New features of Firebird ported from RedDatabase

Roman Simakov, director of system development department
History of Red Database
Security features

- Cryptographic plugin
- Multi-factor authentication
- Cumulative roles
- DML access control
- DDL access control
- Service access control
- Record filtering

Functional features

- Java Stored Procedures
- Full Text Search
- LDAP/AD integration
- StandBy cluster (engine-level replication)
Automated Information System of Federal Service for Officers of Justice of Russia

- AIS is installed and work in 85 regional departments and in the main office of FSOJ of Russia
- Total amount of Red Database installations are about 2720, i.e. every city of Russia has one or several Red Database servers
- AIS handles more than $10^9$ documents per year
- AIS works in 24/7 mode
- Some databases more than 1TB and a lot of data goes to archived set of database files
- 100x of concurrent connections
- 100,000x transactions per hour
Example of load
Development process (schema)

- **Firebird repository**
  - direct commits
  - conflict commits
  - merged commits
- **Developer**
  - resolved commits
- **Red Soft repository**
  - local commits
  - direct commits
DDL access control (CORE-735 since 2003)

**Problem:** Previously **anyone** who connected to database could create a table

**Syntax**

```sql
GRANT CREATE <OBJECT> TO [USER | ROLE] <user/role name> [WITH GRANT OPTION]
GRANT ALTER ANY <OBJECT> TO [USER | ROLE] <user/role name> [WITH GRANT OPTION]
GRANT DROP ANY <OBJECT> TO [USER | ROLE] <user/role name> [WITH GRANT OPTION]
REVOKE [GRANT OPTION FOR] CREATE <OBJECT> FROM [USER | ROLE] <user/role name>
REVOKE [GRANT OPTION FOR] ALTER ANY <OBJECT> FROM [USER | ROLE] <user/role name>
REVOKE [GRANT OPTION FOR] DROP ANY <OBJECT> FROM [USER | ROLE] <user/role name>
```

Where **OBJECT** can be:

- TABLE, VIEW, PROCEDURE, FUNCTION, PACKAGE, GENERATOR, SEQUENCE, DOMAIN,
- EXCEPTION, ROLE, CHARACTER SET, COLLATION, FILTER
DDL access control: Simple example

**GRANT CREATE TABLE TO Joe**

*ALTER possible because of Joe is owner of JoeT*

**GRANT ALTER ANY TABLE TO Joe**

*Triggers and indices re-use table privileges*

**REVOKE CREATE TABLE FROM Joe**

**CREATE TABLE JoeT (I INTEGER)**

**ALTER TABLE JoeT ...**

**ALTER TABLE BobT ...**

**CREATE INDEX ON JoeT ...**

**CREATE INDEX ON BobT ...**

**CREATE TRIGGER ON T ...**

**ALTER TRIGGER ON T ...**
DDL access control: special form for managing database

GRANT CREATE DATABASE TO [USER | ROLE] <user/role name>

GRANT ALTER DATABASE TO [USER | ROLE] <user/role name> [WITH GRANT OPTION]

GRANT DROP DATABASE TO [USER | ROLE] <user/role name> [WITH GRANT OPTION]

REVOKE CREATE DATABASE FROM [USER | ROLE] <user/role name>

REVOKE [GRANT OPTION FOR] ALTER DATABASE FROM [USER | ROLE] <user/role name>

REVOKE [GRANT OPTION FOR] DROP DATABASE FROM [USER | ROLE] <user/role name>

ALTER DATABASE permissions is used to check following actions:

1) Altering database itself
2) Commenting on database or db level triggers
3) Managing db level triggers
4) Managing shadows
5) Direct editing RDB$FILES system table
Cumulative roles (CORE-1815 since 2008)

Now you can grant role to another role except circle references

Syntax:

```
GRANT [DEFAULT] <role name> TO [USER | ROLE] <user/role name> [WITH ADMIN OPTION]
REVOKE [DEFAULT] <role name> FROM [USER | ROLE] <user/role name> [WITH ADMIN OPTION]
```

**ADMIN OPTION** allows grantees to grant the role to another user or role

1) WORKER->MANAGER->Joe
2) WORKER->MANAGER=Joe
3) WORKER=MANAGER->Joe
4) WORKER=>MANAGER=Joe

Syntax:

```
RDB$ROLE_IN_USE(role_name varchar(32)) RETURNS BOOLEAN
```

To get a list of currently active roles you can run:

```
SELECT * FROM RDB$ROLES WHERE RDB$ROLE_IN_USE(RDB$ROLE_NAME)
```
Cumulative roles: DEFAULT explanation

DEFAULT means that this role will be used even if role does not specify explicitly.

DEFAULT WITHOUT DEFAULT

ROLE A

ROLE B

EMPTY ROLE

A

B

C

D

E
SQL SECURITY (SQL STANDARD 2003, 2011)

Syntax

```sql
CREATE TABLE <TABLENAME> (...) [SQL SECURITY {DEFINER | INVOKER}]

ALTER TABLE <TABLENAME> ... [{ALTER SQL SECURITY {DEFINER | INVOKER} | DROP SQL SECURITY}]

CREATE [OR ALTER] TRIGGER <TRIGGERNAME> ... [SQL SECURITY {DEFINER | INVOKER} | DROP SQL SECURITY] [AS ...]

CREATE [OR ALTER] FUNCTION <FUNCTIONNAME> ... [SQL SECURITY {DEFINER | INVOKER}] AS ...

CREATE [OR ALTER] PROCEDURE <PROCEDURENAME> ... [SQL SECURITY {DEFINER | INVOKER}] AS ...

CREATE [OR ALTER] PACKAGE <PACKAGENAME> [SQL SECURITY {DEFINER | INVOKER}] AS ...
```
connect 'localhost:/tmp/db.fdb' user sysdba password 'masterkey';
set term ^;
cREATE FUNCTION f() RETURNS INT
AS
BEGIN
RETURN 3;
END^^
set term ;^;

CREATE TABLE t (i INTEGER, c COMPUTED BY (i + f())) SQL SECURITY DEFINER;
INSERT INTO t VALUES (2);
GRANT SELECT ON TABLE t TO USER us;
GRANT EXECUTE ON FUNCTION f TO USER us;
COMMIT;

CONNECT 'localhost:/tmp/db.fdb' USER us PASSWORD 'pas';
SELECT * FROM t;
connect 'localhost:/tmp/db.fdb' user sysdba password 'masterkey';
set term ^;
create function f (i integer) returns int sql security definer
as
begin
    insert into t values (:i);
    return i + 1;
end^
set term ;^  
grant execute on function f to user us;
grant insert on table t to user us;
commit;

connect 'localhost:/tmp/db.fdb' user us password 'pas';
select f(3) from rdb$database;
connect 'localhost:/tmp/db.fdb' user sysdba password 'masterkey';
set term ^;
create procedure p (i integer) sql security definer
as
begin
    insert into t values (:i);
end^
set term ;^
grant execute on procedure p to user us;
grant insert on table t to user us;
grant insert on table t to procedure p;
commit;

connect 'localhost:/tmp/db.fdb' user us password 'pas';
execute procedure p(1);
connect 'localhost:/tmp/db.fdb' user sysdba password 'masterkey';
create table tr (i integer) sql security definer;
create table t (i integer);
set term ^;
create trigger tr_ins for tr after insert sql security definer as
begin
  insert into t values (NEW.i);
end^
set term ;^
grant insert on table tr to user us;
grant insert on table t to user us;
commit;

connect 'localhost:/tmp/db.fdb' user us password 'pas';
insert into tr values(2);
connect 'localhost:/tmp/db.fdb' user sysdba password 'masterkey';
create table t (i integer);
set term ^;
create package pk sql security definer
as
begin
  function f(i integer) returns int;
end^
create package body pk
as
begin
  function f(i integer) returns int
  as
  begin
    insert into t values (:i);
    return i + 1;
  end
end^
set term ;^
grant execute on package pk to user us;
grant insert on table t to user us;
commit;

connect 'localhost:/tmp/db.fdb' user us password 'pas';
select pk.f(3) from rdb$database;
Thanks!

visit: www.red-soft.ru
roman.simakov@red-soft.ru