Role Based Security

A feature of Oracle Rdb

By Ian Smith
Oracle Rdb Relational Technology Group
Oracle Corporation

The examples in this article use SQL language from Oracle Rdb V7.0 and later versions.

Role Based Security

Someone asked me recently when Oracle Rdb would implement Role based security. In surprise I replied that Rdb had always had role-based security. Since this topic might be of interest to other readers I decided this month to describe how Rdb implements role based security on OpenVMS and what advantages that has for applications.

Users and Roles

In most businesses there are personnel who are authorized to access selected data, or perform certain functions. For instance a manager has access to payroll and performance data for his department, the supervisor has access to performance data only for his project group, and the employee has access limited to their own data. Therefore, for our example we have three classes or groups of users: MANAGER, SUPERVISOR and EMPLOYEE. A non-employee (such as a contractor) would be unclassified, or specially classified so that they could be denied access. We refer to these classes as roles.

Different people perform these roles within the database. For instance, Janet is the manager, Juan is the supervisor, and Jack is an employee. We refer to these people as database *users*.

These users are assumed to have a unique identity on the computing network, let us use their names as the system user identification: JANET, JUAN and JACK. Within an OpenVMS system these names JANET, JUAN and JACK would be OpenVMS usernames and could be used by these people to access the computer system and ultimately allow them access to the database.

The database administrator could grant differing levels of access to each user. For instance, JANET (being a manager) would be granted all access to all tables and procedures in the database, JUAN (being a supervisor) would be granted no access to tables, but granted EXECUTE access to procedures which allow him to inquire about his departments performance, and finally JACK would be granted only EXECUTE access to the SHOW_MY_DATA procedure.

```
SQL> grant all on table * to JANET;
SQL> grant all on view * to JANET;
SQL> grant execute on procedure * to JANET, JUAN;
SQL> grant execute on procedure SHOW_MY_DATA to JACK;
```

This is how many systems are run today, but it assumes stability in the assignment of responsibility within the company. There is also a maintenance task associated with this user-based model.

As new employees arrive their name must be added to the SHOW_MY_DATA procedure. If JUAN was promoted and moved to a different section of the company his access would be revoked from all the special procedures he once used.

Now what happens if JANET takes some extended leave (say to train and then attend the Olympic games), for the duration JUAN is given access as both a manager and supervisor until her return? The poor database administrator now has to keep track of which access is granted to JUAN because he was a supervisor and what is granted temporarily because he is an acting manager.

This is where role based security comes into play. On OpenVMS you would create three roles, or rights identifiers using the OpenVMS AUTHORIZE utility: MANAGER, SUPERVISOR and EMPLOYEE. These would be granted to JANET, JUAN and JACK appropriately again using AUTHORIZE.

```
$ run sys$system:authorize
UAF> add/id manager
UAF> add/id supervisor
UAF> add/id employee
UAF> grant/id manager JANET
UAF> grant/id supervisor JUAN
UAF> grant/id employee JACK
```

The database administrator would no longer grant or manage access for each user (although that is still possible), instead security access to tables, views, functions and procedures is granted to roles. That is, a MANAGER can access all tables and procedures, a SUPERVISOR can access data only through a set of procedures and an EMPLOYEE can access only the SHOW_MY_DATA procedure.

```
SQL> grant all on table * to MANAGER;
SQL> grant all on view * to MANAGER;
SQL> grant execute on procedure * to MANAGER, SUPERVISOR;
SQL> grant execute on module * to MANAGER, SUPERVISOR;
SQL> grant execute on procedure SHOW_MY_DATA to EMPLOYEE;
```

Note: the wildcard * can be used to apply the GRANT and REVOKE command to all modules and procedures. In practice these routines will probably be grouped into a small number of stored modules and therefore GRANT could explicitly name those modules.

Now when JANET heads to the Olympics a single grant of MANAGER to JUAN achieves the affect of allowing enhanced access rights. Now that JUAN has both MANAGER and SUPERVISOR roles he can perform both functions. When JANET returns it requires a single revoke of the MANAGER role leaving JUAN with just the SUPERVISOR role.

Also note that FRANCIS (a contract programmer) is not granted the EMPLOYEE role so she cannot even access the database nor execute the SHOW_MY_DATA procedure. So no special action needs to be taken for unclassified users.

There are several benefits of this role based scheme; it makes security maintenance much easier, and there is less user information stored in the database.

Consider the procedure SHOW_MY_DATA that now has a single access control entry (ACE), instead of one for each employee that would be stored as a long access control list (ACL). The smaller number of entries for role based security means that Rdb uses less space to store the ACL data, uses less I/O to access the information when needed, and less CPU time to scan for matching ACE entries during privilege checking. It also allows the database to be distributed to other systems and used by different sets of users, by simply defining and granting the set of roles on the new system.

Roles and Groups

OpenVMS supports the notion of a group that allows users to share a common root UIC (user identification code). This functionality has been available since the earliest versions of OpenVMS. In many respects this functionality was superseded by the more modern rights identifier system but is still used by some OpenVMS sites.

It is possible to implement these examples using the UIC group (e.g. [group,*]) syntax in the GRANT and REVOKE commands. However, the rights identifiers will be easier to read.

Oracle Rdb V7.1 extends the traditional Rdb users and roles model based on the OpenVMS operating system and allows user and role definitions stored within the database itself. We will examine this topic in a future issue.

Technical Tips: Restricting Database Creation

Question: How do I restrict Rdb database creation to just the database administrators? I don't want databases in production that have not been approved by the database administrators and technical staff.

Answer: This is a common request in environments that must control database creating, object naming and other design standards.

As it turns out this is quite simple with Oracle Rdb. Since Rdb version 4.1 it has been possible to define the system wide logical name RDBVMS$CREATE_DB. Once defined no user is allowed to create a database unless they are granted the OpenVMS rights identifier RDBVMS$CREATE_DB. In essence they are granted the "create database" role.

The OpenVMS system manager must define this system wide logical name (a task which requires OpenVMS privileges to perform or subvert) and grant the RDBVMS$CREATE_DB rights identifier to each database administrator.

This feature is described briefly in the *Oracle Rdb Installation Guide* and in more detail with examples in the *Oracle Rdb Guide to Database Design and Definition*.

Oracle Rdb
Role Based Security, Revision 1
May 2002

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com