Administrative features

Certain improvements to Firebird's administrative features will be welcomed by many.

New RDB$ADMIN system role

Alex Peshkov

A new pre-defined system role RDB$ADMIN has been added for transferring SYSDBA privileges to another user. Any user, when granted the role in a particular database, acquires SYSDBA-like rights when attaching to that database with the RDB$ADMIN role specified.

To assign it, SYSDBA should log in to that database and grant the role RDB$ADMIN to the user, in the same way one would grant any other role to a user. After the user has been granted the role, s/he must include it in the log-in in order to access the “superuser” privileges in that database.

Important: If the user attaches with a user database role passed in the DPB (connection parameters), it will not be replaced with RDB$ADMIN, i.e., he/she will not get SYSDBA rights.

The following example transfers SYSDBA privileges to users named User1 and Admins\ADMINS. The second user in our example is typical for a Windows system user with access enabled via trusted authentication:

```
GRANT RDB$ADMIN TO User1;
GRANT RDB$ADMIN TO "Admins\ADMINS";
```

Multiple databases and Superusers

It should be understood that acquiring the RDB$ADMIN role does not make a regular user into SYSDBA. Rather, it gives that user the same privileges as SYSDBA over the objects *in the database in which the role is granted to that user*.

- If the same user needs Superuser privileges in more than one database, the RDB$ADMIN role must be explicitly granted for that user in each database.
- If more than one user is to have Superuser privileges in a database then each user needs to be granted the RDB$ADMIN role.
- One user that has acquired the RDB$ADMIN role in the database can grant it to another user.
- It is not necessary to specify WITH ADMIN OPTION (for the privilege to grant this role to others) or WITH GRANT OPTION (for the privilege to grant permissions on objects to other users without being the owner of those objects). The ADMIN and GRANT options are implicit.

System "Superusers"

On POSIX hosts, the root user always had SYSDBA privileges, but the same was not possible for a domain administrator on Windows until Firebird 2.1. In v.2.1, a configuration parameter, Authentication, was introduced, whereby a user logged in as a Windows domain administrator could

automatically gain server access with SYSDBA privileges through trusted user authentication. On POSIX, the mechanism has not changed but on Windows, the introduction of the RDB$ADMIN role has changed the way Windows administrators acquire SYSDBA privileges.

Windows trusted user authentication is no longer available by default!

By default, the Authentication parameter in firebird.conf is configured as native. It must be explicitly configured to either trusted or mixed to enable trusted user authentication.

back to top of page

Global admin privileges for Windows administrators

For a trusted domain administrator to get SYSDBA access privileges on a Firebird server that is configured for trusted user authentication, the domain administrator must acquire the RDB$ADMIN role. The manual method described above for ordinary users will work, by granting RDB$ADMIN to each specific administrator, database by database.

However, there is a way for the SYSDBA to configure the server so that the RDB$ADMIN role will be mapped to administrators automatically when they log into any database, thus arriving at a situation parallel to the association of root-privileged users on POSIX systems with the SYSDBA privileges. The new ALTER ROLE statement achieves this (and only this) purpose.

ALTER ROLE statement

To configure a database to map the RDB$ADMIN role to administrators automatically on a Windows server that is configured to enable trusted user authentication, the SYSDBA logs in to any database and issues the following statement:

```
ALTER ROLE RDB$ADMIN
SET AUTO ADMIN MAPPING;
```

To revert to the default setting, preventing administrators from getting SYSDBA privileges automatically, issue this statement:

```
ALTER ROLE RDB$ADMIN
DROP AUTO ADMIN MAPPING;
```

back to top of page

Services API tag items

The same effects are supported in the Services API by the provision of two tag items: isc_action_svc_set_mapping to enable the automatic mapping and isc_action_svc_drop_mapping to disable it.

These tags are supported in the fbsvcmgr utility.

Escalating RDB$ADMIN scope for user management

The new DDL command ALTER USER enables an “ordinary” user (a regular Firebird user, a non-root user on POSIX or a trusted user on a Windows system where trusted authentication is enabled) the ability to change his or her password and/or personal name elements, while logged in to any database. Superusers can also use the same command to create and drop users. For more information about this new command, refer to the topic CREATE/ALTER/DROP USER in the chapter Data Definition Language.

Because security2.fdb is created as (or should be upgraded to) an ODS 11.2 database, it has the pre-defined RDB$ADMIN role, too. Since no user - not even SYSDBA - can log into the security database, alternative means have been provided to enable the SYSDBA or a Superuser to apply the RDB$ADMIN role in security2.fdb to an ordinary user that needs the ability to create or drop users. There are three ways, each having the equivalent effect:

1. Use the optional parameter GRANT ADMIN ROLE with a CREATE USER or ALTER USER statement.

Notes: Notice that GRANT ADMIN ROLE and REVOKE ADMIN ROLE here are not GRANT/REVOKE statements but 3-keyword parameters to the CREATE USER and ALTER USER statements. There is no system role named 'ADMIN'.

Any user that acquires the RDB$ADMIN role in a database implicitly acquires the extended privileges WITH ADMIN OPTION and WITH GRANT OPTION.

Examples

To grant the RDB$ADMIN role to user alex in the security database:

```
ALTER USER alex GRANT ADMIN ROLE;
```

To revoke the RDB$ADMIN role from user alex in the security database:

```
ALTER USER alex REVOKE ADMIN ROLE;
```

To drop user alex and destroy his privileges in all databases:

```
DROP USER alex;
```

2. Use the gsec utility with the new switch -admin. The switch takes one argument: YES to apply the RDB$ADMIN role to the user, or NO to revoke it. For more details, refer to the topic Granting the RDB$ADMIN role to an ordinary user in the gsec section of the Utilities chapter.

3. Use the new SPB parameter isc_spb_sec_admin which implements the assignment of the RDB$ADMIN role for ordinary users in security2.fdb via a SPB connection. It is described in more detail in the chapter, Changes to the Firebird API and ODS, in the topic Parameter isc_spb_sec_admin.

The fbsvmgr utility also supports the use of this parameter.

Tip: Firebird 2.5 does not allow you to set up more than one security database on a server. From v.3.0, it is intended to be possible to have separate security databases for each database. For now,

you can be connected to any database on the server (even employee.fdb) to update its one-and-only security2.fdb.

In future, it will be essential to send these requests from a database that is associated with the security database that is to be affected by them.

back to top of page

Trace and audit services

Vlad Khorsun

The new trace and audit facilities in v.2.5 were initially developed from the TraceAPI contributed to us by Nickolay Samofatov that he had developed for the Red Soft Database, a commercial product based on Firebird's code.

Overview of features

The new trace and audit facilities enable various events performed inside the engine, such as statement execution, connections, disconnections, etc., to be logged and collated for real-time analysis of the corresponding performance characteristics.

A trace takes place in the context of a *trace session*. Each trace session has its own configuration, state and output.

The Firebird engine has a fixed list of events it can trace. It can perform two different sort of traces: a system audit and a user trace. How the engine forms the list of events for a session depends on which sort of trace is requested.

Tip: Every trace session is assigned a unique session ID. When any trace session begins, the Services Manager outputs this ID as the message

```
Trace session ID nnnn started
```

where nnnn is the ID, of course.

The system audit session

A system audit session is started by the engine itself. To determine which events the session is “interested in”, it reads the contents of a *trace configuration file* as it goes to create the session.

A new parameter in firebird.conf, AuditTraceConfigFile points to the name and location of the file. There can be at most one system audit trace in progress. By default, the value of this parameter is empty, indicating that no system audit tracing is configured.

A configuration file contains a list of traced events and points to the placement of the trace log(s) for each event. It is sufficiently flexible to allow different sets of events for different databases to be

logged to separate log files. The template file file fbtrace.conf, found in Firebird's root directory, contains the full list of available events, with format, rules and syntax for composing an audit trace configuration file.

Tip about the fbtrace.conf file:

The file contains a large amount of commented text explaining the purpose and syntax of each entry. As subreleases progress, keep an eye on new events and facilities that will be added from time to time to improve the tracing capabilities.

For example, a late pre-release enhancement enables Services events to be traced by name and targeted using include and exclude filters.

As another example, the matching algorithm for path names was improved to interpret strings in the configuration file according to the platform character set and, basing the strings on UTF-8, to apply platform rules such as treating filenames as case-insensitive on Windows and case-sensitive on POSIX. (Tracker reference CORE-2404, A. dos Santos Fernandes).

An improvement added in v.2.5.2 was the ability to configure a trace session to log details of both user and automatic sweeps. Search the template file for the SWEEP_ options.

back to top of page

User trace sessions

A user trace session is managed by the user, using some new calls to the Services API. There are five new service functions for this purpose:

- **start:** isc_action_svc_trace_start
- **stop:** isc_action_svc_trace_stop
- **suspend:** isc_action_svc_trace_suspend
- **resume:** isc_action_svc_trace_resume
- **list all known trace sessions:** isc_action_svc_trace_list

The syntax for the Services API calls are discussed in the topic New trace functions for applications in the chapter entitled Changes to the Firebird API and ODS.

Workings of a user trace session

When a user application starts a trace session, it sets a session name (optional) and the session configuration (mandatory). The session configuration is a text file conforming to the rules and syntax modelled in the fbtrace.conf template that is in Firebird's root directory, apart from the lines relating to placement of the output.

Note: Such files obviously do not live on the server. It will be the job of the application developer to design a suitable mechanism for storing and retrieving texts for passing in the user trace request.

For example, the command-line fbsvcmgr utility supports a saved-file parameter, trc_cfg.

The output of a user session is stored in set of temporary files, each of 1 MB. Once a file has been

completely read by the application, it is automatically deleted. By default, the maximum total size of the output is limited to 10 MB. It can be changed to a smaller or larger value using the MaxUserTraceLogSize in firebird.conf.

Once the user trace session service has been started by the application, the application has to read its output, using calls to isc_service_query(). The service could be generating output faster than the application can read it. If the total size of the output reaches the MaxUserTraceLogSize limit, the engine automatically suspends the trace session. Once the application has finished reading a file (a 1 MB part of the output) that file is deleted, capacity is returned and the engine resumes the trace session automatically.

At the point where the application decides to stop its trace session, it simply requests a *detach* from the service. Alternatively, the application can use the isc_action_svc_trace_* functions to suspend, resume or stop the trace session at will.

Tip: The name of the character set of the attachment is included in any corresponding trace log records, placed between user:role and protocol:port, e.g.,

```
A.FDB (ATT_36, SYSDBA:NONE, WIN1251, TCPv4:127.0.0.1)
```

If no character set is specified in the DPB, NONE is written to the attachment character set slot in the trace log record.

(CORE-3008)

back to top of page

Who can manage trace sessions?

Any user can initiate and manage a trace session. An ordinary user can request a trace only on its own connections and cannot manage trace sessions started by any other users. Administrators can manage any trace session.

Abnormal endings

If all Firebird processes are stopped, no user trace sessions are preserved. What this means is that if a Superserver or Superclassic process is shut down, any user trace sessions that were in progress, including any that were awaiting a resume condition, are fully stopped and a resume cannot restart them.

Note: This situation doesn't apply to the Classic server, of course, since each connection involves its own dedicated server instance. Thus, there is no such thing as “shutting down” a Classic server instance. No service instance can outlive the connection that instigated it.

User trace sample configuration texts

The following samples provide a reference for composing configuration texts for user trace sessions.

a. Trace prepare, free and execution of all statements within connection 12345:

```
<database mydatabase.fdb>
 enabled               true
 connection_id         12345
 log_statement_prepare true
 log_statement_free    true
 log_statement_start   true
 log_statement_finish  true
 time_threshold 0
</database>
```

b. Trace all connections of given user to database mydatabase.fdb, logging executed INSERT, UPDATE and DELETE statements and nested calls to procedures and triggers, and show corresponding PLANs and performance statistics.

```
<database mydatabase.fdb>
 enabled                true
 include_filter         %(INSERT|UPDATE|DELETE)%
 log_statement_finish   true
 log_procedure_finish   true
 log_trigger_finish     true
 print_plan             true
 print_perf             true
 time_threshold         0
</database>
```

back to top of page

Command-line requests for user trace services

A new command-line utility, named fbtracemgr, has been added for working interactively with trace services. It has its own syntax of switches and parameters, discussed in detail, with examples, in the Utilities chapter.

As well, the general Services utility, fbsvcmgr, can be used for submitting service requests from the command line, as exemplified in the following examples.

a. Start a user trace named My trace using a configuration file named fbtrace.conf and read its output on the screen:

```
fbsvcmgr service_mgr action_trace_start trc_name "My trace" trc_cfg
fbtrace.conf
```

To stop this trace session, press [Ctrl + C] at the fbsvcmgr console prompt. (See also (e), below).

b. List trace sessions:

```
fbsvcmgr service_mgr action_trace_list
```

c. Suspend trace sesson with ID 1

```
fbsvcmgr service_mgr action_trace_suspend trc_id 1
```

d. Resume trace sesson with ID 1

```
fbsvcmgr service_mgr action_trace_resume trc_id 1
```

e. Stop trace sesson with ID 1

```
fbsvcmgr service_mgr action_trace_stop trc_id 1
```

Tip: List sessions (see b.) in another console, look for the ID of a session of interest and use it in the current console to stop the session.

back to top of page

Trace scope on Windows

Tracker reference CORE-2588

On Windows, the trace tools exhibit shared memory conflicts if multiple engine instances in different Windows sessions are allowed to run *trace* in global namespace. Tracing scope has therefore been restricted to only those processes that are accessible from the current Windows session.

Use cases

There are three general use cases:

1. Constant audit of engine activity

This is served by system audit trace. Administrator creates or edits the trace configuration file, sets its name via the AuditTraceConfigFile setting in firebird.conf and restarts Firebird. Later, the administrator could suspend, resume or stop this session without needing to restart Firebird.

Important: To make audit configuration changes known to the engine, Firebird must be restarted.

2. On-demand interactive trace of some (or all) activity in some (or all) databases

An application (which could be the fbtracemgr utility) starts a user trace session, reads its output and shows traced events to the user in real time on the screen. The user can suspend and resume the trace and, finally, stop it.

3. Engine activity collection for a significant period of time (a few hours or perhaps even a whole day) for later analysis

An application starts a user trace session, reading the trace output regularly and saving it to one or more files. The session must be stopped manually, by the same application or by another one. If multiple trace sessions are running, a listing can be requested in order to identify the session of

interest.

Trace plug-in facilities

A new Trace API will provide a set of hooks which can be implemented as an external plug-in module to be called by the engine when any traced event happens. A plug-in will assume responsibility for logging such events in some custom way.

This Trace API exists in Firebird 2.5 and is in use. However, since it will be changed in forthcoming sub-releases, it is not officially published and must be regarded as unstable.

The implemented “standard” trace plug-in, fbtrace.dll (.so), resides in the \plugins folder of your Firebird 2.5 installation.

back to top of page

Monitoring improvements

Dmitry Yemanov

Firebird 2.5 sees the enhancement of the MON$ database monitoring features introduced in v.2.1, with new tables delivering data about context variables and memory usage in ODS 11.2 and higher databases. Also, in these databases, it becomes possible to terminate a client connection from another connection through the MON$ structures.

Extended access for ordinary users

The original design allowed non-privileged database users to see monitoring information pertaining only to their CURRENT_CONNECTION. Now they can request information for any attachment that was authenticated using the same user name.

Tracker reference CORE-2233.

Notes:

1. For an application architecture that entails a middleware tier logging in multiple times concurrently with the same user name on behalf of different end users, consideration should be given to the impact on performance and privacy of exposing the monitoring features to the end users.

2. The same extension was implemented in v.2.1.2.

New MON$ metadata for ODS 11.2 databases

Note: For the ODS 11.1 metadata please refer to the v.2.1 documentation.

Character set change for MON$ metadata

The system domain RDB$FILE_NAME2, that is used to define those columns in the MON$ tables that pertain to file specifications has been altered from CHARACTER SET NONE to CHARACTER SET UNICODE_FSS. The columns currently affected are MON$DATABASE_NAME, MON$ATTACHMENT_NAME and MON$REMOTE_PROCESS. This change makes the affected data consistent with the updated v.2.5 handling of filespec and other character parameter items in the DPB.

(Tracker entry CORE-2551, A. dos Santos Fernandes)

MON$MEMORY_USAGE (current memory usage)

- MON$STAT_ID (statistics ID)
- MON$STAT_GROUP (statistics group)

0: database

1: attachment

2: transaction

3: statement

4: call

- MON$MEMORY_USED (number of bytes currently in use)

High-level memory allocations performed by the engine from its pools. Can be useful for tracing memory leaks and for investigating unusual memory consumption and the attachments, procedures, etc. that might be responsible for it.

```
* <color #c3c3c3>MON$MEMORY_ALLOCATED</color> (number of bytes currently
allocated at the OS level)
```

Low-level memory allocations performed by the Firebird memory manager. These are bytes actually allocated by the operating system, so it enables the physical memory consumption to be monitored.

Note: Not all records have non-zero values. On the whole, only MON$DATABASE and memory-bound objects point to non-zero “allocated” values. Small allocations are not allocated at this level, being redirected to the database memory pool instead.

- MON$MAX_MEMORY_USED (maximum number of bytes used by this object)
- MON$MAX_MEMORY_ALLOCATED (maximum number of bytes allocated from the operating system by this object)

MON$CONTEXT_VARIABLES (known context variables)

- MON$ATTACHMENT_ID (attachment ID)

Contains a valid ID only for session-level context variables. Transaction-level variables have this field set to NULL.

- MON$TRANSACTION_ID (transaction ID)

Contains a valid ID only for transaction-level context variables. Session-level variables have this field set to NULL.

- MON$VARIABLE_NAME (name of context variable)
- MON$VARIABLE_VALUE (value of context variable)

Memory Usage in MON$STATEMENTS and MON$STATE: Memory usage statistics in MON$STATEMENTS and MON$STATE represent actual CPU consumption. Tracker reference: CORE-1583)

back to top of page

Usage notes

Examples

“Top 10” statements ranked according to their memory usage:

```
SELECT FIRST 10
  STMT.MON$ATTACHMENT_ID,
  STMT.MON$SQL_TEXT,
  MEM.MON$MEMORY_USED
FROM MON$MEMORY_USAGE MEM
  NATURAL JOIN MON$STATEMENTS STMT
  ORDER BY MEM.MON$MEMORY_USED DESC
```

To enumerate all session-level context variables for the current connection:

```
SELECT
  VAR.MON$VARIABLE_NAME,
  VAR.MON$VARIABLE_VALUE
FROM MON$CONTEXT_VARIABLES VAR
  WHERE VAR.MON$ATTACHMENT_ID = CURRENT_CONNECTION
```

Terminating a client

The MON$ structures are, by design, read-only. Thus, user DML operations on them are prohibited. However, a mechanism is built in to allow deleting (only) of records in the MON$STATEMENTS and MON$ATTACHMENTS tables. The effect of this mechanism is to make it possible, respectively, to cancel running statements and, for ODS 11.2 databases, to terminate client sessions.

To cancel all current activity for a specified connection:

```
DELETE FROM MON$STATEMENTS
  WHERE MON$ATTACHMENT_ID = 32
```

To disconnect all clients except the Me connection:

```
DELETE FROM MON$ATTACHMENTS
  WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

Note:

1.

- A statement cancellation attempt becomes a void operation (“no-op”) if the client has no statements currently running.
- Upon cancellation, the execute/fetch API call returns the isc_cancelled error code.
- Subsequent operations are allowed.

2.

- Any active transactions in the connection being terminated will have their activities cancelled immediately and they are rolled back.
- Once terminated, the client session receives the isc_att_shutdown error code.
- Subsequent attempts to use this connection handle will cause network read/write errors.