Red Database

version 2.5

Release notes

Table of content

1 DDL operations access

Functional GRANT and REVOKE operators is expanded for following objects: PROCEDURE, FUNCTION, ROLE, TABLE, VIEW, EXCEPTION, GENERATOR, DOMAIN, SHADOW, POLICY.

Privileges on CREATE operation can be granted to or revoked from user or role with taken of the following GRANT and REVOKE statements:

```
GRANT CREATE OBJECT TO USER|ROLE [WITH GRANT OPTION];

REVOKE CREATE OBJECT FROM USER|ROLE;
```

Privileges on ALTER and DROP operations can be granted to or revoked from user or role with taken of the following GRANT and REVOKE statements:

```
GRANT ALTER|DROP [ANY] OBJECT TO USER|ROLE [WITH GRANT
OPTION];

REVOKE ALTER|DROP [ANY] OBJECT FROM USER|ROLE;
```

Note: ANY will enable the grantee to alter or drop any objects of stated type. Otherwise the grantee can alter or drop objects if it is their owner.

WITH GRANT OPTION will enable the grantee to grant those object privileges to other users and roles.

In Table 1 statements for appointment privileges on DDL operations are presented.

Table 1 - Appointment privileges on DDL operations

Operations	Object	Statement
CREATE, ALTER, DROP	TABLE	GRANT CREATE | ALTER | DROP [ANY] TABLE TO USER|ROLE [WITH GRANT OPTION]
CREATE, ALTER, DROP	TRIGGER	The privileges are defined by the ALTER permissions on table/view
CREATE, ALTER, DROP	PROCEDURE	GRANT CREATE | ALTER | DROP [ANY] PROCEDURE TO USER|ROLE [WITH GRANT OPTION]
CREATE, ALTER, DROP	VIEW	GRANT CREATE | ALTER | DROP [ANY] VIEW TO USER|ROLE [WITH GRANT OPTION]
CREATE, ALTER, DROP	DOMAIN	GRANT CREATE | ALTER | DROP [ANY] DOMAIN TO USER|ROLE [WITH GRANT OPTION]
CREATE, ALTER, DROP	ROLE	GRANT CREATE | ALTER | DROP [ANY] ROLE TO USER|ROLE [WITH GRANT OPTION]
CREATE, DROP	GENERATOR	GRANT CREATE | ALTER | DROP [ANY] GENERATOR TO USER|ROLE [WITH GRANT OPTION]
CREATE, DROP	SEQUENCE	GRANT CREATE | ALTER | DROP [ANY] SEQUENCE TO USER|ROLE [WITH GRANT OPTION]
CREATE, ALTER, DROP	EXCEPTION	GRANT CREATE | DROP [ANY] EXCEPTION TO USER|ROLE [WITH GRANT OPTION]
CREATE, DROP	SHADOW	GRANT CREATE | DROP [ANY] SHADOW TO USER|ROLE [WITH GRANT OPTION]
DECLARE, ALTER, DROP	FUNCTION	GRANT CREATE | ALTER | DROP [ANY] FUNCTION TO USER|ROLE [WITH GRANT OPTION]
CREATE, ALTER, DROP	INDEX	The privileges are defined by the ALTER permissions on table/view
CREATE, DROP	POLICY	GRANT SECADMIN TO USER|ROLE

2 DML operations access

Functional GRANT and REVOKE operators are expanded for a full priveleges differentiation on DML operations.

In the table 2 statements for appointment rights on DML operations are presented.

Table 2 - Appointment rights on DML operations

Operations	Object	Statement

SELECT, INSERT, UPDATE, DELETE	TABLE,VIEW	GRANT SELECT | INSERT | UPDATE | DELETE ON [TABLE] {table} TO {user | role} [WITH GRANT OPTION]
		REVOKE SELECT | INSERT | UPDATE | DELETE ON [TABLE] {table} FROM {user | role}
		REVOKE GRANT OPTION FOR SELECT | INSERT | UPDATE | DELETE ON [TABLE] {table} FROM {user | role}

SET GENERATOR, GEN_ID	GENERATOR	GRANT SELECT | ALTER ON GENERATOR {generator} TO {user | role} [WITH GRANT OPTION]
		REVOKE SELECT | ALTER ON GENERATOR {generator} FROM {user | role}
		REVOKE GRANT OPTION FOR SET | GET ON GENERATOR {generator} FROM {user | role}

SELECT, INSERT, UPDATE	TABLE (column, ...), VIEW (column, ...)	GRANT SELECT | INSERT | UPDATE {(column [, ...])} ON [TABLE] {table} TO {user | role} [WITH GRANT OPTION]
		REVOKE SELECT | INSERT | UPDATE {(column [, ...])} ON [TABLE] {table} FROM {user | role}
		REVOKE GRANT OPTION FOR SELECT | INSERT | UPDATE {(column [, ...])} ON [TABLE] {table } FROM {user | role}

EXECUTE	PROCEDURE	GRANT EXECUTE ON PROCEDURE {procedure} TO {user | role} [WITH GRANT OPTION]
		REVOKE EXECUTE ON PROCEDURE {procedure} FROM {user | role}
		REVOKE GRANT OPTION FOR EXECUTE ON PROCEDURE {procedure} FROM {user | role}

3 Pre-defined global roles

There are three pre-defined global roles: SYSADMIN, SECADMIN, PUBLIC.

Table 3 - Pre-defined roles rights

Operations	SYSADMIN	SECADMIN	PUBLIC
DML-operations	+	+/- (Can grant privileges himself)	-
DDL-operations	+	Operations with roles only	-
GRANT/REVOKE	+	+	-
backup/restore	+	-	-
Operations with users (GSEC)	-	+	-
GFIX and GSTAT	+	-	-

All new users have role PUBLIC as default.

SYSDBA or user with global role SECADMIN can create additional global roles. They can grant to this roles privileges on execute services[1].

Note: Global roles must be created when user directly connected to security2.fdb

[1] See chapter 4 «Administration functions access (services)" for details

4 Administration functions access (services)

Red Database 2.5 supports rights assignment on services execution.

The list of administrative services:

- Backup and restore (GBAK);
- Database validation (GFIX);
- Database statistics (GSTAT).
- Users management (GSEC)

The permissions on services execution is a global right for database server therefore it is stored in the system database security2.fdb. It is possible to grant the right on service execution for users and global roles. SYSDBA and SECADMIN can grant right on services execution only.

Right on service execution can be granted to or revoked from user or global role with taken of the following GRANT and REVOKE statements:

```
GRANT EXECUTE ON SERVICE <SERVICE_NAME> TO USER | ROLE
REVOKE EXECUTE ON SERVICE <SERVICE_NAME> FROM USER | ROLE
```

These statements must be executed in the system database security2.fdb.

The parameter <SERVICE_NAME> can have the following values:

- BACKUP
- RESTORE
- GFIX
- GSTAT

Any user can work with GSEC service but his rights are limited by a possibility to change its password. SYSDBA and SECADMIN have the full rights on GSEC service only.

5 Combine roles

New ability granting role to role is added. The next GRANT and REVOKE syntax is used for granting role to role:

```
GRANT ROLE1 TO ROLE2;
REVOKE ROLE1 FROM ROLE2;
```

Cycle roles reference to each other is forbidden.

Combine roles rules are:

- if user doesn't specify any role, all roles granted to user are used;
- if user specifies the role, he takes privileges of this role only;
- role existing and granting to user is verified on connection.

Users and roles names must be different. But this rule can be not always executed, because users are global objects (stored in single database security2.fdb) and roles are local (stored in any databases). There is the next rule when the user name and role name is equal: privileges are firstly granted to role then to user if role doesn't exist.

6 Executing procedures

New ability of procedures execution with owner context is added. To execute procedure successfully users must have privilege on procedure execution instead of objects which are used by procedure.

Option AUTHID OWNER|CALLER is added to definition and alter of procedure to point context of execution.

Procedure definition has the next syntax:

```
CREATE PROCEDURE <procedure_name>
     [AUTHID OWNER|CALLER]
     [(param <datatype> [, param <datatype> …])]
     [RETURNS <datatype> [, param <datatype> …])]
     AS <procedure_body> [terminator]
     <procedure_body> =
     [<variable_declaration_list>]
     <block>
     <variable_declaration_list> =
     DECLARE VARIABLE var <datatype>;
     [DECLARE VARIABLE var <datatype>; …]
     <block> =
     BEGIN
     <compound_statement>
     [<compound_statement> …]
     END
```

7 Multifactor authentication[2]

Recognition is mapping unknown user to the well known identifier so as to make he known for the RDBMS engine.

Authentication is the act of confirming someone as *authentic*, that is, that claims made by or about the thing are true.

Authentication factor is a piece of information used to authenticate or verify a person's identity for security purposes. Red Database uses two kinds of authentication factors. They are primary factors and secondary factors. The primary factors are password, operating system security context and certificate. The secondary factors can be presented after primary authentication by using protected channel that is a result of primary authentication. Now Red Database can use such secondary factors as finger print and retinal scan.

Multifactor authentication is authentication using more than one authentication factors, for example, password, certificate and other. The factors required for authentication are defined by the security policy of the user. The conformance inspection of presented factors to the security policy is performed in multifactor authentication mode only.

Multifactor authentication features:

- ability to use a result of authentication in operating system at local machine or at domain controller;
- multifactor authentication. The access to database is defined by security policy. It says what factors user must present for right authentication.
- While authentication process all authentication factors are transfered as encoded.

While primary authentication server checks all presented primary factors and generates the session keys as result. After authentication a client have generated the session keys too.

By the session key client encodes secondary authentication factors and transfers they to the server.

The Red Database security database contains security policy that define authentication conditions. The user authentication is successful if all the user have presented all required factors.

With password hash security database keeps algorithm name.

The password changing is happen by using session keys for symmetric encoding of new password. On the server side the new password is decoded and verified to password policy requirements. If it is successful the password is saved.

Table 4 - Configuration parameters for multifactor authentication

Parameter name	Parameter value	Comment
Authentication	native | trusted | mixed | multifactor	native — server uses only native firebird authentication trusted — only trusted authentication is allowed (Windows authentication) multifactor — multifactor authentication only is allowed mixed — the server can accept clients with any authentication methd

[2] This feature requires a cryptoprovider (now it's CryptoPro) and cryptoplugin (it 's included into Red Database installer).

Parameter name	Parameter value	Comment
LegacyHash	0 | 1	Use 1 to allow users created as not multifactor connection to server with Autentication=multifactor or mixed. It's useful at the first times for creating a lot of users with multifactor authentication.
CryptoPluginName	Default value is wincrypt_plugin	The name of crypto plugin library (it's located at plugins folder of server installation folder)
KeyRepository	<Repository name>	The repository name with security keys for establishing connections
CertRepository	<Repository name>	The repository name with certificates for server authentication

8 Login policies

The login policies consist of:

- password complexity requirements;
- minimum number of previous passwords that must not be repeated by the new password;
- the password validation interval;
- the maximum number of sessions of user to the server (not database);
- the idle time for user disconnection.

The login policies are keeping in server security database «securuty2.fdb». The login policy defines the system behavior by failed attachments and allows server to lock the user at some time or permanently. The login policy controls password complexity.

There is a special utility named idlectrl implementing sush functions as:

- periodically validates the users that did not have any activity while interval defined by login policy;
- periodically compares the number of sessions of each user and maximum value defined by login policy.

The Red Database uses login policies with features:

- The login policies are using only with multifactor authentication.
- There is a login policy for each multifactor user. The DEFAULT policy automatically granted to all new users;
- The password complexity check is allowed while multifactor authentication only. In this case the session keys are generated. While password is changed it is encoded by the symmetric algorithm with sessions keys generated after authentication. The server decodes received encoded password and validate accordance to the login policy. After that new password hash is calculated and stored in the server security database with hash algorithm name.
- If the login policy sets the password valid interval it checks by connecting to the server. If the password is old connection will be rejected with error message about need to change the password.
- While connecting the server checks a number of existing attachments this user to the server. When it's equal to maximum defined by login policy new connection is rejected with accordance message. The failed attempts increase the failed attempts counter for user. The current counter value is stored in server security database security2.fdb. Next failed attempt blocks user both temporary or permanently if the failed attempt counter value grater then maximum defined by login policy. The time after that user can access to server is stored at security2.fdb.
- The login policy changing is applying immediately and will take effect for the next attachment.

Use the next DDL statements to control login policy:

```
CREATE POLICY <policy_name> AS [param = value [, param =
value]];

DROP POLICY <policy_name>;

ALTER POLICY <policy_name> AS [param = value [, param =
value]];
```

```
GRANT POLICY <policy_name> TO <user_name>;
```

There is no REVOKE POLICY statement as user must have at least one granted policy. Instead of calling REVOKE POLICY you should grant DEFAULT policy to user.

```
GRANT POLICY "DEFAULT" TO <user_name>;
```

The authentication factors are described as symbolic condition. It is combination of possible sets of needed authentication factors. It is allow you to build required condition with AND, OR operators.

Table 5 - The symbolic definition of authentication factors

Symbol	Description
WINDOWS_NTLM	The result of authentication in Windows
CERT_X509	Uuser certificate
PASSWORD	The password
FINGERPRINT	The finger print
EYE	Retinal scan

The required factors condition has syntax:

(A1A2... Al) | (B1B2... Bm) | ... | (Z1Z2 ... Zn) — where Ax, Bx, Zx — the symbolic definition of authentication factors.

Table 6 - The login policy parameters

Parameter name	Type	Description
AUTH_FACTORS	auth_factors_expr_list	See table 5.
PSWD_NEED_CHAR	long_integer	The minimum number of characters in the password
PSWD_NEED_DIGIT	long_integer	The minimum number of digits in the password
PSWD_NEED_DIFF_CASE	bool_const	Need to use different case of characters in the password
PSWD_MIN_LEN	long_integer	The minimum password length
PSWD_VALID_DAYS	long_integer	The password validation interval in days
PSWD_UNIQUE_COUNT	long_integer	The minimum number of the last unique passwords
MAX_FAILED_COUNT	long_integer	The maximum number of failed attempt of authentication
MAX_SESSIONS	long_integer	The maximum number of user sessions to database server
MAX_IDLE_TIME	long_integer	The maximum idle time interval to user disconnecting

9 Record and column filter

There is an experimental implementation of ISO/IEC 27000 series security requirements for Red Database 2.5.

While you are creating or altering a table the next syntax is used to set filter for records or columns:

```
CREATE TABLE <table_name> [EXTERNAL [FILE] "<filespec>"]
(<col_def> [, <col_def> | <tconstraint> ...], [COLFILTER
<col_name> (<condition>), ...]) [, RECFILTER
(<condition>)]
```

COLFILTER allows you to hide column content for a particular row, and RECFILTER controls when a particular record is visible.

<condition> can check access rights for the user. There are special built-in functions provided for this purpose (CHECK_DDL_RIGHTS, CHECK_DML_RIGHTS).

The next syntax is used to ALTER or DROP conditions for filtering records:

```
ALTER TABLE table SET RECFILTER (<condition>);

ALTER TABLE table DROP RECFILTER;
```

The next syntax is used to ALTER or DROP conditions for filtering columns:

```
ALTER TABLE table SET COLFILTER <col_name> (<condition>);

ALTER TABLE table DROP COLFILTER <col_name>;
```

COLFILTER/RECFILTER are implemented as special SELECT triggers. If you want, you can define custom SELECT trigger yourself, and post audit event which can get recorded in external audit log. Trace API configuration file allows you to set up filtering rules saying for which audit events to record information into the log file[3].

Audit log can be stored on restricted write-only media, and security administrator can attach it later as external table to another database for analysis, if necessary. Automatically posting audit event if RECFILTER or COLFILTER condition is violated is something which we should consider implementing.

To switch on Record/Column filter mode is used parameter UseRecordFilter = 0/1 in firebird.conf file.

There are a pre-defined conditions for system tables:

Table 7 - Pre-defined conditions for record and column filter

Table name	Table rights	Record access	Column access
RDB$BACKUP_HISTORY	SYSDBA only	Not filtering	RDB$BACKUP_ID - Not hide RDB$TIMESTAMP - Not hide RDB$BACKUP_LEVEL - Not hide RDB$GUID - Not hide RDB$SCN - Not hide RDB$FILE_NAME - Not hide
RDB$CHARACTER_SETS	All	Not filtering	RDB$CHARACTER_SET_NAME - Not hide RDB$FORM_OF_USE - Not hide RDB$NUMBER_OF_CHARACTERS - Not hide

[3] See chapter 8 «Audit system» for details

Table name	Table rights	Record access	Column access
			RDB$DEFAULT_COLLATE_NAME - Not hide RDB$CHARACTER_SET_ID - Not hide RDB$SYSTEM_FLAG - Not hide RDB$DESCRIPTION - Not hide RDB$FUNCTION_NAME - Not hide RDB$BYTES_PER_CHARACTER - Not hide
RDB$CHECK_CONSTRAINTS	All	Only constraints of which user has rights (rights on any DDL or DML operations).	RDB$CONSTRAINT_NAME - Not hide RDB$TRIGGER_NAME - Not hide
RDB$COLLATIONS	All	Not filtering	RDB$COLLATION_NAME - Not hide RDB$COLLATION_ID - Not hide RDB$CHARACTER_SET_ID - Not hide RDB$COLLATION_ATTRIBUTES - Not hide RDB$SYSTEM_FLAG - Not hide RDB$DESCRIPTION - Not hide RDB$FUNCTION_NAME - Not hide RDB$BASE_COLLATION_NAME - Not hide RDB$SPECIFIC_ATTRIBUTES - Not hide
RDB$DATABASE	All	Not filtering	RDB$DESCRIPTION - Not hide RDB$RELATION_ID - Not hide RDB$SECURITY_CLASS - Not hide RDB$CHARACTER_SET_NAME - Not hide
RDB$DEPENDENCIES	All	All of which user has DML rights on field RDB$FIELD_NAME	RDB$DEPENDENT_NAME - Not hide RDB$DEPENDED_ON_NAME - Not hide RDB$FIELD_NAME - Not hide RDB$DEPENDENT_TYPE - Not hide RDB$DEPENDED_ON_TYPE - Not hide
RDB$EXCEPTIONS	All	All of which user has DDL-rights	RDB$EXCEPTION_NAME - Not hide RDB$EXCEPTION_NUMBER - Not hide RDB$MESSAGE - Not hide RDB$DESCRIPTION - Not

Table name	Table rights	Record access	Column access
			hide RDB$SYSTEM_FLAG - Not hide
RDB$FIELDS	All	All of which user has the rights (DDL or DML)	RDB$FIELD_NAME - Not hide RDB$QUERY_NAME - Not hide RDB$VALIDATION_BLR – Hide RDB$VALIDATION_SOURCE - Not hide RDB$COMPUTED_BLR - Hide RDB$COMPUTED_SOURCE - Hide RDB$DEFAULT_VALUE - Not hide RDB$DEFAULT_SOURCE - Hide RDB$FIELD_LENGTH- Not hide RDB$FIELD_SCALE - Not hide RDB$FIELD_TYPE - Not hide RDB$FIELD_SUB_TYPE - Not hide RDB$MISSING_VALUE - Not hide RDB$MISSING_SOURCE - Not hide RDB$DESCRIPTION - Not hide RDB$SYSTEM_FLAG - Not hide RDB$QUERY_HEADER - Not hide RDB$SEGMENT_LENGTH - Not hide RDB$EDIT_STRING - Not hide RDB$EXTERNAL_LENGTH- Not hide RDB$EXTERNAL_SCALE - Not hide RDB$EXTERNAL_TYPE- Not hide RDB$DIMENSIONS- Not hide RDB$NULL_FLAG - Not hide RDB$CHARACTER_LENGTH - Not hide RDB$COLLATION_ID - Not hide RDB$CHARACTER_SET_ID - Not hide RDB$FIELD_PRECISION - Not hide
RDB$FIELD_DIMENSIONS	All	All of which user has permission	RDB$FIELD_NAME - Not hide RDB$DIMENSION - Not hide RDB$LOWER_BOUND - Not hide

Table name	Table rights	Record access	Column access
			RDB$UPPER_BOUND - Not hide
RDB$FILES	All	Only if user has DDL-rights on operations with shadow.	RDB$FILE_NAME - Not hide RDB$FILE_SEQUENCE - Not hide RDB$FILE_START - Not hide RDB$FILE_LENGTH - Not hide RDB$FILE_FLAGS - Not hide RDB$SHADOW_NUMBER - Not hide
RDB$FILTERS	All	Not filtering	RDB$FUNCTION_NAME - Not hide RDB$DESCRIPTION - Not hide RDB$MODULE_NAME - Not hide RDB$ENTRYPOINT - Not hide RDB$INPUT_SUB_TYPE - Not hide RDB$OUTPUT_SUB_TYPE - Not hide RDB$SYSTEM_FLAG - Not hide
RDB$FORMATS	All	Only of which user has rights on table	RDB$RELATION_ID - Not hide RDB$FORMAT - Not hide RDB$DESCRIPTOR - Not hide
RDB$FUNCTIONS	All	All of which user has DDL-rights	RDB$FUNCTION_NAME - Not hide RDB$FUNCTION_TYPE - Not hide RDB$QUERY_NAME - Not hide RDB$DESCRIPTION - Not hide RDB$MODULE_NAME - Not hide RDB$ENTRYPOINT - Not hide RDB$RETURN_ARGUMENT - Not hide RDB$SYSTEM_FLAG - Not hide
RDB$FUNCTION_ARGUMENTS	All	All of which user has rights on functions	RDB$FUNCTION_NAME - Not hide RDB$ARGUMENT_POSITION - Not hide RDB$MECHANISM - Not hide RDB$FIELD_TYPE - Not hide RDB$FIELD_SCALE - Not hide RDB$FIELD_LENGTH - Not hide RDB$FIELD_SUB_TYPE - Not hide RDB$CHARACTER_SET_ID -

Table name	Table rights	Record access	Column access
			Not hide RDB$FIELD_PRECISION - Not hide RDB$CHARACTER_LENGTH - Not hide
RDB$GENERATORS	All	All of which user has DDL-rights	RDB$GENERATOR_NAME - Not hide RDB$GENERATOR_ID - Not hide RDB$SYSTEM_FLAG - Not hide RDB$DESCRIPTION - Not hide
RDB$INDEX_SEGMENTS	All	All of which user has rights on tables	RDB$INDEX_NAME - Not hide RDB$FIELD_NAME - Not hide RDB$FIELD_POSITION - Not hide RDB$STATISTICS - Not hide
RDB$INDICES	All	All of which user has permissions	RDB$INDEX_NAME - Not hide RDB$RELATION_NAME - Not hide RDB$INDEX_ID - Not hide RDB$UNIQUE_FLAG - Not hide RDB$DESCRIPTION - Not hide RDB$SEGMENT_COUNT - Not hide RDB$INDEX_INACTIVE - Not hide RDB$INDEX_TYPE - Not hide RDB$FOREIGN_KEY - Not hide RDB$SYSTEM_FLAG - Not hide RDB$EXPRESSION_BLR - Hide RDB$EXPRESSION_SOURCE - Hide RDB$STATISTICS - Not hide
RDB$LOG_FILES	SYSDBA only	Not filtering	RDB$FILE_NAME - Not hide RDB$FILE_SEQUENCE - Not hide RDB$FILE_LENGTH - Not hide RDB$FILE_PARTITIONS - Not hide RDB$FILE_P_OFFSET - Not hide RDB$FILE_FLAGS - Not hide
RDB$PAGES	SYSDBA only	Not filtering	RDB$PAGE_NUMBER - Not hide RDB$RELATION_ID - Not hide RDB$PAGE_SEQUENCE - Not hide RDB$PAGE_TYPE - Not hide

Table name	Table rights	Record access	Column access
RDB$PROCEDURES	All	All of which user has permissions	RDB$PROCEDURE_NAME - Not hide RDB$PROCEDURE_ID - Not hide RDB$PROCEDURE_INPUTS - Not hide RDB$PROCEDURE_OUTPUTS - Not hide RDB$DESCRIPTION - Not hide RDB$PROCEDURE_SOURCE - Hide RDB$PROCEDURE_BLR - Hide RDB$SECURITY_CLASS - Not hide RDB$OWNER_NAME - Not hide RDB$RUNTIME - Not hide RDB$SYSTEM_FLAG - Not hide RDB$PROCEDURE_TYPE - Not hide RDB$VALID_BLR - Not hide RDB$DEBUG_INFO - Not hide
RDB$PROCEDURE_PARAMETERS	All	All of which user has permissions	RDB$PARAMETER_NAME - Not hide RDB$PROCEDURE_NAME - Not hide RDB$PARAMETER_NUMBER - Not hide RDB$PARAMETER_TYPE - Not hide RDB$FIELD_SOURCE - Not hide RDB$DESCRIPTION - Not hide RDB$SYSTEM_FLAG - Not accessible RDB$DEFAULT_VALUE - Not hide RDB$DEFAULT_SOURCE - Hide RDB$COLLATION_ID - Not hide RDB$NULL_FLAG - Not hide RDB$PARAMETER_MECHANISM - Not hide
RDB$REF_CONSTRAINTS	All	All of which user has permissions	RDB$CONSTRAINT_NAME - Not hide RDB$CONST_NAME_UQ - Not hide RDB$MATCH_OPTION - Not hide RDB$UPDATE_RULE - Not hide RDB$DELETE_RULE - Not hide

Table name	Table rights	Record access	Column access
RDB$RELATIONS	All	All of which user has permissions	RDB$VIEW_BLR - Hide RDB$VIEW_SOURCE - Hide RDB$DESCRIPTION - Not hide RDB$RELATION_ID - Not hide RDB$SYSTEM_FLAG - Not hide RDB$DBKEY_LENGTH - Not hide RDB$FORMAT - Not hide RDB$FIELD_ID - Not hide RDB$RELATION_NAME - Not hide RDB$SECURITY_CLASS - Not hide RDB$EXTERNAL_FILE - Not hide RDB$RUNTIME - Not hide RDB$EXTERNAL_DESCRIPTION - Not hide RDB$OWNER_NAME - Not hide RDB$DEFAULT_CLASS - Not hide RDB$FLAGS - Not hide RDB$RELATION_TYPE - Not hide
RDB$RELATION_CONSTRAINTS	All	All of which user has DDL-rights	RDB$CONSTRAINT_NAME - Not hide RDB$CONSTRAINT_TYPE - Not hide RDB$RELATION_NAME - Not hide RDB$DEFERRABLE - Not hide RDB$INITIALLY_DEFERRED - Not hide RDB$INDEX_NAME - Not hide
RDB$RELATION_FIELDS	All	All of which user has permissions	RDB$FIELD_NAME - Not hide RDB$RELATION_NAME - Not hide RDB$FIELD_SOURCE - Hide RDB$QUERY_NAME - Not hide RDB$BASE_FIELD - Not hide RDB$EDIT_STRING - Not hide RDB$FIELD_POSITION - Not hide RDB$QUERY_HEADER - Not hide RDB$UPDATE_FLAG - Not hide RDB$FIELD_ID - Not hide RDB$VIEW_CONTEXT - Not hide RDB$DESCRIPTION - Not hide RDB$DEFAULT_VALUE - Not

Table name	Table rights	Record access	Column access
			hide RDB$SYSTEM_FLAG - Not hide RDB$SECURITY_CLASS - Not hide RDB$COMPLEX_NAME - Not hide RDB$NULL_FLAG - Not hide RDB$DEFAULT_SOURCE - Not hide RDB$COLLATION_ID - Not hide
RDB$ROLES	All	All roles of current user	RDB$ROLE_NAME - Not hide RDB$OWNER_NAME - Not hide RDB$DESCRIPTION - Not hide RDB$SYSTEM_FLAG - Not hide
RDB$SECURITY_CLASSES	SYSDBA only	Not filtering	RDB$SECURITY_CLASS - Not hide RDB$ACL - Not hide RDB$DESCRIPTION - Not hide
RDB$TRANSACTIONS	All	Not filtering	RDB$TRANSACTION_ID — Not hide RDB$TRANSACTION_STATE — Not hide RDB$TIMESTAMP — Not hide RDB$TRANSACTION_DESCRIPTION — Not hide
RDB$TRIGGERS	All	Only if user has permissions on ALTER TABLE, INSERT/UPDATE/DELETE record in table	RDB$TRIGGER_NAME - Not hide RDB$RELATION_NAME - Not hide RDB$TRIGGER_SEQUENCE - Not hide RDB$TRIGGER_TYPE - Not hide RDB$TRIGGER_SOURCE – is Hide RDB$TRIGGER_BLR - Hide RDB$DESCRIPTION - Not hide RDB$TRIGGER_INACTIVE - Not hide RDB$SYSTEM_FLAG — Not hide RDB$FLAGS - Not hide RDB$VALID_BLR — Not hide RDB$DEBUG_INFO - Not hide
RDB$TRIGGER_MESSAGES	All	All of which user has permissions	RDB$TRIGGER_NAME - Not hide RDB$MESSAGE_NUMBER -

Table name	Table rights	Record access	Column access
			Not hide RDB$MESSAGE - Not hide
RDB$TYPES	All	Not filtering	RDB$FIELD_NAME - Not hide RDB$TYPE - Not hide RDB$TYPE_NAME - Not hide RDB$DESCRIPTION - Not hide RDB$SYSTEM_FLAG — Not hide
RDB$USER_PRIVILEGES	All	All privileges of current user	RDB$USER - Not hide RDB$GRANTOR - Not hide RDB$PRIVILEGE - Not hide RDB$GRANT_OPTION - Not hide RDB$RELATION_NAME - Not hide RDB$FIELD_NAME - Not hide RDB$USER_TYPE - Not hide RDB$OBJECT_TYPE - Not hide
RDB$VIEW_RELATIONS	All	All of which user has permissions	RDB$VIEW_NAME — Not hide RDB$RELATION_NAME - Not hide RDB$VIEW_CONTEXT - Not hide RDB$CONTEXT_NAME - Not hide

10 Audit system

FBTrace is a tool for saving queries to Red Database server. It starts at the same time as the server and registers requests to databases into log files. By default log file is created in database directory and named like: <database_name.fbtrace_text> (text log format) or <database_name.fbtrace_bin> (binary log format). When you create database or connect to it FBTrace begins to work and saves queries to this database. Disconnecting from database stops logging.

10.1 Types of audit events

There are following types of audit events:

- start and stop logging of the database;
- create/attach and drop/detach database:
- attach, start, query and detach service;
- accept authentication factor;
- prepare, execute and free SQL statement;
- compile and execute BLR statement;
- execute DYN request;
- execute stored procedure;
- set context variable;
- start/end transaction.

For all events (except accept authentication factor) FBTrace saves common information:

- date, time and type of the event;
- ID of the process raised the event;
- event result: successful, failed, unauthorized;
- some information about attachment:
  - path to database file;
  - attachment ID;
  - user on behalf of which the event occurs;
  - network connection protocol;
  - host name created connection.

FBTrace saves different information for each audit event depending on event type.

1. Attach to database: whether is connect to existing database or create new one.
2. Detach from database: whether is disconnect from database or drop it.
3. Authentication factor: date, time, event type, process ID, user name, factor type, factor data (depends on factor type), result of factor verification (successful, failed, unauthorized).
4. Attach, detach, start service: common information about event except database path and attachment ID. In addition to common information service name is saved.
5. For service start event command line switches also saved.
6. For service query event command also saved: service name and query data.
7. Set context variable: information about transaction, variable namespace, variable name and new value.

8. Execute stored procedure: information about transaction, procedure name, input parameters, execution time, performance statistics.
9. Transaction: transaction ID, transaction options. For transaction end event also saved: commit or rollback completion type.
10. Prepare SQL statement: information about transaction, statement ID, statement data, preparation time (milliseconds), execution plan.
11. Execute SQL or BLR statement: information about transaction, statement ID, statement data (for text log format only), execution time, performance statistics, execute parameters.
12. Execute DYN request: information about transaction, statement data (depends on log format type), execution time.
13. Compile BLR statement: information about transaction, statement ID, statement data (depends on log format type), execution time.

Annotation: Unauthorized event means event with failed authentication or insufficient rights. Failed event – any other unsuccessful action.

Annotation: Audit is turned off by default.

Annotation: Messages about services using and authentication factors accepting are written to security2.fdb log file.

Annotation: You can use log rotating, which automatically backup current log file when it grows to a specify size. Current log file is renamed to a new file named <log_filename>.<date_and_time>.<log_ext>, where date and time used following format: <YYYY-MM-DDThh-mm-ss>; <log_ext> – log file extension. After renaming new log file is created and used as current log. Deleting and compressing old log files are not provided and can be performed using OS scheduler.

Binary log files have a version of their format. It's checked at audit initialization event. If version number of used log file differs from version number of current release then log rotating is performed.

10.2 Audit configuration

For audit configuration used file fbtrace.conf which is placed in the Red Database installation directory. Blank lines in the file are ignored. Comments may be present on lines that start with the # character, which are also ignored. Configuration file is self-documented by in-line comments.

You can specify particular logged databases using regular expressions. For example:

Ex. 1:

```
#Logged databases with the names test.fdb, azk2.fdb,
rules.fdb
[^.*[\/](test|azk2|rules)\.fdb$]:
enabled = 1
#Logs are saved to files test.log, azk2.log, rules.log
log_filename = $1.log
```

Ex. 2:

```
#Use text log format for all databases on disk C:
[^C:.*\.fdb$]:
```

```
enabled = 1
format = 0
#Use binary log format for all databases on disk D:
[^D:.*\.fdb$]:
enabled = 1
format = 1
```

You can use grouping in regular expressions:
Ex. 3:

```
#First group (.*[\/]) - any path to database file
#Second group (test|azk) — file name of database test or azk
[^(.*[\/])(test|azk)\.fdb$]:
enabled = 1
#$1 — First group of regular expression – path.
#$2 — Second group of regular expression – file name.
#This example creates log files <database_name.log> in “logs” directory.
log_filename = $1../logs/$2.log
```

Ex. 4:

```
#All types of event will be registered
enabled = 1
format = 0
time_threshold = 0
max_sql_length = 300
max_blr_length = 500
max_dyn_length = 500
max_arg_length = 80
max_arg_count = 30
max_log_size = 0
print_blr = 1
print_dyn = 1
log_auth_factors = 1
log_connections = 1
log_transactions = 1
log_statements = 1
log_context = 1
```

```
log_procedures = 1
log_execute = 1
log_blr_requests = 1
log_dyn_requests = 1
log_services = 1
```

11 Log analyzer

Log analyzer is a tool for analysis audit journals in binary format with ability of filtering journal records using SELECT statement.

Analyzer parses binary log files created by Red Database 2.5 trace system and provides features for viewing and filtering records. Editing and deleting records are not allowed.

Way to work with Log analyzer:

- attach binary log file to Red Database 2.5 as an external table;
- use SQL statements to select records from attached table.

So Log analyzer can be used in any supported operating system even from command line. Also you can work with external table using your favorite GUI tool, such as IBExpert or FenixSQL.

To register an audit journal execute statement:

```
CREATE TABLE <table_name> EXTERNAL [FILE] '<filespec>'
ADAPTER 'fbtrace' [<col_defs>]
```

where:

- table_name – name of the table which will be store audit records;
- filespec – path to log file to be opened;
- ADAPTER – keyword for adapter register;
- fbtrace – adapter name for log files created by RDB trace system;
- col_defs – audit table fields definitions.

Some fields of audit record can be empty depending on type of audit event.

It is not necessary to define fields set if you use “fbtrace” adapter because in this case table structure is known beforehand.

- If you use keyword “ADAPTER” without fields set definition, table of the specified adapter will be created.
- If you specify both adapter type and table fields, specified fields must be a subset of adapter table fields. Fields name and type must correspond to adapter fields specification. Fields definition order is unimportant.
- If you define a field and there is no corresponded field in adapter table, raises error and adapter table will not be created.
- If you define not all adapter fields, missed fields of adapter table will be ignored.
- Binary log file contains a version number of its format. If it differs from format number of the release, you get an error.
- If you try to use adapter in database whose ODS not supports adapters, you get an error.

Table 8 - Fields of audit table

Field name	Field type	Description
event_time	TIMESTAMP	Date and time of the event
event_process_id	INTEGER;	ID of the process, raising the event
event_object_id	VARCHAR(16)	ID of the object, raising the event (field size if depends on size of the pointers in OS)
event_att_id	INTEGER	ID of the attachment in which the event occurs
event_database	BLOB	Database in which the event occurs
event_user	BLOB	User on behalf of which the event occurs

Field name	Field type	Description
event_protocol	BLOB	Protocol of network connection
event_hostname	BLOB	Host name created connection
event_type	CHAR(20)	Event type
auth_factor_type	BLOB	Types of authentication factors
auth_factor_data	BLOB	Content of authentication factors
trans_id	INTEGER	Transaction ID
trans_opt	BLOB	Transaction options
stmt_id	VARCHAR(16)	Statement ID
stmt_sql	BLOB	SQL statement data
blr_data	BLOB SUB_TYPE BLR	BLR request content
dyn_data	BLOB SUB_TYPE BLR	DYN request content
stmt_access_path	BLOB	Statement execution plan
stmt_params	BLOB	Statement parameters
var_name	BLOB	Context variable name
var_ns	BLOB	Namespace of context variable
var_value	BLOB	Setting context variable value
svc_id	VARCHAR(16)	Service ID
svc_name	BLOB	Service name
svc_switches	BLOB	Service switches
svc_query	BLOB	Service query
proc_name	BLOB	Stored procedure name
proc_params	BLOB	Procedure parameters
perf_time	INTEGER;	Execution time
perf_info	BLOB	Performance statistics
row_fetched	INTEGER	Fetched rows count
event_result	VARCHAR(12)	Event result (successful, failed, unauthorized)

12 Integrity checking[4]

12.1 Metadata integrity checking

The Red Database 2.5 has utility for watching to metadata integrity. It's called “mint” and it's located in bin folder. This tool can extract and hashing metadata from user databases. So the database administrator can protect the database structure from any modifications.

The mint utility allows you to extract both all metadata from database or part of it according to user mask. It can also check the current state of database structure. For this purpose it repeatedly extracts necessary metadata, calculates they hashs and compare they with stored before.

Mint has such switches and options as:

```
-M <extract|check>
[<-m mask>]
-d <database name>
-u <user name for connection to database>
-p <user password for connection to database>
-r <key repository for storing cryptographic keys>
-R <Encoding algorithm for key repository>
-i <file to store|check the metadata digital signature>
-I <name of digital signature algorithm>
```

There are two modes of mint working – hash extracting and checking.

When -m switch isn't absent all metadata are extracting. The symbol % in mask replaces any number of symbols. For example, to generate digital signatures for all system objects which names starting with “RDB$” enter the next command

```
mint -M extract -m RDB$% -d ../security2.fdb -u sysdba -p
masterkey -r test -R "Crypto Pro" -i sign -I AT_SIGNATURE
```

for checking digital signature:

```
mint -M check -m RDB$% -d ../security2.fdb -u sysdba -p
masterkey -r test -R "Crypto Pro" -i sign -I AT_SIGNATURE
Signature verification complete successfully
```

12.2 Server files integrity checking

The server files integrity check means that for all critical important server files (such as security database security2.fdb, binary server files, fb_inet_server, different library and utilities, etc) server can automatically check hashes when it started. The standard set of hashes are included into installation pack of Red database 2.5

For switch on server files integrity check mode parameter HashesFile from firebird.conf is used. The value of this parameter is name of file with hashes. This name can include absolute or relative path to file (in case of relative path it specified from Red Database root directory).

[4] Note: Value of parameter CryptoPluginName from firebird.conf must be set for correct work metadata integrity check, server files integrity check, mint and hashgen utilites

Each row in this file have next format:

```
<hash_value> <hash_alghoritm_name> <file_name>
```

When server is starting each row from hashes file are checked. For each file new hash is generated and compare with stored for it in hashes file. If new and stored in file hashes are not equivalent server is not started and name of corrupt file is stored in firebird.log.

Installation pack of Red database 2.5 included a special utility for manual check and generate hash of files. It is called hashgen from directory bin. System or database administrator can generate new hash for any file, modify list of checked files or manually check hash for any file.

The input parameters of this utility are mode, hashing file name, file name for storing hash and hash algorithm name. Running utility without parameters displays help about available parameters and known hashing algorithms.

Utility have two work modes — generating and checking — for generating and checking files hash accordingly.

```
hashgen.exe generate CALG_GR3411 ../security2.fdb
>test.sign
```

In this example the hash for file security2.fdb is generated with algorithm name CALG_GR3411 and stored into file test.sign.

```
hashgen.exe check test.sign

../security2.fdb: Success
```

In this case utility hashgen is checking hash from file test.sign. The result of checking is success.

13 Functional

13.1 Superclassic architecture

Red Database 2.5 is supporting multi-threaded classic server – superclassic architecture. In this mode one process is starting and all attachments are working parallel.

To use another features as:

- Java external functions;
- Full text search.

Your need to install JDK 1.6. Also you must execute Init.sql to have in target database file necessary service tables. This file is located in root installation directory of Red Database 2.5.

13.2 Java procedures and triggers

For more information see «Manual for external procedures» on our web site http://www.red-soft.biz.

13.3 Full Text Search

For more information see «Manual for full text search» on our web site http://www.red-soft.biz.

Appendix A - ODS changing

DDL operations access

New privileges values of field RDB$PRIVILEGE in table RDB$USER_PRIVILEGES:

```
C – CREATE,
L – ALTER,
T – ALTER ANY,
O – DROP,
P – DROP ANY.
```

RDB$RELATION_NAME stores object type name (RDB$TABLE, RDB$PROCEDURE and etc.).

RDB$OBJECT_TYPE stores object type.

New field RDB$OWNER_NAME is added into system tables RDB$EXCEPTIONS, RDB$GENERATORS, RDB$FUNCTIONS, RDB$FIELDS, RDB$FILES.

RDB$SECURITY_CLASSES stores global objects types (OBJ$TABLE, OBJ$PROCEDURE etc).

DML operations access

If user adds record with value S in RDB$USER_PRIVILEGES into RDB$PLIVILEGE, user RDB$USER can access to select values of field RDB$FIELD_NAME from relation RDB$RELATION_NAME.

Default roles

Defined access rights for roles SYSADMIN, SECADMIN, PUBLIC. Global roles support.

View USERS in security2.fdb was improved to allow SECADMIN work with users:

```
CREATE VIEW USERS (
     USER_NAME, SYS_USER_NAME, GROUP_NAME, UID, GID,
PASSWD,    PRIVILEGE, COMMENT, FIRST_NAME, MIDDLE_NAME,
LAST_NAME, FULL_NAME, HASH_ALG, HASH_SIZE) AS
SELECT
     RDB$USER_NAME, RDB$SYS_USER_NAME, RDB$GROUP_NAME,
RDB$UID,   RDB$GID, RDB$PASSWD,
     RDB$PRIVILEGE,RDB$COMMENT,
     RDB$FIRST_NAME, RDB$MIDDLE_NAME, RDB$LAST_NAME,
     COALESCE (RDB$first_name || _UNICODE_FSS ' ', '') ||
     COALESCE (RDB$middle_name || _UNICODE_FSS ' ', '')
||
     COALESCE (RDB$last_name, ''),
     RDB$HASH_ALG,
     RDB$HASH_SIZE
```

```
FROM RDB$USERS
WHERE
     CURRENT_USER = 'SYSDBA' OR CURRENT_ROLE = 'SECADMIN'
     OR CURRENT_USER = RDB$USERS.RDB$USER_NAME;
/*   Access rights */
GRANT ALL ON RDB$USERS to VIEW USERS;
GRANT SELECT ON USERS to PUBLIC;
GRANT UPDATE(PASSWD, GROUP_NAME, UID, GID, FIRST_NAME,
MIDDLE_NAME, LAST_NAME)
     ON USERS TO PUBLIC;
GRANT INSERT ON USERS to SECADMIN;
GRANT DELETE ON USERS to SECADMIN;
```

Administration functions access (services)

Rights on Services running is stored in table RDB$USER_PRIVILEGES of DB security2.fdb.

Combine roles

New field RDB$PROCEDURE_CONTEXT of type SMALLINT is added to store procedure context in table RDB$PROCEDURES. The value of this field must be equal 0 (OWNER) or 1 (CALLER).

Multifactor authentication

Field RDB$HASH_ALG containing hash algorithm name is added into table RDB$USERS of security2.fdb. This algorithm will be used to verify password. If field doesn't contain value old hash algorithm is used.

Login policies

The next table in security2.fdb is created to store user policies:

```
CREATE TABLE RDB$POLICIES (
     RDB$POLICY_NAME RDB$POLICY_NAME NOT NULL PRIMARY
KEY,
     RDB$PSWD_NEED_CHAR RDB$COUNT,
     RDB$PSWD_NEED_DIGIT RDB$COUNT,
     RDB$PSWD_NEED_DIFF_CASE RDB$BOOL,
     RDB$PSWD_MIN_LEN RDB$LENGTH,
     RDB$PSWD_VALID_DAYS RDB$DAY_COUNT,
     RDB$PSWD_UNIQUE_COUNT RDB$COUNT,
     RDB$MAX_FAILED_COUNT RDB$COUNT,
     RDB$MAX_SESSIONS RDB$COUNT,
     RDB$MAX_IDLE_TIME RDB$MAX_IDLE_TIME,
     RDB$AUTH_FACTORS RDB$AUTH_FACTORS
);
```

Default policy is RDB$POLICY_NAME == 'DEFAULT'.

Field RDB$AUTH_FACTORS contains string with logical expression. This value consists of symbolic definitions of authentication factors.

The next fields are added into RDB$USERS to store failed user authentication count, password install time, policy name defined to user, hash algorithm name, time after which user can access database:

```
RDB$PASSWD_TIME TIMESTAMP,
RDB$FAILED_COUNT RDB$COUNT,
RDB$POLICY_NAME RDB$POLICY_NAME REFERENCES RDB$POLICIES,
RDB$HASH_ALG        RDB$HASH_ALG,
RDB$ACCESS_TIME RDB$ACCESS_TIME
```

The next table is added to store old passwords hashes

```
CREATE TABLE RDB$PASSWD_HISTORY (
     RDB$KEY_ID   INTEGER PRIMARY KEY NOT NULL,
     RDB$USER_NAME RDB$USER_NAME REFERENCES RDB$USERS,
     RDB$PASSWD RDB$PASSWD,
     RDB$HASH_ALG RDB$HASH_ALG
);
```

Audit journal analyzer

New field RDB$ADAPTER of char[31] type is added into system table RDB$RELATIONS.