Firebird user privileges and object ownership

Vítězslav Švejdar, August 25, 2013

1 System tables, users and user privileges in Firebird

If simplified drastically, a Firebird database, as well as any other database, can be viewed as a set of user defined tables. There could be, for example, a table invoices where records correspond to invoices sent to customers, and another table customers in which all those customers are listed. However, there are other database objects besides tables, like views, triggers, or (stored) procedures. Firebird maintains also some *system tables* whose names usually begin with rdb$ or mon$. The same set of system tables is present in every database since the moment it was created. For example, rdb$procedures is the table of all (stored) procedures; rdb$relations is the table of all tables; rdb$database is a simple table with one record only, containing a brief information about the entire database. The table rdb$relations contains an information about all tables in the database including the system tables. Thus it has a record in it containing an information about the table rdb$relations itself.

To control access to data in databases, Firebird has the concepts of *users*, object ownership, database ownership and *user privileges*. Each installation of Firebird has a set of database users. These users can connect to databases maintained by that installation. Information about users is stored in the so called security database, one per installation. Since Firebird 2.0, security database is visible as the file security2.fdb in the home directory of Firebird installation. This file has the format of normal database. However, it is never connected to or accessed via sql statements like normal databases; there are api calls and a command line utility gsec to add, modify, and delete users. When a user connects to a database, his password is first verified against the security database of the installation. Then his login name determines what he can and what he cannot do with the database to which he is connected. So in contrast to passwords, user names must occur somewhere in the connected database.

Some database objects, namely tables, procedures, and roles, have an *owner*. The name of the user who owns a table (procedure, role) is stored in the field rdb$owner_name of the record in the table rdb$relations (rdb$procedures, rdb$roles) corresponding to the table (procedure, role) in question. A *database owner* is defined as the user who owns the table rdb$database. The database owner is supposed to simultaneously be an owner of all system objects, i.e. system tables and system roles. As far as I know, no system procedures exist.

2 Granting privileges and changing object ownership

The grant sql statement is used to associate privileges with users. For example, and if the connected user has the appropriate privileges, the statements

```
grant select on invoices to eagle with grant option;
grant execute on procedure sp_expenses to eagle;
```

associate the privilege to select from a table, and to execute a procedure, with user eagle. ‘With grant option’ allows eagle to grant the given privilege to other users. Privileges can be granted not only to users, but also to views, procedures, roles, and triggers. Information about privileges is stored in the system table rdb$user_privileges. The grant sql statement results in addition or modification of a record in this table, while the revoke sql statement may result in deletion of a record. The table rdb$user_privileges normally contains at least tens (rather hundreds) records because some privileges are granted automatically to the owner of an object when the object is created, and some privileges exist since the moment the database was created.

There is no dedicated sql statement to change ownership of an object. However, Firebird in many cases allows direct modification of system tables using update, insert, or delete. There are rules that determine who can change what. However, these rules are not strict enough to prevent you from creating inconsistent data, and thus care is needed when writing to system tables. Consider the following example. User A, the owner of table T that he created, says

```
update rdb$relations set rdb$owner_name='B'
where rdb$relation_name='T';
```

This statement changed ownership of table T, and user A cannot revert this change. The ownership change did not affect privileges in rdb$user_privileges. So user A can still work with data in table T. Moreover, he can still grant and revoke privileges on table T. We classify his privileges on table T as *invalid* but they still work. He can for example say

```
revoke all on T from A;
```

thus deleting his own privileges on T. After committing this statement (and after the next connection to this database is established), *nobody* can read data from table T, and neither A nor B can do anything about that.

3 User substitution

We see that a direct modification of system tables is a somewhat risky thing, but it is necessary if there is a need to change the database owner, or more generally, if there is a need to substitute one database user for another. Such a change is a task of the script privprocs.sql and the associated utility DBUSubst.exe. This utility is a wrapper for the script privprocs.sql and for a few other scripts, among them invalidpriv.sql. The script invalidpriv.sql tries to detect those invalid privileges that would threaten the process of user substitution. These files can be downloaded from http://www1.cuni.cz/~svejdar/?s=oact; it is either possible to use DBUSubst.exe without much thinking, or to to run several sql statements "manually", as described in the script privprocs.sql.

Comments in the script invalidpriv.sql also contain a simple table, found experimentally, that specifies who can and who cannot change ownership and grant or revoke privileges. E.g., if no privilege on table T exists in rdb$user_privileges, like in our example above, user sysdba can correct this situation. Still, sysdba cannot change ownership of a table he does not own.

Our process of database owner change never directly inserts data into the table rdb$user_privileges nor it updates that table; all changes are made through the grant statement. In the end however, there are some invalid lines in that table that cannot be revoked and must be deleted manually.

One can sometimes google out that to change the database owner it is sufficient to change the owner of the table rdb$relations, or to change the owner of all system object, and then do backup-restore of the database under the name of the new owner. It is not the case! It is true that changing the owner of the table rdb$relations and then doing backup-restore under the name of the new owner changes the ownership of all remaining system objects. Which is not enough.

The whole process of database user substitution is inspired by and partly based on Thomas Steinmaurer's ideas, but is more general and does not need Delphi utility.