named ..\bin.

On POSIX platforms, you will find the tools and other executable programs in similar locations to those used for earlier versions. Exactly where depends on the distribution you are using.

Other Windows Changes

- Notice that the installer no longer gives the option to build fbclient.dll as gds32.dll or to locate it in the system folder. If your applications still require this, you will have to do it manually.

aliases.conf Is No More

The file aliases.conf is replaced by databases.conf in the Firebird root directory. The format for database aliases has not changed so you can copy/paste the contents of your existing aliases.conf file into databases.conf successfully. The new file is capable of carrying a lot more configuration detail, however, to enable database-level configuration of many features that were previously available only at the global server level.

Embedded Connections

For an embedded connection, an authenticated login is no longer required on POSIX platforms. If you provide a user name and password, the password is ignored. Applications may still require a user name and possibly a role name, due to SQL privileges in databases.

This feature is new for embedded Firebird on POSIX but it is the way embedded always worked on Windows, when the embedded engine was a separate executable.
By default, Firebird 3 is configured for the new authentication model which uses SRP to work with user passwords and generate unique session identifiers for traffic encryption. The security database (security3.fdb) has no predefined users. This is intentional.

However, when using the standard installers for Windows, Linux and MacOSX, the SYSDBA user is created during the final step of the installation, with a password that is either random or defined via a user input routine.

In a situation where the SYSDBA initialization step either fails, or is missing from an OS-specific install process, such as a Windows .zip kit install or a dedicated POSIX platform port, it may be necessary to initialize the security database manually for use with the SRP plugins. You will need to create the user SYSDBA and set up the password for it using SQL CREATE USER command syntax in embedded mode as your first step to getting access to databases and utilities.

The gsec utility can be used instead but, having been deprecated, it is not discussed here.

---

**Initialization Steps**

Initialization is performed in embedded mode using the *isql* utility. For an embedded connection, an authentication password is not required and will be ignored if you provide one. An embedded connection will work fine with no login credentials and “log you in” using your host credentials if you omit a user name. However, even though the user name is not subject to authentication, creating or modifying anything in the existing security database requires that the user be SYSDBA; otherwise, *isql* will throw a privilege error for the CREATE USER request.

The SQL user management commands will work with any open database. Because the sample database employee.fdb is present in your installation and already aliased in databases.conf, it is convenient to use it for the user management task.

1. **Stop the Firebird server.** Firebird 3 caches connections to the security database aggressively. The presence of server connections may prevent *isql* from establishing an embedded connection.

2. In a suitable shell, start an *isql* interactive session, opening the employee database via its alias:

   ```
   > isql -user sysdba employee
   ```

3. Create the SYSDBA user:
SQL> create user SYSDBA password 'SomethingCryptic';
SQL> commit;
SQL> quit;

Note
The SYSDBA user will have full administrator rights automatically. Do not assign the ADMIN role (RDB $ADMIN) to SYSDBA.

4. To complete the initialization, start the Firebird server again. Now you will be able to perform a network login to databases, including the security database, using the password you assigned to SYSDBA.

About Passwords

• An effective password, using the default user manager Srp, can be up to 20 characters, although a password of up to 255 characters will be valid.

• If you intend in future to configure the server to use legacy authentication (not recommended!), then only the first 8 characters of any password, including that of the SYSDBA, will be read.

• masterkey is NOT a recommended password for SYSDBA!

Additional Notes About Security Databases

Since Firebird 2, users—including SYSDBA—could not log in to the security database directly. With Firebird 3 it is possible to establish a direct, embedded connection to the security database. With the appropriate configuration parameters, you can control the ability to connect remotely to other security databases.

The gsec utility, although deprecated in Firebird 3, is still available for access to security3.fdb only. It cannot be used to manage a custom user database. It is recommended that database admins plan to move away from relying on gsec and become familiar with the newer SQL user management features.

Legacy Authentication

If you do not intend to use SRP encrypted log-ins right away and want to use the security database—security3.fdb—as you have done in previous Firebird versions, proceed as follows:

1. Using a text editor, open firebird.conf and find the entry for the parameter UserManager:

    #UserManager = Srp

    Delete the “#” symbol and change the value to:

    UserManager = Legacy_UserManager

2. Find the entry for the WireCrypt parameter:
#WireCrypt = Enabled (for client) / Required (for server)

Delete the “#” symbol and change the value to:

WireCrypt = Enabled
-- or, if you don't plan to use SRP encryption at all --
WireCrypt = Disabled

3. Find the entry for the **AuthServer** parameter:

#AuthServer = Srp, WinSspi, Legacy_Auth

Delete the “#” symbol and change the order of the arguments:

AuthServer = Legacy_Auth, Srp, WinSspi

4. Find the entry for the **AuthClient** parameter:

#AuthClient = Srp, WinSspi, Legacy_Auth

Delete the “#” symbol and change the order of the arguments:

AuthClient = Legacy_Auth, Srp, WinSspi

5. Save the changes.

6. Stop and restart Firebird for the changes to take effect.

### Legacy Passwords

- The old **masterke** password is available for your first login as SYSDBA. It is known to the whole world and should be changed as soon as possible.
- Reminder: Legacy authentication reads only the first 8 characters of any password.

---

**Upgrading a v.2.x Security Database**

Because of the new authentication model in Firebird 3, upgrading a v.2.5 security database (security2.fdb) directly for use under Firebird 3 is not possible. However, an upgrade procedure is in place to enable retention of the user account data—user name, first name, etc., but not passwords—from the security2.fdb database that was used under v.2.x. servers.

The SYSDBA user is not touched: after the upgrade procedure, it will remain as it was before.
Compatibility Issues

**Steps**

**Note**

- In the commands below, replace `masterkey` with the actual SYSDBA password for the server version, as appropriate.
- The procedure requires running the script `security_database.sql` that is located in the `misc/upgrade` directory of your Firebird 3 installation. These instructions assume you have a temporary copy of this script in the same directory as the `isql` executable.

1. Under the Firebird 2.5 server, back up the security database. Here, use the SYSDBA password for the v.2.5 installation:

   `gbak -user sysdba -pas masterkey -b {host/path}security2.fdb security2.fbk`

2. Under the Firebird 3 server, restore a copy of v.2.5 backup. Here, use the SYSDBA password for the v.3 installation:

   `gbak -user sysdba -pas masterkey -c security2.fbk {host/path}security2db.fdb`

3. Under the Firebird 3 server, go to the directory where the `isql` utility is located and run the upgrade script:

   `isql -user sysdba -pas masterkey -i security_database.sql {host/path}security2db.fdb`

   "security2db.fdb" is just a sample name for the database: it can be any preferred name.

4. The procedure will generate new, random passwords and will output them to screen afterwards. Capture the output and notify users of their new passwords.

5. When you are ready, shut down Firebird. Rename `security3.fdb` or move it another location; then rename the upgraded database to `security3.fdb`.

**Local Connections to Superserver on Windows**

In previous Firebird versions, a serverless protocol known as “Windows Local” was available to local clients connecting to Superserver on a Windows platform, using the XNET subsystem. A typical connection string looked like this:

`c:\Program Files\Firebird_2_5\examples\empbuild\employee.fdb`

Under the new unified server, that form of connection is no longer valid for a serverless client connection to Superserver. It attempts to load an embedded server. If you try whilst Superserver is connected to your database, you will get a refusal message to the effect “File is in use by another process”.

This is not a bug. Since Superserver clients share resources, another server (in this case, an embedded server) cannot attach a client to the same database that Superserver has any clients attached to.
Compatibility Issues

However, all is not lost. The XNET subsystem can still do local client sessions for Superserver. You just need a more elaborate connection string now:

\texttt{xnet://alias-or-path-to-database}

So, for our connection to the employee database:

\texttt{xnet://c:\Program Files\Firebird_3_0\examples\empbuild\employee.fdb}

or using an alias:

\texttt{xnet://employee}

\textbf{Note}

New connection strings are available as alternatives for other local connection protocols, too. For more information, see \textit{New Connection Formats for Local Clients on Windows}.

Configuration Parameters

The previously deprecated $\texttt{firebird.conf}$ parameters \texttt{CompleteBooleanEvaluation}, \texttt{OldColumnNaming} and \texttt{OldSetClauseSemantics} are no longer supported anymore and have been removed. The lack of one or more of these parameters may break your application code, so please check these settings in $\texttt{firebird.conf}$ on your older server version.

The parameters \texttt{UsePriorityScheduler}, \texttt{PrioritySwitchDelay}, \texttt{PriorityBoost}, \texttt{LegacyHash} and \texttt{LockGrantOrder} no longer have any use and have been removed.

\textbf{Important}

Ensure that you study the chapter \textit{Configuration Additions and Changes} in preparation for upgrading user software to Firebird 3.

System Tables are Now Read-only

The system tables (RDB$—) are now read-only. Attempting any DDL or insert/update/delete statement on them will be rejected.

SQL Language Changes

It will be necessary to pay attention to some changes in the SQL language implementation.
**Support for Mixed-Syntax Joins is Gone**

Improperly mixed explicit and implicit joins are no longer supported, in accordance with the SQL specification. It also means that, in the explicit `A JOIN B ON <condition>`, the condition is not allowed to refer to any stream except A and B. See Tracker ticket CORE-2812 for more details.

**Enforcement of Length Limit for Local Alias and Variable Names**

The names for column and table aliases and for local variables names in PSQL are now restricted to 31 bytes in length. This enforcement has been made to comply with the SQL specification that requires them to be regular SQL identifiers, in accordance with the implementation-dependent limits. In Firebird, SQL identifiers are limited to 31 bytes.

See also: CORE-2350

**Changes for User Names**

User names are treated as SQL identifiers and are accordingly now limited to 31 bytes in length.

Case-sensitive user names are also supported now. The CONNECT and CREATE DATABASE statements in `isql` will thus allow user names to be specified in double quotes.

**Cursor Output Columns Must Be Named**

The `DECLARE CURSOR` statement in PSQL now requires all of the output columns to be explicitly named or aliased. The same requirement applies to the `FOR SELECT ... AS CURSOR <cursor name> DO ...` statement in PSQL. This requirement is necessitated by the new capability to read cursor elements directly as pseudo columns, e.g. `MY_CURSOR.COLUMN_A/`.

Illustration:

```sql
create procedure sp_test
as
  declare c cursor for (select 1 /* as a */ from rdb$database);
  declare n int;
begin
  open c;
  fetch c into n;
  close c;
end
```

Statement failed, SQLSTATE = 42000
unsuccessful metadata update
-ALTER PROCEDURE SP_TEST failed
-Dynamic SQL Error
-SQL error code = -104
-Invalid command
“Cursor Stability” Effects

Some statements may now work differently due to the “cursor stability” improvement. Statements affected will be:

- those that modify the table that is being explicitly or implicitly selected from within the same statement
- (as a side effect) some MERGE statements, that might work differently if multiple matches are possible.

**Note**
The SQL standard stipulates that the MERGE statement must raise an error if multiple matches are found. Firebird is not so strict in this regard, but its behaviour should be considered undefined in these cases.

See also: CORE-3362, CORE-3317, CORE-4796.

Reserved Words

A number of new reserved keywords are introduced. Please refer to the chapter Reserved Words and Changes and ensure your DSQL statements and procedure/trigger sources do not contain any of those keywords as identifiers. Otherwise, it will be necessary either to use them quoted (in Dialect 3 only) or to rename them.

Pay special attention to the keywords INSERTING, UPDATING and DELETING in your PSQL modules. They are now reserved words and thus must not be used as identifiers.

Also check very carefully that the keyword BOOLEAN is not used as a domain name in your databases.

Trace Configuration Syntax

The format of sections and key-value pairs in the trace configuration file has changed:

**Old format:**

```xml
<database %[\/]\{test\|azk2\|rulez\}.fdb>
  enabled true
  time_threshold 100
  log_statement_finish true
</database>
```

**New format:**

```xml
database = %[\/]\{test\|azk2\|rulez\}.fdb
{
  enabled = true
}
```
time_threshold = 100
log_statement_finish = true
}

**Important**
Pay attention to the “database” section header and equality signs between parameter names and values.

---

**Unexpected GDSCODE Errors**

An unexpected GDSCODE may be returned for UPDATE/DELETE statements, viz. `isc_deadlock` instead of `isc_lock_conflict`.

In fact, the complete error stack previously contained `{isc_lock_conflict, isc_deadlock, isc_update_conflict}`, whereas it now contains, more correctly, `{isc_deadlock, isc_update_conflict}`.

However, the GDSCODE system variable returns the first error element, thus causing possible compatibility issues. Be sure that, besides `isc_lock_conflict`, your error handlers for updates and deletes also check for the other error codes, `isc_deadlock` and `isc_update_conflict`.

---

**Remote Client Recognition**

From this version forward, the remote client address for TCPv4 and TCPv6 protocols in `$ATTACHMENTS.MON$REMOTE_ADDRESS` includes the port number, separated from the IP address with a slash:

```
<IP address>/<port>
```

A new column in that table also reports the host name of the remote client.

For details, see *Changes to Client Address Reporting* in the chapter entitled *Changes to the Firebird API and ODS*. 
Chapter 13

Bugs Fixed

Firebird 3.0.0 Release: Bug Fixes

The following bug fixes were reported prior to the v.3.0.0 official release:

Core Engine

(CORE-5184) An assertion would occur in the cloop dispatcher when trying to save exception information in the status interface.

fixed by A. Peshkov

(CORE-5182) Failure to find a record on re-fetch would produce inconsistent effects on subsequent statement execution.

fixed by D. Yemanov

(CORE-5166) Unique key violation on a BOOLEAN column was returning an erroneous message.

fixed by A. dos Santos Fernandes

(CORE-5165) HAVING COUNT(*) NOT IN ( <Q> ) would erroneously exclude a record from the outer result set. ( <Q> represents a result set without nulls)

fixed by A. dos Santos Fernandes

(CORE-5161) It was possible to create a unique index on non-unique data.

fixed by V. Khorsun

(CORE-5159) Regression: the engine could throw transliteration errors when running from a non-ASCII system path.
Bugs Fixed

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-5157)  The server could go into an infinite loop when data was being sent from a client to the server in small packets.

fixed by A. Peshkov

~ ~ ~

(CORE-5149)  Regression: LEFT JOIN was incorrectly pushing COALESCE into the inner stream, causing wrong results.

fixed by D. Yemanov

~ ~ ~

(CORE-5147)  CREATE TRIGGER was failing with the error "Ambiguous field name between table B and table A".

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-5146)  Join order was sub-optimal if one table had a selective predicate and MIN was calculated for the other one.

fixed by D. Yemanov

~ ~ ~

(CORE-5144)  Deadlock could occur when a database was being encrypted or decrypted under high parallel load.

fixed by A. Peshkov

~ ~ ~

(CORE-5133)  ALTER SEQUENCE RESTART WITH was failing to change the initial value.

fixed by D. Yemanov

~ ~ ~

(CORE-5130)  Minor regression: wrong error message was being returned on an attempt to compile a view with both a subquery and the WITH CHECK OPTION clause in its definition—an illegal combination.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-5122)  The optimizer could avoid using an expression index if the query was running in a different connection character set to the one that was in effect when the index was created.

fixed by D. Yemanov
Bugs Fixed

~ ~ ~

(CORE-4645) Internal Firebird consistency check (cannot find tip page (165), file: tra.cpp line: 2375) would occur if too many transactions were created for one connection to a read-only database. This was a very old bug, dating back to InterBase 6.

fixed by V. Khorsun

~ ~ ~

Server Crashes

(CORE-5153) Regression: The server would crash when aggregate functions were used together with the NOT IN predicate.

fixed by A. dos Santos Fernandes

~ ~ ~

Utilities

gbak

(CORE-5143) gbak restore could fail when there was some SQL function accessing a table while the switch -O(NE_AT_A_TIME) was in use.

fixed by D. Yemanov

~ ~ ~

gfix

(CORE-5140) An attempt to set the number of page buffers to a non-supported value would return the wrong error message.

fixed by V. Khorsun

~ ~ ~

isql

(CORE-5194) An invalid COMPUTED BY definition was being generated by the 'extract metadata' operation.

fixed by A. dos Santos Fernandes

~ ~ ~
**Monitorings**

(CORE-5179) Database-level statistics (MON$STAT_GROUP = 0) could contain multiple rows under concurrent load.

fixed by D. Yemanov

~ ~ ~

**User Management**

(CORE-5189) The codes of operation for user management plug-ins were missing from the public API.

fixed by A. Peshkov

~ ~ ~

(CORE-5162) Users with the same name can co-exist in the security database if they are created by different plug-ins. However, tags or attributes for a user created using one plug-in were being displayed in the SECS virtual tables for the same-name user created by the other plug-in.

fixed by A. Peshkov

~ ~ ~

(CORE-5155) [CREATE OR] ALTER USER statement would fail if the PASSWORD parameter was present and was not specified immediately after the USER parameter.

fixed by A. Peshkov

~ ~ ~

**Builds**

(CORE-5181) The build would fail if HAVE_MMAP was not defined.

fixed by A. Peshkov

~ ~ ~

(CORE-5180) Root rights were unnecessarily required to build a Linux package.

fixed by A. Peshkov

~ ~ ~

(CORE-5134) The samples of dbcrypt-related plug-ins contained references to internal files.

fixed by A. Peshkov

~ ~ ~
Firebird 3.0 Release Candidate 2: Bug Fixes and Minor Improvements

The following tweaks and bug fixes were reported prior to v.3.0.0 release candidate 2:

**Minor Improvements**

(CORE-4996) NEW: SECURITY2.FDB —> SECURITY3.FDB upgrade feature

*Implemented by A. Peshkov*

~ ~ ~

(CORE-1538) NEW: An option was added to the API function `isc_database_info()` to return the number of free pages in a database.

*Implemented by V. Khorsun*

~ ~ ~

(CORE-5112) IMPROVEMENT: The error messages for a malformed SQDLA were improved.

*Implemented by D. Sibiryakov*

~ ~ ~

(CORE-5100) IMPROVEMENT: Improved control over database crypt keys, see Encryption Tasks.

*Implemented by A. Peshkov*

~ ~ ~

(CORE-5083) IMPROVEMENT: A method was provided to set the correct offsets in user-implemented IMessageMetadata.

*Implemented by A. Peshkov*

~ ~ ~

(CORE-5077) IMPROVEMENT: In `isql`, SHOW DATABASE includes the encryption status of the database, as does `gstat -h[eaders].`

*Implemented by A. Peshkov*

~ ~ ~

(CORE-5063) IMPROVEMENT: In `isql`, SHOW DATABASE includes the numbers of used and free pages in the database.

*Implemented by V. Khorsun*

~ ~ ~
**Bugs Fixed**

*(CORE-5028)* IMPROVEMENT: The remote port number was added to the monitoring table MON $ATTACHMENTS.

*Implemented by V. Khorsun*

~ ~ ~

*(CORE-4978)* IMPROVEMENT: Validation and repair of lost data pages have been improved.

*Implemented by R. Simakov, V. Khorsun*

~ ~ ~

*(CORE-2493)* IMPROVEMENT: The IP address of the remote host is now appended to error messages in firebird.log for TCP connections.

*Implemented by V. Khorsun*

~ ~ ~

**Bugs**

**Core Engine**

*(CORE-5110)* A false primary or foreign key violation could be reported when an attachment used the isc_dpb_no_garbage_collect flag.

*fixed by V. Khorsun*

~ ~ ~

*(CORE-5104)* SELECTs from an external data source would become slow.

*fixed by V. Khorsun, A. Peshkov*

~ ~ ~

*(CORE-5097)* COMPUTED BY expressions were not being converted to their data type inside the engine.

*fixed by A. dos Santos Fernandes*

~ ~ ~

*(CORE-5093)* ALTER COLUMN was not changing the type of a computed column where it should.

*fixed by A. dos Santos Fernandes*

~ ~ ~

*(CORE-5082)* The server was not validating the correctness of the user/password pair provided in the EXECUTE STATEMENT operation.

*fixed by A. Peshkov*

~ ~ ~
**Bugs Fixed**

(CORE-5078) The "Invalid BLOB ID" error was being thrown unexpectedly after a large VARCHAR column was changed to text BLOB.

fixed by V. Khorsun

~ ~ ~

(CORE-5075) ON DISCONNECT trigger with EXECUTE STATEMENT was not working and would lead to a server crash when it was recreated.

fixed by V. Khorsun

~ ~ ~

(CORE-5062)

fixed by A. dos Santos Fernandes Function CHAR_TO_UUID on an indexed column would throw the error "expression evaluation not supported".

~ ~ ~

(CORE-5060) CREATE VIEW that selected from a system table would fail, despite all grants being available.

fixed by A. Peshkov

~ ~ ~

(CORE-5056) The write lock on the database file was being cleared when retrieving the database statistics via the Services API.

fixed by A. Peshkov

~ ~ ~

(CORE-5049) Byte-length for view columns was being calculated incorrectly.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-5047) mapUser() routine would establish a secondary attachment using the wrong provider.

fixed by A. Peshkov

~ ~ ~

(CORE-5039) Connecting to a service with an invalid service name would yield an incorrect error message.

fixed by A. Peshkov

~ ~ ~

(CORE-5034) A delay of at least 5 seconds could occur on disconnect if the disconnect followed closely on an Event Manager initialization.

fixed by V. Khorsun
Bugs Fixed

~ ~ ~

(CORE-5033) The file fb_user_mapping could be left behind in the Firebird lock directory after a correct shutdown of all Firebird worker processes.

fixed by V. Khorsun

~ ~ ~

(CORE-5032) Connection to a database with its own security database was not possible in the absence of security3.fdb.

fixed by A. Peshkov

~ ~ ~

(CORE-5026) Firebird authentication would give up after failing to establish authentication via the first common plugin.

fixed by A. Peshkov

~ ~ ~

(CORE-5020) An ORDER BY clause on the key columns of a compound index could disable usage of other indices.

fixed by D. Yemanov

~ ~ ~

(CORE-5018) Non-indexed predicates might not have been applied immediately after retrieval when tables were being joined.

fixed by D. Yemanov

~ ~ ~

(CORE-5017) Interrupt of aux connection during TCP setup phase would cause a server exit due to an unhandled exception.

fixed by A. Peshkov

~ ~ ~

(CORE-5014) Interrupt of aux connection during TCP setup phase was producing unclear error messages in firebird.log.

fixed by A. Peshkov

~ ~ ~

(CORE-4997) Races would occur when two threads tried simultaneously to establish the first embedded connection to the engine.

fixed by A. Peshkov

~ ~ ~
The Keyholder plug-in was being loaded on connection to a database but not on creating a database.

fixed by A. Peshkov

The REVOKE operator could modify the privileges granted to system tables at database creation time.

fixed by A. Peshkov

The SEC$USERS table became unavailable when an error occurred in any configured UserManager plug-in.

fixed by A. Peshkov

Real errors during connection to a security database were being hidden by the Srp user manager.

fixed by A. Peshkov

Retrieving the current user name when using Windows trusted authentication and an OS user name containing non-ASCII characters would cause an error.

fixed by A. Peshkov, D. Yemanov

Transactions with isc_tpb_autocommit could hang the server.

fixed by D. Yemanov

Dependencies with views were exhibiting a problem.

fixed by A. dos Santos Fernandes

SET AUTODDL OFF and sequence of CREATE + ALTER + DROP VIEW would lead to a server crash.

fixed by A. dos Santos Fernandes
Bugs Fixed

(CORE-5087) A database shutdown could cause a server crash if multiple attachments were running EXECUTE STATEMENT.

fixed by V. Khorsun

(CORE-5016) The server could crash during garbage collection when a DELETE is executed after adding a new referencing column.

fixed by D. Yemanov

(CORE-5015) The server could crash when an event was posted from an ON DISCONNECT trigger.

fixed by V. Khorsun

(CORE-4982) The server could crash during a select from a procedure if an exception was thrown while backing out the current savepoint.

fixed by D. Yemanov

Database Encryption

(CORE-5105) Deadlocks could occur when a database was initially encrypted.

fixed by A. Peshkov

(CORE-5044) Pages were being encrypted for each shadow separately.

fixed by A. Peshkov

(CORE-5041) A database could be corrupted after ALTER DATABASE ENCRYPT/DECRYPT was issued multiple times per database open time.

fixed by A. Peshkov

(CORE-4999) Trying to access a database with an invalid crypt key would cause a deadlock.

fixed by A. Peshkov

(CORE-4990) Databases were not being decrypted completely.
fixed by A. Peshkov

~ ~ ~

**API/Remote Interface**

**(CORE-5102)**  
fbclient.dll API calls were (wrongly) modifying the FPU control word.

fixed by V. Khorsun

~ ~ ~

**(CORE-5067)**  
New connections were being blocked as a consequence of sweeps of the security database taking too long.

fixed by V. Khorsun

~ ~ ~

**(CORE-5042)**  
The V.3 client could not create a database on a V.2.5 server.

fixed by A. Peshkov

~ ~ ~

**(CORE-5029)**  
A crash could occur in fbclient when a multi-threaded application was using the XNET protocol.

fixed by V. Khorsun

~ ~ ~

**(CORE-4998)**  
Neither client nor server could close the connection after failed authentication.

fixed by V. Khorsun

~ ~ ~

**Utilities**

**gbak**

**(CORE-5102)**  
Restore was slow when the database contained many small tables with indices.

fixed by V. Khorsun

~ ~ ~

**gstat**

**(CORE-5088)**  
A segmentation fault could occur while `gstat -h` was closing.
fixed by A. Peshkov

---

**isql**

(CORE-5092) The EXTRACT command would lose the data types of COMPUTED BY columns.

fixed by A. dos Santos Fernandes

---

(CORE-5089) Metadata extraction (isql -X): A "CREATE PROCEDURE/FUNCTION" statement would retain the TYPE OF reference to a table column that no longer existed, if the procedure or function had a parameter of that type declared at the time it was created.

fixed by A. dos Santos Fernandes

---

(CORE-5061) Plan output was being unexpectedly truncated after a query was simplified by shortening it.

fixed by D. Yemanov

---

**Monitoring**

(CORE-4708) Content of MON$EXPLAINED_PLAN in MON$STATEMENTS was being truncated if it exceeded the 32KB limit.

fixed by D. Yemanov

---

(CORE-4179) The MON$ tables became unavailable when there was intensive disk-bound load.

fixed by D. Yemanov

---

**Builds and Installers**

(CORE-5053) (POSIX): changeServerMode.sh could mess with configuration.

fixed by A. Peshkov

---

(CORE-5002) (WINDOWS): The Windows installer exhibited some character set confusion (ANSI vs. UTF8) in its GUI for non-English environments.

fixed by P. Reeves
Bugs Fixed

~ ~ ~

(CORE-4910)  (WINDOWS): The Windows build would fail if a path to source files contained spaces.

fixed by D. Sibiryakov

~ ~ ~

(CORE-4862)  (WINDOWS): ibase.h from the x64 package included a nonexistent file.

fixed by D. Yemanov

~ ~ ~

(CORE-4842)  (POSIX): the installer script would fail in some Linux environments.

fixed by A. Peshkov, M. Masyk

~ ~ ~

Firebird 3.0 Release Candidate 1:  
Bug Fixes and Minor Improvements

The following bug fixes and tweaks were reported prior to v.3.0.0 release candidate 1:

Minor Improvements

(CORE-4911)  IMPROVEMENT: Most errors that occur in providers are fatal and it is pointless to try other providers in the hope another will succeed when that one failed. As an example, if the remote redirector detects that the format of the database connection string matches a certain protocol, trying other providers will be futile if connection on that protocol failed. Or, suppose the engine reports some internal database error (other than invalid ODS version) it will be no use trying to open the same database through the loopback provider.

A better option appeared to be to enumerate the codes for those errors after which it would make sense to try other providers—primarily isc_unavailable and isc_no_priv. Given the desirability of retaining the content of the status vector as-is, the solution was to add some more error codes.

Implemented by A. Peshkov

~ ~ ~

(CORE-4898)  IMPROVEMENT: Creation and loading of functions was slow when there were many functions in the database. This process has been given a speed boost.

Implemented by A. dos Santos Fernandes

~ ~ ~

(CORE-4880)  IMPROVEMENT: Creating packages containing many functions was slow. This process has been given a speed boost.

Implemented by A. dos Santos Fernandes
Bugs Fixed

~ ~ ~

(Core-4939) IMPROVEMENT: I/O operations on the nBackup backup file are now aligned at the page size boundary

Implemented by V. Khorsun

~ ~ ~

(Core-4936) IMPROVEMENT: Internal contention in the background garbage collector has been reduced

Implemented by V. Khorsun

~ ~ ~

(Core-4935) IMPROVEMENT: Performance was improved on backing out a dead record version or undoing a newly created primary record version when the back-version chain gets too long

Implemented by V. Khorsun

~ ~ ~

Bugs

Core Engine

(Core-4984) Ordering by a compound index together with a range condition was giving wrong results

fixed by D. Yemanov

~ ~ ~

(Core-4954) Packaged procedure with a default input value would not be called if the default parameter was not specified

fixed by A. dos Santos Fernandes

~ ~ ~

(Core-4952) SuperServer could hang on database shutdown

fixed by V. Khorsun

~ ~ ~

(Core-4951) The message “Incorrect parameters provided to internal function INF_transaction_info Invalid token” was reported when trying to commit a two-phase-commit transaction

fixed by A. Peshkov

~ ~ ~

(Core-4950) Statistics could not be recalculated or updated for any system index

fixed by A. dos Santos Fernandes
Bugs Fixed

~ ~ ~

(CORE-4947)  A compound ALTER TABLE statement that tried to ADD and DROP the same check constraint would fail

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4943)  Casting a Dialect 1 date to string would break when a domain with a check constraint was present

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4938)  Operations (get statistics, change forced writes, make backup) could not be completed on a database when working in embedded mode

fixed by V. Khorsun

~ ~ ~

(CORE-4929)  PSQL source with "ELSE IF ( <expr> ) THEN" block containing commands to manipulate an explicit cursor would not compile

fixed by D. Yemanov

~ ~ ~

(CORE-4921)  Predicate IS [NOT] DISTINCT FROM was not being pushed into unions/aggregates, resulting in sub-optimal plans

fixed by D. Yemanov

~ ~ ~

(CORE-4917)  ALTER DOMAIN ... TO <new_name> would allow the <new_name> to be specified in the same pattern as domains generated by the system, i.e., matching to 'RDB$[[:DIGIT:]]*

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4914)  Moving a database into the 'full shutdown' state could corrupt indexes on long-key text columns

fixed by V. Khorsun

~ ~ ~

(CORE-4909)  MERGE / HASH JOINs would produce incorrect results when VARCHAR join keys differed only by trailing spaces

fixed by D. Yemanov

~ ~ ~
Bugs Fixed

**(CORE-4906)** An access violation would occur if the server was shut down with active trace session(s) fixed by V. Khorsun

**(CORE-4904)** Index corruption could occur while inserting data into a long-key-indexed field fixed by V. Khorsun

**(CORE-4902)** Failure of assertion (m_format->fmt_length == m_data.getCount()) fixed by D. Yemanov, V. Khorsun

**(CORE-4897)** The *fbsvcmgr* could produce a broken log when restoring a database fixed by A. Peshkov

**(CORE-4890)** Placing a comment (single-lined or multi-lined) between the final END statement and the terminator character (^) in stored procedure code produced a compiling error if the utility tool used the legacy API fixed by A. Peshkov

**(CORE-4889)** Using *fbsvcmgr* with action_trace_start under SuperSsrver would cause a livelock and prevent attachments using local protocol fixed by V. Khorsun

**(CORE-4887)** An AFTER CREATE/ALTER PACKAGE DDL trigger would run before inserts and updates of RDB$PROCEDURES and RDB$FUNCTIONS, whereas it should run after all changes to the system tables. fixed by A. dos Santos Fernandes

**(CORE-4884)** A script containing EXECUTE BLOCK with MULTIPLE nested BEGIN..END statements would cause a crash during parsing fixed by A. dos Santos Fernandes

**(CORE-4872)** The V.3 fbclient could not work with servers older than V.2.5 via the remote protocol fixed by D. Yemanov
A segmentation fault could occur when working with a saved exception in a request

fixed by A. Peshkov

The client library was handling non-UTF8 representation of international characters incorrectly in the SPB

fixed by A. Peshkov

MERGE ... WHEN NOT MATCHED ... RETURNING was returning wrong (non-null) values when no insert was performed

fixed by A. dos Santos Fernandes

Altering a trigger to attempt to change the table to which it belongs did not succeed but it failed with bizarre results and an unhelpful error message

fixed by A. dos Santos Fernandes

GRANT UPDATE(c) ON T TO U01 WITH GRANT OPTION: user U01 would be unable to revoke this privilege if some DML was executed before REVOKE

fixed by A. Peshkov

“Invalid request BLR at offset 361 context already in use (BLR error)” was the wrong message for the error concerned, occurring only when the context count was near the 256 limit

fixed by D. Yemanov

Deadlock could occur during attachment

fixed by V. Khorsun

A column CHECK constraint could not be created if the column was based on a domain

fixed by A. dos Santos Fernandes

Regression: wrong handling in FOR-cursor when a “NOT EXISTS( select from <view> )” expression was used to check results obtained from a stored procedure

fixed by D. Yemanov
Bugs Fixed

(CORE-4292) The server would ignores asynchronous requests (monitoring or cancellation) while preparing a query with a lot of windowed functions

fixed by A. dos Santos Fernandes

(CORE-4279) Invalid error message: “CHARACTER SET OCTETS is not defined” when creating a database specifying `isc_dpb_lctype = OCTETS`

fixed by A. dos Santos Fernandes

(CORE-4277) A database could be created with an invalid character set as its default

fixed by A. dos Santos Fernandes

(CORE-4276) CREATE TABLE would throw an error if a column was being defined with the character set `DOS775`

fixed by A. dos Santos Fernandes

(CORE-4107) (subquery + derived table + union) would produce a wrong result set

fixed by D. Yemanov

(CORE-3717) Anomalies existed between the three methods of user management, regarding valid user names and passwords

fixed by A. Peshkov

(CORE-3545) Validation of domain constraints in PSQL was inconsistent

fixed by A. dos Santos Fernandes

POSIX-Specific

(CORE-4919) On AIX and Solaris, all processes on a Classic server would hang after the death of one process

fixed by A. Peshkov
Bugs Fixed

Windows-Specific

(CORE-4859) Embedded server could not work without the folder C:\ProgramData\firebird

fixed by V. Khorsun

(CORE-4860) Online validation on Windows would fail if the dbname argument contained forward slash (/) and a concurrent attachment existed that also used '/'

fixed by V. Khorsun

Services API

(CORE-4939) When using the Services API to manage users (from fbsvcmgr or by building a SPB in some program, for example) it was required—but not documented—to place the user name parameter immediately after the desired action (add/modify). If any other parameter was placed before the user name, the Services Manager on the server would go crazy and build a wrong command line for the gsec service.

fixed by A. Peshkov

Utilities

fbsvcmgr

(CORE-4855) Online validation during DML activity in another connection would lead to errors: “Error while trying to read from file” and “Page in use during flush (210)”

fixed by V. Khorsun

(CORE-4876) A named trace session launched by fbsvcmgr with a non-empty value for the parameter trc_name could not be stopped using its name

fixed by V. Khorsun

gbak

(CORE-4928) The connection information in an ON CONNEC T trigger could not be saved if the connection was established by gbak
fixed by A. dos Santos Fernandes

---

gfix

(CORE-4899) Using gfix -online in Classic returned the message "IPrvider::attachDatabase failed when loading mapping cache" if access was using a remote protocol

fixed by A. Peshkov

---

(CORE-4865) Online validation could return a false report of double-allocated pages

fixed by V. Khorsun

---

(CORE-3548) gfix would return an error after correctly shutting down a database

fixed by A. Peshkov

---

isql

(CORE-4882) The isql INPUT command (or the -i option at the command line) was reading large (> 64K) lines incorrectly

fixed by A. dos Santos Fernandes

---

(CORE-4864) CREATE DATABASE fails in isql

fixed by A. Peshkov

---

(CORE-4870) In isql, SET COUNT ON reports the wrong number of affected rows when issuing UPDATE on a view which was created WITH CHECK OPTION

fixed by A. dos Santos Fernandes

---

Firebird 3.0 Second Beta Release: Bug Fixes

The following bug fixes were reported prior to the v.3.0.0 Beta 2 release:
Core Engine

(Core-4819) The engine failed to check the validity of \texttt{RETURNING\_VALUES} for \texttt{EXECUTE PROCEDURE} and \texttt{INTO} for \texttt{EXECUTE STATEMENT} could lead to bugchecks.

\textit{fixed by A. dos Santos Fernandes}

(Core-4809) \texttt{HASH/MERGE JOIN} was not used for more than two streams if they were joined via \texttt{USING/NATURAL} clauses and the join was based on \texttt{DBKEY} concatenations.

\textit{fixed by D. Yemanov}

(Core-4806) Regression: generators could be seen and modified by unprivileged users.

\textit{fixed by D. Yemanov}

(Core-4798) Regression: \texttt{MIN()} or \texttt{MAX()} with a join would ignore possible index navigation.

\textit{fixed by D. Yemanov}

(Core-4789) Timestamps that were out of valid range were not prevented from being cast to \texttt{VARCHAR} or \texttt{CHAR}.

\textit{fixed by A. dos Santos Fernandes}

(Core-4786) When an attempt was made to insert a duplicate value into a primary or unique key column, where the length of the key was 127 characters or more, the problematic key was not shown in the error message.

\textit{fixed by D. Yemanov}

(Core-4781) Maximum string length (32765 bytes) was not being validated by the parser.

\textit{fixed by A. dos Santos Fernandes}

(Core-4774) Table aliasing was being required unnecessarily when doing \texttt{UPDATE ... RETURNING RDB$ pseudo-columns}.

\textit{fixed by A. dos Santos Fernandes}
CREATE USER ... TAGS ( argument_1 = 'value1', ..., argument_N = 'valueN' ) would return wrong results from the statement when there were many arguments.

fixed by A. Peshkov

In CREATE USER ... TAGS ( attr = 'prefix #suffix' ) the character “#” in the attribute value would cause the subsequent characters to be dropped from storage.

fixed by A. Peshkov

A user with non-ascii (multi-byte) characters in the name could not be created.

fixed by A. Peshkov

A request synchronization error could occur when building an expression index.

fixed by V. Khorsun

A parameterized exception would produce wrong output when the number of arguments exceeded seven.

fixed by A. dos Santos Fernandes

Manipulations with a GTT from several attachments (using EXECUTE STATEMENT with an ON EXTERNAL clause and different roles) could lead to the error “Internal Firebird consistency check (invalid SEND request (167), file: JrdStatement.cpp line: 325)”.

fixed by V. Khorsun

Firebird could hang in the embedded mode.

fixed by V. Khorsun

EXECUTE STATEMENT using BLOB parameters would result in an “Invalid BLOB ID” error.

fixed by A. dos Santos Fernandes

The error “Invalid BLOB ID” could occur when retrieving MON$STATEMENTS.MON $SQL_TEXT using EXECUTE STATEMENT with an ON EXTERNAL clause with the db_connect argument unspecified.
Bugs Fixed

fixed by V. Khorsun

~ ~ ~

(CORE-4744) The statement ALTER DATABASE SET DEFAULT CHARACTER SET exhibited two faults: 1) it would take effect only once for the current attachment; 2) it was not checking that the new character set existed until it was actually used.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4743) A granted role would not work with a non-ASCII user name.

fixed by A. Peshkov

~ ~ ~

(CORE-4740) SIMILAR TO with quantifier \{n,\} in the pattern was failing in v.2.5 with the error “Invalid pattern” and was producing strange results in v.3.0.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4733) The command `ALTER TABLE <T> ALTER TYPE <C> DOMAIN_WITH_NOT_NULL` was not verifying the data in column `<C>` and would make incorrect assignments in `<C>` to ZERO / JULIAN_DATE / ASCII(0) for types INT, TIMESTAMP and VARCHAR.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4725) Inconsistencies were exhibited with ALTER DOMAIN and ALTER TABLE involving DROP NOT NULL and PRIMARY KEYs.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4719) The message “Statement failed, SQLSTATE = 00000 + unknown ISC error 0” would appear when issuing REVOKE ALL ON ALL FROM <existing_user>.

fixed by A. Peshkov

~ ~ ~

(CORE-4713) A “BLOB not found” error would be thrown at rollback after inserting into a table with an expression index.

fixed by V. Khorsun

~ ~ ~

(CORE-4712) Messages stating “Error in isc_release_request() ... when working with legacy security database” were appearing in `firebird.log` for the Classic server when connecting with legacy authentication.
Bugs Fixed

fixed by A. Peshkov

~ ~ ~

(CORE-4702) Join order was less optimal than in v2.x.

fixed by D. Yemanov

~ ~ ~

(CORE-4684) An error was being thrown while preparing a complex query (“Too many Contexts of Relation/Procedure/Views. Maximum allowed is 256”.)

fixed by D. Yemanov

~ ~ ~

(CORE-4675) Conditions like WHERE `<field> = <cursor>.<field>` would not use an existing index.

fixed by D. Yemanov

~ ~ ~

(CORE-4673) A computed index based on a computed column would store NULL for all its keys.

fixed by D. Yemanov

~ ~ ~

(CORE-4670) Constraint violation errors could be swallowed in some cases.

fixed by D. Yemanov

~ ~ ~

(CORE-4668) SELECT FROM MON$TABLE_STATS did not work on Superclassic and Classic.

fixed by D. Yemanov

~ ~ ~

(CORE-4665) Search criteria would produce wrong results on WHERE `<field_C> STARTING WITH <:value> ORDER BY <field_N>` when field_C was the leading part of a compound index key consisting of { field_C, field_N }.

fixed by D. Yemanov

~ ~ ~

(CORE-4659) A bug was introduced when implementing support for multiple user managers, whereby the error “Missing security context for ....\SECURITY3.FDB” could appear when performing some operations where per-database security was configured.

fixed by A. Peshkov

~ ~ ~
Bugs Fixed

(CORE-4656) The server could hang while chasing dead record versions.
fixed by D. Starodubov

(CORE-4655) MSVC8 and MSVC9 builds of Beta 1 were failing.
fixed by D. Starodubov

(CORE-4648) A user with the RDB$ADMIN role was denied the privilege for CREATE DATABASE.
fixed by A. Peshkov

(CORE-4644) Security context and open database errors could present under heavy concurrent load.
fixed by A. Peshkov

(CORE-4634) The error “No current record for fetch operation” could occur when the same indexed column was specified in both the WHERE and the ORDER BY clauses.
fixed by D. Yemanov

(CORE-4631) When the shared memory region backing the lock table could not be remapped, the status vector would return the error message “Lock manager out of room” without any low-level details (e.g. OS level error), thus hiding the real cause of the problem.
fixed by D. Yemanov

(CORE-4624) Firebird's handling of ':' characters in mount table entries was invalid.
fixed by A. Peshkov

(CORE-4623) PSQL variables based on “Domain” and “Type Of” referring to BLOB with sub_type < 0 were no longer working.
fixed by A. dos Santos Fernandes

(CORE-4622) Triggers with the UPDATE OR INSERT statement and IIF() were not working as expected.
fixed by D. Yemanov
Rollback was failing to undo changes when a MERGE statement was updating the same target rows multiple times and the optimizer used PLAN MERGE.

fixed by D. Yemanov

EXECUTE STATEMENT was inflating the CHAR_LENGTH() size for VARCHARs.

fixed by A. dos Santos Fernandes

The REPLAC() function was not working correctly with multi-byte character sets.

fixed by A. dos Santos Fernandes

The embedded engine would try to load the ICU DLLs from PATH folders.

fixed by A. Peshkov

During the “linger” period, problems would occur when attempting to change some database properties.

fixed by A. Peshkov

The embedded server was trying to load UDFs from wrong place.

fixed by V. Khorsun

The incorrect error message was returned for PSQL functions when the number of actual arguments did not match the number of formal arguments.

fixed by D. Yemanov

Selecting from a table with ICU columns would fail where the database had been created on Linux with ICU 4.2.1 and then copied to Windows.

fixed by A. Peshkov

BUGCHECK(183) was thrown when using a cursor with ORDER BY ID+0 and FOR UPDATE WITH LOCK.

fixed by D. Yemanov
Bugs Fixed

~ ~ ~

(CORE-4539) The server would not accept the right plan.

fixed by D. Yemanov

~ ~ ~

(CORE-4507) Procedure source could not be deleted.

fixed by D. Yemanov

~ ~ ~

(CORE-4472) The message “Modifying function <F> which is currently in use” was displayed when <F> was called from an internal function declared in another unit.

fixed by D. Yemanov

~ ~ ~

(CORE-4384) Problems would occur when a table grew beyond 65535 pointer pages.

fixed by D. Yemanov

~ ~ ~

(CORE-4383) Index and BLOBs garbage collection was not working for in-place updates.

fixed by D. Sibiryakov & D. Yemanov

~ ~ ~

(CORE-4382) User savepoints were not being released on COMMIT.

fixed by D. Sibiryakov

~ ~ ~

(CORE-4369) BUGCHECK(177) was being thrown for a MERGE involving multiple matches.

fixed by D. Yemanov

~ ~ ~

(CORE-4280) Stored PSQL functions would accept duplicate input arguments.

fixed by D. Yemanov

~ ~ ~

(CORE-4161) It was not possible to insert records into a table having a column **GENERATED BY DEFAULT AS IDENTITY** in its DDL.

fixed by D. Yemanov

~ ~ ~
Bugs Fixed

(Core-3373) It was possible to store a 31-character string in a VARCHAR(25) column.
fixed by A. dos Santos Fernandes

(Core-2932) Field position would be wrong after ALTER POSITION.
fixed by A. dos Santos Fernandes

(Core-2848) Page-level “lock conversion denied” or “lock denied” errors could occur under high load when no deadlocks were apparent.
fixed by V. Khorsun

(Core-214) COUNT (DISTINCT ...) was too slow.
fixed by D. Yemanov

Server Crashes

(Core-4766) An access violation would occur on an attempt to manage the users list using EXECUTE STATEMENT on behalf of a non-SYSDBA user having the RDB$ADMIN role.
fixed by V. Khorsun

(Core-4705) Superclassic would crash at disconnect after running EXECUTE STATEMENT as non-current user.
fixed by D. Yemanov

(Core-4653) Infinite mutual stored procedure calls were causing the server to crash instead of returning the appropriate error, “Too many concurrent executions of the same request”.
fixed by V. Khorsun

(Core-4075) The server would bugcheck or crash on an exception in a calculated index.
fixed by D. Yemanov

(Core-3632) The application would crash after calling fork() in a process using libfbembed.so.
Bugs Fixed

fixed by A. Peshkov

~ ~ ~

**API/Remote Interface**

**(CORE-4795)** fbclient was executing the event callback function twice.

fixed by A. Peshkov

~ ~ ~

**(CORE-4794)** `isc_cancel_events()` was returning the error “invalid events id (handle) (code: 335545021)” if an event was no longer queued.

fixed by A. Peshkov

~ ~ ~

**(CORE-4788)** The Superclassic server could hang when receiving a network packet.

fixed by A. Peshkov

~ ~ ~

**(CORE-4785)** Bad `op_execute` packet would kill the server.

fixed by A. Peshkov

~ ~ ~

**(CORE-4756)** Events were broken in Firebird 3.

fixed by A. Peshkov

~ ~ ~

**(CORE-4646)** Attachment would fail when using LegacyAuth on the server and the default configuration on the client.

fixed by A. Peshkov

~ ~ ~

**(CORE-4639)** Linux servers could reject connections with a transliteration error.

fixed by D. Yemanov

~ ~ ~

**(CORE-4471)** The Legacy_Auth plugin would not connect from a FB3 to a FB2.5 server when tried after Win_Sspi.

fixed by A. Peshkov
Bugs Fixed

~ ~ ~

Utilities

isql

(CORE-4782) SHOW TABLE command in isql would fail when the table contained a field with Unicode collation in its DDL.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4706) isql was padding BLOB columns wrongly when the column alias had more than 17 characters.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4578) In isql, INPUT files were not being properly closed.

fixed by A. Peshkov

~ ~ ~

(CORE-4503) In isql, the command SHOW USERS would display only the current user.

fixed by A. Peshkov

~ ~ ~

(CORE-4259) The call to setlocale(LC_CTYPE, '') should not be made in isql for Windows.

fixed by F. Schlottmann-Goedde & D. Yemanov

~ ~ ~

gbak

(CORE-4715) Restore of a shadowed database would fail if the -k ("restore without shadow") switch was used.

fixed by A. Peshkov

~ ~ ~

(CORE-4704) Permissions for generators and exceptions were being corrupted after backup/restore.

fixed by D. Yemanov

~ ~ ~
**Bugs Fixed**

---

**gsec**

(CORE-4698) Typing `add -user SYSDBA -pw masterkey -admin yes` into `gsec` would crash firebird.exe.

fixed by V. Khorsun & A. Peshkov

---

**Firebird 3.0 Second Beta Release: Improvements**

The following improvements were implemented prior to the v.3.0.0 Beta 2 release:

(CORE-4791) INSERTING, UPDATING and DELETING were made reserved words to fix ambiguity with Boolean expressions.

implemented by A. dos Santos Fernandes

---

(CORE-4735) An expression ‘where bool_field IS true | false’ can now use the same index as ‘where bool_field = true | false’ if such an index exists.

implemented by A. dos Santos Fernandes

---

(CORE-4731) Issuing DML or DDL statements on the RDB$-- system tables is prohibited in Firebird 3.

implemented by A. Peshkov

---

(CORE-4729) A flag was added to MON$DATABASE to assist in determining what type of security database is used - default, self or other.

implemented by A. dos Santos Fernandes

---

(CORE-4696) Flushing dirty pages to disk after creation of a temporary table index is now avoided.

implemented by V. Khorsun

---

(CORE-4685) Resolved some risky issues on POSIX when aliases in `databases.conf` were pointing to symbolic or hard links, allowing invalid database accesses.

implemented by A. Peshkov

---

(CORE-4671) Internal temporary blobs are released early, to free up some memory\disk space.
implemented by V. Khorsun

(CORE-4610) Diagnostics will now report a tag name when transliteration errors occur in Parameter Block values.

implemented by A. Peshkov

(CORE-4607) Support added for having multiple UserManagers in firebird.conf and for using them from SQL.

implemented by A. Peshkov

(CORE-4605) The configuration parameter CryptPlugin was renamed to WireCryptPlugin.

implemented by A. Peshkov

(CORE-4590) The data type of the result returned by the functions CHAR_LENGTH(), BIT_LENGTH() and OCTET_LENGTH() for BLOBs was changed to BIGINT.

implemented by A. dos Santos Fernandes

(CORE-4386) “Object in use” errors now come with more details.

implemented by D. Yemanov

(CORE-3526) Support was added for WHEN SQLSTATE error handlers in PSQL.

implemented by D. Yemanov

(CORE-3234) Support was added for text BLOBs >= 32K as the first argument for the TRIM() function.

implemented by A. dos Santos Fernandes

(CORE-3226) IPv6 support for the remote interface.

implemented by M. Kubecek

(CORE-733) Compression of data over the network.

implemented by A. Peshkov
Firebird 3.0 First Beta Release

The following improvements and bug fixes were reported prior to the v.3.0.0 Beta 1 release:

**Core Engine**

*CORE-4576*  
The Cache Writer thread could not start.  
固定 by V. Khorsun

*CORE-4574*  
Regression: Incorrect result in subquery with aggregate.  
固定 by A. dos Santos Fernandes

*CORE-4570*  
ALTER PACKAGE was returning a wrong error.  
固定 by A. dos Santos Fernandes

*CORE-4566*  
Incorrect size of the output parameter or argument when EXECUTE BLOCK, procedure or function used a system field in the metadata character set.  
固定 by A. Peshkov

*CORE-4565*  
GDSCODE could have value = 0 in WHEN-section under some concurrent environments. This bug affected Superclassic and Classic models but not Superserver.  
固定 by V. Khorsun

*CORE-4555*  
A DDL trigger was remaining active after being dropped.  
固定 by A. dos Santos Fernandes

*CORE-4530*  
A DB_KEY based join of two tables could be ineffective.  
固定 by D. Yemanov

*CORE-4522*  
DDL permissions were not protecting against removal of BLOB filters.
Bugs Fixed

---

fixed by R. Simakov

---

(CORE-4515) Regression: trace was reporting UPDATEs in statistics when doing INSERT into <some_table>.

fixed by D. Yemanov

---

(CORE-4505) Use of a named cursor would fail if a statement was not executed.

fixed by A. Peshkov

---

(CORE-4488) A FOR SELECT \(<L>\) FROM \(<T>\) AS CURSOR \(<C>\) seems to return a wrong result if table \(<T>\) is modified inside the cursor's BEGIN...END block. Cursor references, which are not variables, should represent the current state of the record. If it was updated "in place" (via "where current of"), then cursor references should return the new values. The first example reported in CORE-4488 should return NULLs.

fixed by A. dos Santos Fernandes

---

(CORE-4478) Failure to load a provider or plugin was not reported anywhere.

fixed by A. Peshkov

---

(CORE-4477) The field RDB$MAP_TO_TYPE was missing from the system table RDB$TYPES.

fixed by A. Peshkov

---

(CORE-4468) CREATE USER GRANT ADMIN ROLE did not work.

fixed by A. Peshkov

---

(CORE-4464) Duplicate tags for CREATE/ALTER USER were not handled correctly.

fixed by A. Peshkov

---

(CORE-4453) Regression: The NOT NULL constraint, if declared in a domain, did not work.

fixed by A. dos Santos Fernandes

---

(CORE-4447) A positioned UPDATE statement would preclude its index usage for the subsequent cursor field references.
Bugs Fixed

fixed by D. Yemanov

Engine could hang and block all attachments in an out-of-disk-space condition during physical backup.

fixed by V. Khorsun

GlobalRWLock could not downgrade an EX lock to SH if readers were present.

fixed by A. Peshkov

After calling release() instead of detach() for an attachment to a database in embedded mode, the attachment would remain interminably active.

fixed by A. Peshkov

Properties of a user created in Legacy_UserManager were padded with spaces up to 10 characters.

fixed by A. dos Santos Fernandes

Pointless extraction of generic DDL trigger.

fixed by A. dos Santos Fernandes

A query executed via EXECUTE STATEMENT was returning the wrong result.

fixed by A. Peshkov

EXECUTE STATEMENT ON EXTERNAL was not finding a Firebird 2.5 database.

fixed by A. Peshkov

"Cursor not found error" when using the legacy API.

fixed by A. Peshkov

SELECT WITH LOCK could enter an infinite loop for a single record.

fixed by D. Yemanov
Run-time errors were returning incorrect line/column information.

fixed by A. dos Santos Fernandes

Explicit cursors containing correlated subqueries in the select list were performing poorly.

fixed by D. Yemanov

Preparation of an erroneous DDL statement was not indicating that the main command failed.

fixed by A. dos Santos Fernandes

A procedure would execute infinitely if it contained more than 32767 statements inside any BEGIN/END block.

fixed by D. Yemanov

Truncation error when using EXECUTE STATEMENT with a BLOB.

fixed by A. dos Santos Fernandes

Package compilation was not checking for duplicate names.

fixed by A. dos Santos Fernandes

Deadlock could occur when two data pages contained record fragments pointing to each other.

fixed by V. Khorsun

A CREATE FUNCTION or CREATE PROCEDURE statement that referred to a nonexistent exception would return the error message “Error while parsing function's BLR” instead of “exception not defined”.

fixed by D. Yemanov

A WHERE predicate containing NULL IS NOT DISTINCT FROM (select min(NULL) from ...) was returning the wrong result.

fixed by D. Yemanov
Bugs Fixed

~ ~ ~

(CORE-4365) Equality predicate distribution was not working for some complex queries.
fixed by D. Yemanov

~ ~ ~

(CORE-4360) SELECT from derived table which contains GROUP BY on a column with a literal value was returning wrong results.
fixed by D. Yemanov

~ ~ ~

(CORE-4354) Parsing of a recursive query would return the error “Column does not belong to referenced table” when the source table did have such a column.
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4353) Sorting records were larger than was really necessary.
fixed by D. Yemanov

~ ~ ~

(CORE-4344) Error “no current record for fetch operation” when table inner joins procedure inner joins table.
fixed by D. Yemanov

~ ~ ~

(CORE-4334) Resources (e.g. sort files) owned by a trigger could be left unreleased when the trigger was interrupted asynchronously.
fixed by D. Yemanov

~ ~ ~

(CORE-4331) LAG, LEAD and NTH_VALUE would raise an error when the second argument was NULL.
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4330) The function LAG returned an incorrect result if the OFFSET value was assigned from a table.
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4326) The keyword SET was required in the ALTER USER statement when it should have been optional.
fixed by A. Peshkov
Bugs Fixed

~ ~ ~

(CORE-4318) Regression: Predicates involving PSQL variables or parameters were not pushed inside the aggregation.

d fixed by D. Yemanov

~ ~ ~

(CORE-4313) Error “Attempt to reopen an open cursor” could be raised if the query handle was reused in a different transaction.

d fixed by D. Yemanov

~ ~ ~

(CORE-4309) The ‘Cache Writer’ record in MONSATTACHMENTS would vanish when deleting, via delete from MONSATTACHMENTS, another connection that was running a heavy update on a big table.

d fixed by V. Khorsun

~ ~ ~

(CORE-4307) Any fields present only in the WHERE clause of a view WITH CHECK OPTION would cause an invalid CHECK CONSTRAINT violation.

d fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4303) Possible races during service destruction.

d fixed by A. Peshkov

~ ~ ~

(CORE-4301) Non-ASCII data in SECSUSERS was not read correctly.

d fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4286) The error “Statement already has a cursor assigned” would be thrown when trying to execute another SQL statement using a different cursor name.

d fixed by A. Peshkov

~ ~ ~

(CORE-4118) Expression index might be overlooked for derived fields or view fields.

d fixed by D. Yemanov

~ ~ ~

(CORE-3305) A “BLOB not found” error would be returned after creating or altering an invalid trigger.

d fixed by D. Yemanov
Bugs Fixed

---

(CORE-2350) An over-long column name for a SELECT alias was not being rejected, as it should have been.
fixed by A. dos Santos Fernandes

---

(CORE-1475) A database which had active attachments could not be replaced from a backup file even after the database was shut down.
fixed by D. Yemanov

---

Server Crashes

(CORE-4575) The server would crash in the garbage collector thread at disconnect of the last attachment.
fixed by V. Khorsun

---

(CORE-4568) The server could crash while disconnecting from the database under load.
fixed by D. Yemanov

---

(CORE-4510) A database validation bug could cause the server to crash.
fixed by V. Khorsun

---

(CORE-4506) The server would crash when executing almost any "show ..." commands after a reconnect.
fixed by A. Peshkov

---

(CORE-4500) Firebird would crash after an unsuccessful remapping of the lock table's shared memory.
fixed by D. Yemanov

---

(CORE-4498) The server would crash when getting an explained plan for a DBKEY-based retrieval.
fixed by D. Yemanov

---

(CORE-4422) The server would crash when using ROW_NUMBER()over(PARTITION BY x) in an ORDER by clause.
Bugs Fixed

fixed by D. Yemanov

(CORE-4419) The server could crash while sorting records longer than 128KB.

fixed by D. Yemanov

(CORE-4322) The engine would crash when aggregate or window functions were used in a recursive query.

fixed by A. dos Santos Fernandes

(CORE-4321) Regression: isql was not destroying the SQL statement.

fixed by A. Peshkov

(CORE-4319) The engine would crash when the Trace config contained the line “connection_id=NN” and an attempt was made to connect to a non-existent database or alias.

fixed by V. Khorsun

(CORE-4304) The engine would crash when an attempt to REcreate a table with a foreign key was made after a syntax error that preceded the RECREATE attempt.

fixed by A. dos Santos Fernandes

API/Remote Interface

(CORE-4275) CREATE DATABASE would fault if fbclient.dll was loaded from another directory (Providers = Engine12).

fixed by V. Khorsun

Utilities

isql

(CORE-4480) isql would issue the warning “Bad debug info format” when connecting to a database with stored functions after a restore.

fixed by D. Yemanov
Bugs Fixed

~ ~ ~

(CORE-4440)  `isql` would crash without connecting when executing the command SHOW VERSION.
fixed by A. Peshkov
~ ~ ~

(CORE-4380)  `isql` would truncate a BLOB when reading an empty segment.
fixed by A. dos Santos Fernandes
~ ~ ~

(CORE-4320)  Regression: `isql` would crash when receiving statistics from the execution of a SQL query.
fixed by V. Khorsun
~ ~ ~

**gbak**

(CORE-4470)  `gbak` restore would fail on a database containing dependency between views and packaged functions.
fixed by A. dos Santos Fernandes
~ ~ ~

(CORE-4425)  User collations based on UNICODE were not being upgraded to a newer ICU version on restore.
fixed by A. dos Santos Fernandes
~ ~ ~

(CORE-4417)  `gbak` refused to commit the index for a primary key with characters accented with an umlaut.
fixed by A. dos Santos Fernandes
~ ~ ~

(CORE-4346)  V.3 `gbak` was unable to restore backups made on earlier server versions.
fixed by D. Yemanov
~ ~ ~

**nBackup**

(CORE-4461)  `nBackup` was printing error messages to `stdout` instead of `stderr`.
fixed by A. Peshkov
~ ~ ~
Bugs Fixed

**qli**

(CORE-4327) qli was throwing an error when copying NULL blobs between databases.

fixed by A. Peshkov

~ ~ ~

**Firebird 3.0 Second Alpha Release**

The following improvements and bug fixes were reported as fixed prior to the v.3.0.0 release:

**Core Engine**

(CORE-4302) Descending index could be very inefficient for some keys

fixed by V. Khorsun

~ ~ ~

(CORE-4289) A NOT NULL field from a derived table could become NULL when referred to from outside the derived table

fixed by D. Yemanov

~ ~ ~

(CORE-4281) TYPE OF arguments of stored functions could cause the server to hang if depending on a domain or column that had been changed

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4270) A subquery involving a windowed function and a \texttt{where <field> IN(select ...)} condition could cause an error

fixed by D. Yemanov

~ ~ ~

(CORE-4265) An unexpected lock conflict error could be raised while connecting to a heavily loaded database

fixed by D. Yemanov

~ ~ ~

(CORE-4262) Context parsing errors could occur with derived tables and CASE functions

fixed by D. Yemanov
~ ~ ~

(CORE-4261) JOIN result could be wrong when joined fields had been created via the `row_number()` function
fixed by D. Yemanov

~ ~ ~

(CORE-4258) The boundary for the minimum value for BIGINT/DECIMAL(18) was wrong
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4251) The Guardian service could write garbage after the end of a message in the Event Log
fixed by V. Khorsun

~ ~ ~

(CORE-4250) Access violation could occur in Guardian at process shutdown
fixed by V. Khorsun

~ ~ ~

(CORE-4237) Metadata being reported from system table queries for UDF return arguments was different to that returned in Firebird 2.5
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4234) IF (subfunc()) would throw an error when subfunc returned a Boolean
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4229) Bidirectional cursor was not being positioned by the first call of FETCH LAST
fixed by D. Yemanov

~ ~ ~

(CORE-4227) A parser conflict was causing wrong evaluation of BETWEEN and Boolean expressions
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4216) Memory leak with TRIGGER ON TRANSACTION COMMIT
fixed by V. Khorsun

~ ~ ~
The embedded engine would hang for 5 seconds when closing, with errors about timeout in shutdown process and invalid mutex being written into firebird.log

fixed by A. Peshkov

A computed field would return NULL inside a BEFORE INSERT trigger

fixed by D. Yemanov

An incorrect “token unknown” error would occur when an SQL string ended with a hex number literal

fixed by A. dos Santos Fernandes

Some Boolean expressions were not being allowed

fixed by A. dos Santos Fernandes

Incorrect memory statistics were being reported

fixed by D. Yemanov

RDB$GET_CONTEXT/RDB$SET_CONTEXT parameters were being described incorrectly as CHAR NOT NULL instead of VARCHAR NULLABLE

fixed by A. dos Santos Fernandes

Bad performance and slow response were exhibited when many concurrent sorts were executed

fixed by D. Yemanov

With bugcheckabort=1 and sweep starting at gap ~21000, “Bugcheck 186 (record disappeared)” and 100% CPU load would occur

fixed by V. Khorsun

Unnecessary index reads could occur when using a strict inequality condition

fixed by V. Khorsun
Server Crashes

(\texttt{CORE-4293}) The server could crash on a SELECT with a long or complex list of compound AND/OR'd predicates

\textit{fixed by D. Yemanov}

\textendash \textendash \textendash

(\texttt{CORE-4271}) Recreation of an errant package body could cause the engine to crash

\textit{fixed by A. dos Santos Fernandes}

\textendash \textendash \textendash

(\texttt{CORE-4268}) Disconnecting from a database could cause a server crash

\textit{fixed by D. Yemanov}

\textendash \textendash \textendash

(\texttt{CORE-4267}) Sweeping a database could cause a server crash

\textit{fixed by D. Yemanov}

\textendash \textendash \textendash

(\texttt{CORE-4225}) The server could crash when trace activity was attempted on a database having a database-level trigger

\textit{fixed by V. Khorsun}

\textendash \textendash \textendash

(\texttt{CORE-4185}) Server crashes could occur, reporting “invalid lock id (NNNNN)”

\textit{fixed by A. Peshkov}

\textendash \textendash \textendash

API/Remote Interface

(\texttt{CORE-4283}) “Resource temporarily unavailable” errors could occur while events were being registered simultaneously

\textit{fixed by A. Peshkov}

\textendash \textendash \textendash

(\texttt{CORE-4236}) Database shutdown was being reported as successfully completed before all active connections had actually been interrupted

\textit{fixed by D. Yemanov}
Bugs Fixed

(CORE-4178) The new API was still returning obscure historical definition artifacts of data fields, instead of proper metadata properties that would make the interface actually usable
fixed by A. Peshkov

(CORE-4162) Warnings were not being returned from calls to attachDatabase()
fixed by A. Peshkov

Security/User Management

(CORE-4241) Log-in could succeed with an empty password
fixed by A. Peshkov

(CORE-4200) An uncommitted SELECT from the pseudo table sec$users would block new database connections
fixed by A. Peshkov

Procedural Language

(CORE-4247) Positioned DELETE (WHERE CURRENT OF <CURSOR>) could fail for tables with newly added fields
fixed by A. dos Santos Fernandes

(CORE-4244) Creating a procedure could be a problem if it involved adding text in DOS864 character set
fixed by A. dos Santos Fernandes

(CORE-4184) An error would be raised while executing an empty EXECUTE BLOCK with NOT NULL output parameter
fixed by A. dos Santos Fernandes

(CORE-4160) A parameterized exception would mishandle non-ASCII characters passed as the parameter
fixed by A. dos Santos Fernandes
Bugs Fixed

~ ~ ~

(CORE-4145) Preparing an EXECUTE BLOCK that used domains was causing a memory leak
fixed by A. dos Santos Fernandes

~ ~ ~

Data Definition Language

(CORE-4214) Global temporary tables were able to reference permanent relations, which they should not be able to do
fixed by V. Khorsun

~ ~ ~

(CORE-4212) Dropping a foreign key on a Global temporary table would cause a server crash
fixed by V. Khorsun

~ ~ ~

(CORE-4203) Packaged routines with CHAR or VARCHAR parameters could not be created
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4180) CREATE COLLATION was not verifying the base collation character set
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4173) Setting a generator value twice in a single transaction would set it to zero
fixed by D. Yemanov

~ ~ ~

(CORE-4155) External routines DDL in packages was wrongly reporting termination with semi-colon as an error
fixed by A. dos Santos Fernandes

~ ~ ~

Data Manipulation Language

(CORE-4269) Wrong output would be produced when a field with the result of a windowed function was used in a query with a useless WHERE 0=0 and GROUP BY clause
fixed by D. Yemanov

~ ~ ~
Parameterized queries using RDBSDB_KEY would not work
fixed by D. Yemanov

Recursive query would return incorrect results if passage through more than one branch was requested
fixed by D. Yemanov

LIKE with escape was not working
fixed by A. dos Santos Fernandes

Utilities

gfix

gfix would crash when the size of the description of a limbo transaction was larger than 1 KB
fixed by V. Khorsun

fbsvcmgr

fbsvcmgr was not recognising sts_record_versions and other sts switches
fixed by A. Peshkov

isql

Bug in the isql command setlocale(LC_CTYPE, "") on Windows due to a reference to editline, which is not available on that platform
fixed by F. Schlottmann-Goedde

ISQL -x was failing to output the START WITH clause of generators/sequences
fixed by A. dos Santos Fernandes
Bugs Fixed

( CORE-4149 ) New permission types were not being displayed by *isql*
fixed by D. Yemanov

( CORE-362 ) It was impossible to enter certain characters in *isql*
fixed by F. Schlottmann-Goedde

---

**gbak**

( CORE-4202 ) Backup/restore from an older version to v.3.0 would fail with a BLR error
fixed by D. Yemanov

---

( CORE-4168 ) A backup containing procedures or triggers that selected from external tables could not be restored with *ExternalFileAccess = None*
fixed by D. Yemanov

---

( CORE-4164 ) Owner name was missing for generators and exceptions restored from a backup
fixed by D. Yemanov

---

**nbackup**

( CORE-2648 ) *nBackup's* delta file was ignoring the Forced Writes setting of the database
fixed by V. Khorsun

---

**Database Monitoring (MON$)**

( CORE-4235 ) Deadlock could occur while accessing the monitoring tables under concurrent load
fixed by D. Yemanov & V. Khorsun

---

( CORE-4176 ) Monitoring tables were returning incomplete information in Classic and Superclassic configurations
Bugs Fixed

fixed by D. Yemanov

~ ~ ~

Trace

(CORE-4219) Regular expressions with double-slash would fail in trace

fixed by A. Peshkov

~ ~ ~

(CORE-4163) Configuration file fbtrace.conf contained syntax errors

fixed by A. Peshkov

~ ~ ~

International Language Support

Installation Issues

(CORE-4153) Attempting to use Legacy_Auth directly after install would not work without restarting the service

fixed by P. Reeves

~ ~ ~

Firebird 3.0 First Alpha Release

The following improvements and bug fixes were reported as fixed prior to the v.3.0.0 release:

Core Engine

(CORE-4135) Sweep was blocking the establishment of concurrent attachments in Superserver.

fixed by V. Khorsun

~ ~ ~

(CORE-4134) A race condition could occur when auto-sweep was started.

fixed by V. Khorsun

~ ~ ~

(CORE-4074) COMPUTED BY columns and POSITION function could produce garbled results.
Bugs Fixed

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-4027) Creating a table with computed fields containing SELECT FIRST could produce a corrupted result.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3973) The SQLDA for an aliased column in a grouped query was missing the original table name, column name and owner.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3947) Wrong results were produced when a column in the WHERE clause used the collation option (NUMERIC-SORT=1).

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3941) A unique expression index would exhibit a memory alignment problem.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3929) The invalid error “attempted update of read-only column” would appear when selecting MINVALUE from a list of more than 255 elements.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3894) When an attempt was made to reduce the size of a CHAR or VARCHAR column, the numbers delivered in the error message were incorrect.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3874) A computed column would appear in non-existent rows output from a left join.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3820) Some character sets were duplicated in the system table RDB$TYPES.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3754) SIMILAR TO was not working correctly.
fixed by A. dos Santos Fernandes

(CORE-3735) An unprivileged user could delete from the system tables RDB$DATABASE, RDB$COL-
LATIONS and RDB$CHARACTER_SETS.

fixed by D. Yemanov

(CORE-3694) “Internal consistency check” would occur in a query with grouping by subquery+stored
procedure+aggregate.

fixed by A. dos Santos Fernandes

(CORE-3672) It was not possible to use the SUBSTRING function to create a computed index for large
columns.

fixed by A. dos Santos Fernandes

(CORE-3638) Some collation tweaking: FR_CA_CI_AI collation was introduced; FR_FR was changed
to be identical to FR_CA and FR_FR_CI_AI was changed to be identical to the new FR_CA_CI_AI.

fixed by A. dos Santos Fernandes

(CORE-3476) The LIST function was concatenating binary blobs as though they were text.

fixed by A. dos Santos Fernandes

(CORE-3401) Collation errors could occur with the use of [type of] <domain> and type of <column>.

fixed by A. dos Santos Fernandes

(CORE-3373) It was possible to store a string of length 31 characters into a VARCHAR(25) column.

fixed by A. dos Santos Fernandes

(CORE-3338) Regression: Code changes had disabled support for expression indexes with COALESCE,
CASE and DECODE.

fixed by A. dos Santos Fernandes

(CORE-3317) Success of row deletion depended on the order of insertion of the rows.
Bugs Fixed

fixed by V. Khorsun

   ~ ~ ~

(CORE-3310)   A complex expression involving RDB$GET_CONTEXT and BETWEEN worked in DSQL
   but failed with a conversion error when selected in a view definition.

fixed by A. dos Santos Fernandes

   ~ ~ ~

(CORE-3260)   Interlock.h was not portable.

fixed by A. Peshkov

   ~ ~ ~

(CORE-3250)   The Firebird server could not be started under any user name other than “root”, “firebird”,
   “interbas” or “interbase”.

fixed by A. Peshkov

   ~ ~ ~

(CORE-3239)   The collation UTF8 UNICODE_CI could not be used in a compound index.

fixed by A. dos Santos Fernandes

   ~ ~ ~

(CORE-3204)   A constraint violation error involving CAST was not being raised inside views.

fixed by A. dos Santos Fernandes

   ~ ~ ~

(CORE-3052)   Comparisons involving multiple index segments could produce wrong result sets.

fixed by A. dos Santos Fernandes

   ~ ~ ~

(CORE-2988)   The concurrent transaction number was not being reported when a lock timeout occurred.

fixed by N. Samofatov

   ~ ~ ~

(CORE-2957)   COUNT(*) from a big table could return a negative result.

fixed by D. Yemanov

   ~ ~ ~

(CORE-2952)   Character class names in SIMILAR TO expressions could be case-sensitive or case-
   insensitive, depending on the collation of the left part, whereas they should be unequivocally case-insensitive.

fixed by D. Sibiryakov
An ALTER TABLE..ALTER COLUMN..ALTER POSITION operation could result in wrong column positions.

fixed by A. dos Santos Fernandes

The character set used in a constant was not being registered as a dependency.

fixed by A. dos Santos Fernandes

COLLATE expressions were being applied incorrectly.

fixed by A. dos Santos Fernandes

Plan output lacked the names of views when selecting from views that contained procedure calls.

fixed by D. Yemanov

DB_KEY was always zero for rows in external tables.

fixed by D. Yemanov

A full outer join could not use available indices, resulting in very slow execution sometimes.

fixed by D. Yemanov

Use of certain choices of character in double-quoted index names, for example a bracket character, could defeat the parsing logic when generating a human-readable plan.

fixed by D. Yemanov

A join of a stored procedure with a view or a table could fail with the error “No current record for fetch operation”.

fixed by D. Yemanov

A buffer overrun error was being caught erroneously in a DOUBLE PRECISION to VARCHAR conversion in a Dialect 1 database.

fixed by C. Valderrama C.
Bugs Fixed

~ ~ ~

(CORE-1605) An aggregated query was causing “Bugcheck 232 (invalid operation)”.  
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1550) An unnecessary index scan was executed when the same index is mapped to both WHERE and ORDER BY clauses.  
fixed by D. Yemanov

~ ~ ~

API/Remote Interface

(CORE-3718) The client library could hang after an unsuccessful attempt to connect to the remote auxiliary (events) port.  
fixed by A. Peshkov

~ ~ ~

(CORE-3475) Parameters inside the CAST function were being wrongly described in the SQLDA as non-nullable.  
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3269) The client would perform detach incorrectly when the server became unavailable.  
fixed by A. Peshkov

~ ~ ~

(CORE-2484) An erroneous “Success” message would be returned in the error status vector when failing to connect to a trash database file.  
fixed by C. Valderrama C.

~ ~ ~

(CORE-2431) String values in error messages were not converted to the connection character set.  
fixed by A. dos Santos Fernandes

~ ~ ~

Procedural Language

(CORE-4018) Use of a system domain in declarations of arguments or return values in a stored procedure could prevent the procedure from being modifiable.
fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3737) EXECUTE BLOCK parameter definitions were not being respected and could cause wrong behavior with respect to character sets.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3545) Validation of domain CHECK constraints when used in PSQL declarations was inconsistent: it was using the type of the expression, instead of the type of the variable.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3055) The names of variables or arguments could be wrong or absent in error messages when more than 256 variables were used.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3047) Resolution of EXECUTE BLOCK parameter collations was using wrong logic.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2204) Constraints on stored procedure output parameters were checked even when the procedure returned no rows.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1620) Incorrect error message (an absurd column number) was returned if an empty SQL string was prepared for EXECUTE STATEMENT.

fixed by D. Yemanov

~ ~ ~

Data Definition Language

(CORE-3114) Attempting to drop a non-existent generator (sequence) would result in a serious exception.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3056) Problems could occur if further DDL commands were issued in the same transaction following a CREATE COLLATION command.
Bugs Fixed

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2696) The ALTER TABLE command allowed the addition of a column with a NOT NULL definition, allowing a non-savvy DBAdmin to wreck the table.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1748) Unrestorable backup: a problem which would occur if ALTER TABLE...ADD COLUMN added a column with a NOT NULL constraint. The fix for CORE-2696 has now made it impossible to do this.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1518) Adding a non-nullable column to a populated table would render the table inconsistent. The fix for CORE-2696 has now made it impossible to do this.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-1355) Client tools tended to be confused about how to interpret a NULL that is returned from a non-nullable column. The fix for CORE-2696 has now made it impossible to add a non-nullable column to a populated table.

It is not clear, though, whether this part of the fix makes it mandatory to specify a default value for a non-nullable column.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-634) Bad behaviour of DELETE when the WHERE clause was a subquery involving FIRST/SKIP: the operation would zap every row in the table.

fixed by V. Khorsun

~ ~ ~

(CORE-304) Any user could alter or drop generators and exceptions#a metadata security hole.

fixed by D. Yemanov

~ ~ ~

Data Manipulation Language & DSQL

(CORE-4144) When when preparing a query with UNION, the error “context already in use (BLR error)” was wrongly being thrown.

fixed by V. Khorsun
Bugs Fixed

~ ~ ~

(CORE-4005) Recursive CTEs were returning a wrong error message.

fixed by V. Khorsun

~ ~ ~

(CORE-3416) Inserting a word containing the 8-bit character ‘ä’ into a CHARACTER SET ASCII column would succeed instead of throwing a transliteration error.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3201) The internal function ATAN2 was returning an incorrect value with arguments (0, 0).

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-3174) An expression index involving TRIM could lead to an incorrect indexed lookup.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2699) A common table expression context could be used with parameters.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2606) A multi-byte CHAR value requested as VARCHAR was returned with padded spaces.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-2238) With UTF8 and large varchar fields, IS DISTINCT FROM would cause the error “Implementation limit exceeded”.

fixed by D. Yemanov

~ ~ ~

(CORE-1188) STARTING WITH ? (where the parameter value supplied is an empty string) would fail if the plan used a compound index.

fixed by A. dos Santos Fernandes

~ ~ ~

(CORE-92) Infinite insertion cycle: INSERT INTO THIS_TABLE SELECT ... FROM THIS_TABLE would loop forever until resources were exhausted.

fixed by V. Khorsun
Command-line Utilities

(CORE-2547) Utilities did not always honour the minimum number of characters required to recognise an option.

fixed by C. Valderrama C.

Other old bugs that were fixed in utilities:

FbGuard

(CORE-2784) Guardian would keep creating more and more threads each time FBServer died.

fixed by C. Valderrama C.

(FIREBIRD-1595) Firebird Guardian's tray icon would disappear after a Windows Explorer crash.

fixed by C. Valderrama C.

isql

(CORE-4137) isql was generating metadata script output with syntax errors in the CHARACTER SET clause, e.g., CHARACTER SET ISO8859_1.

fixed by A. dos Santos Fernandes

(CORE-3431) isql was padding UTF-8 data incorrectly.

fixed by A. dos Santos Fernandes

(CORE-2788) isql would extract the array dimensions after the character set name.

fixed by C. Valderrama C.

gbak

(CORE-3575) gbak did not support backup volumes of size greater than 4GB.
Bugs Fixed

fixed by A. Peshkov

~ ~ ~

(CORE-2740)  gbak would restore invalid views without any warning to the user.

fixed by C. Valderrama C.

~ ~ ~

(CORE-2545)  Several validations were lacking in gbak.

fixed by C. Valderrama C.

~ ~ ~

nbackup

(CORE-2543)  nbackup could hide the real cause of a failure.

fixed by C. Valderrama C.

~ ~ ~

International Language Support

(CORE-4136)  The “Sharp-S” character was being treated incorrectly in the UNICODE_CI_AI collation.

fixed by A. dos Santos Fernandes

~ ~ ~
# Chapter 14

## Firebird 3.0 Project Teams

### Table 14.1. 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>Alex Peshkov</td>
<td>Russian Federation</td>
<td>Full-time security features coordinator; buildmaster; porting authority</td>
</tr>
<tr>
<td>Claudio Valderrama</td>
<td>Chile</td>
<td>Code scrutineer; bug-finder and fixer; ISQL enhancements; UDF fixer, designer and implementor</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>Paul Beach</td>
<td>France</td>
<td>Release Manager; HP-UX builds; MacOS Builds; Solaris Builds</td>
</tr>
<tr>
<td>Pavel Cisar</td>
<td>Czech Republic</td>
<td>QA tools designer/coordinator</td>
</tr>
<tr>
<td>Philippe Makowski</td>
<td>France</td>
<td>QA tester</td>
</tr>
<tr>
<td>Paul Reeves</td>
<td>France</td>
<td>Win32 installers and builds</td>
</tr>
<tr>
<td>Mark Rotteveel</td>
<td>The Netherlands</td>
<td>Jaybird implementor and co-coordinator</td>
</tr>
<tr>
<td>Jiri Cincura</td>
<td>Czech Republic</td>
<td>Developer and coordinator of .NET providers</td>
</tr>
<tr>
<td>Alexander Potapchenko</td>
<td>Russian Federation</td>
<td>Developer and coordinator of ODBC/JDBC driver for Firebird</td>
</tr>
<tr>
<td>Stephen Boyd</td>
<td>Canada</td>
<td>GPRE contributions</td>
</tr>
<tr>
<td>Alexey Kovyazin</td>
<td>Russian Federation</td>
<td>Website coordinator</td>
</tr>
<tr>
<td>Paul Vinkenoog</td>
<td>The Netherlands</td>
<td>Coordinator, Firebird documentation project; documentation writer and tools developer/implementor</td>
</tr>
<tr>
<td>Norman Dunbar</td>
<td>U.K.</td>
<td>Documentation writer</td>
</tr>
<tr>
<td>Developer</td>
<td>Country</td>
<td>Major Tasks</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Pavel Menshchikov</td>
<td>Russian Federation</td>
<td>Documentation translator</td>
</tr>
<tr>
<td>Tomneko Hayashi</td>
<td>Japan</td>
<td>Documentation translator</td>
</tr>
<tr>
<td>Umberto (Mimmo) Masotti</td>
<td>Italy</td>
<td>Documentation translator</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 http://www.firebirdsql.org/pdfmanual/pdl.pdf (PDF) and http://www.firebirdsql.org/manual/pdl.html (HTML).

The Original Documentation is entitled Firebird 3.0 Release Notes.

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

Copyright (C) 2004-2015. All Rights Reserved. Initial Writer contact: helebor at users dot sourceforge dot net.