Access Control Features in Oracle

CS 590U

April 7, 2005

Ji-Won Byun

Access Control Features in Oracle

- Broadly, Oracle supports five features for access controls.
  1. Privileges
  2. Views
  3. Stored Procedures
  4. Roles
  5. Virtual Private Databases (VPD)

Privileges

- System privileges
  - the right to perform a particular action or to perform an action on any schema objects of a particular type
  - E.g., ALTER DATABASE and SELECT ANY TABLE
- Object privileges
  - the right to perform a particular action on a specific schema object such as tables, views, procedures and types
  - E.g., SELECT, INSERT

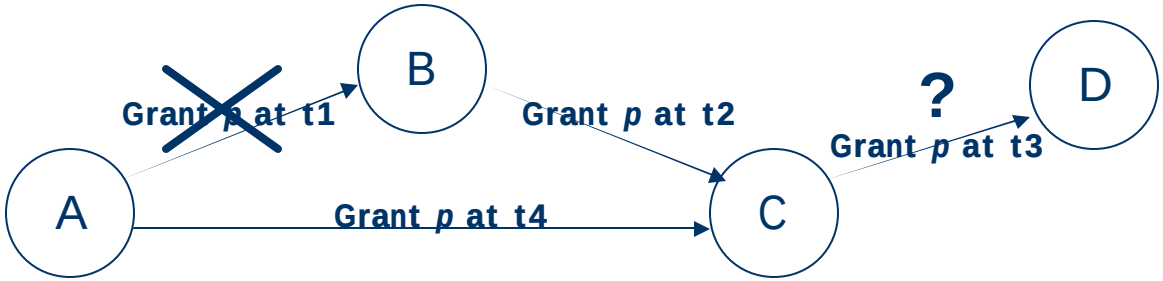
Grant/Revoke Privileges

- System privileges
  - GRANT Create Table to Bob [WITH ADMIN OPTION]
  - REVOKE Create Table from Bob
  - Users with ADMIN OPTION can not only grant the privilege to other users, but also revoke the privilege from any user.
- Object privileges
  - GRANT Select ON table1 to Bob [WITH GRANT OPTION]
  - REVOKE Select ON table1 from Bob
  - Users who revokes a particular object privileges must be the direct grantor of the privilege.
  - There is always a cascading effect when an object privilege is revoked.

Some issues (1)

- There is no column privilege for SELECT.
  - Column privileges exist for INSERT and UPDATE.
  - Coarse-grained control for SELECT.

- There is no timestamp for privileges.
  - Revocation (i.e., cascading effect) is also coarse.



Some issues (2)

- An UPDATE statement does NOT require SELECT privileges.
  - Convicts (name, sentence)
  - Bob has UPDATE privilege on Convicts but no SELECT.
  - To find out if Alice is in the Convict table,
    - UPDATE Convicts SET name = name WHERE name = 'Alice';
    - If the table is updated, then Bob knows that Alice is a convict.
  - To find out the sentence of Alice,
    - UPDATE Convicts SET name = name WHERE name = 'Alice' AND sentence = guessed_value;
    - If the table is updated, then the guess is correct.

Views

- To create a view
  - the user must have been explicitly (i.e., not through roles) granted one of SELECT, INSERT, UPDATE or DELETE object privileges on all base object underlying the view or corresponding system privileges.
- To access the view,
  - The creator must have the proper privilege to the underlying base tables.
- To grant access to the view
  - The creator must have been granted the privileges to the base tables with Grant Option.

Stored Procedures

- Two types of procedures in terms of access control
  - Definer's right procedures
  - Invoker's right procedures

- Definer's right procedures
  - A user of a definer's right procedure requires only the privilege to execute the procedure and no privileges on the underlying objects that the procedure access.
  - Fewer privileges have to be granted to users, resulting in tighter control of database access.
  - At runtime, the privileges of the owner are always checked.

Issues (1)

- A user with Create Procedure privilege can effectively share any privilege he/she owns with other users without grant option.
  - Just create a definer's right procedure that uses a privilege.
  - Then grant Execute privilege to others.
  - Create Procedure privilege is too powerful.
  - Possible fix: when one grants Execute privilege, the system should check if all the necessary privileges are in fact grantable.

Stored Procedures

- Invoker's right procedures
  - A user of an invoker's right procedure needs privileges on the objects that the procedure accesses.
  - Invoker's right procedures can prevent illegal privilege sharing.
  - More like function calls in operating systems.

Issues (2)

- Invoker's right procedures can be embedded with Trojan Horse.
  - Users of invoker's right procedures can blindly run malicious procedures.

More Issues (3)

- A procedure's owner (i.e., definer) must have all the necessary object privileges for referenced objects.
  - Those privileges must have given directly, not through roles.
  - At runtime, the owner's privileges are always checked.
  - This applies to even invoker's right procedures.
  - What this means is that if definers must have the necessary privileges as long as the procedures are in use.
  - What if temporary contractors define procedures?

More Issues (4)

- Confusing (seems inconsistent) if definer's right procedures and invoker's right procedures are used together.
  - def_def: definer's procedure invoking definer's procedure
  - inv_inv: invoker's procedure invoking invoker's procedure
  - def_inv: definer's procedure invoking invoker's procedure
  - inv_def: invoker's procedure invoking definer's procedure

Def and Def

- def_def
  - Carl creates def_def using bob's def procedure.

```
Create procedure def_def As
   Begin
      bob.def(); // insert into alice.t1
   End;
```

Doris needs Execute on def_def

Run as Carl:
Carl needs Execute on Bob's def

Run as Bob:
Bob needs Insert on Alice's t1

  - Then grant Execute to Doris and Doris runs def_def.

  - The invoker needs only Execute privilege on def_def.
  - All definers must have the necessary privileges.

Inv and Inv

- inv_inv
  - Carl creates inv_inv using bob's inv procedure.

```
Create procedure inv_inv
        Authid Current_User As
  Begin
    bob.inv(); // insert into alice.t1
  End;
```

Doris needs Execute on inv_inv

Run as Doris:
Doris needs Execute on Bob's inv
Carl needs Execute on Bob's inv

Run as Doris:
Doris needs insert on Alice's t1
Bob needs Insert on Alice's t1
Carl does not need Insert

  - Then grant Execute to Doris and Doris runs inv_inv.
  - Very restrictive.

Def and Inv

- def_inv
  - Carl creates def_inv using bob's inv procedure.

```
Create procedure def_inv As
    Begin
        bob.inv(); // insert into alice.t1
    End;
```

Doris needs Execute on def_inv

Run as Carl:
Carl needs Execute on Bob's inv

Run as Carl:
Bob needs Insert on Alice's t1
Carl needs Insert on Alice's t1
Doris does NOT need insert on Alice's t1

  - Then grant Execute to Doris and Doris runs def_inv.

Inv and Def

- inv_def
  - Carl creates inv_def using bob's def procedure.

```
Create procedure inv_def
        Authid Current_User As
  Begin
    bob.def(); // insert into alice.t1
  End;
```

Doris needs Execute on inv_def

Run as Doris:
Carl needs Execute on Bob's def
Doris does NOT need Execute on Bob's def??

Run as Bob:
Bob needs Insert on Alice's t1

  - Then grant Execute to Doris and Doris runs inv_def.

Issues

- Very confusing
- It does not seem that all consistent.
- Need a formal

Administering roles

- Four system privileges
  1. Create Role
  2. Drop Any Role
  3. Grant Any Role
  4. Alter Any Role

- Admin Option
  - When a role is granted with “Admin Option”, the grantee can grant, alter or drop the role.
  - When a user creates a role, the creator is granted the role with “Admin Option”.

Five role authorization types (1)

1. By user ID
   - CREATE ROLE clerk;
   - SET ROLE clerk;

2. By password
   - CREATE ROLE manager IDENTIFIED BY password;
   - SET ROLE manager IDENTIFIED BY password;

3. By application
   - CREATE ROLE admin_role IDENTIFIED USING hr.admin;
   - *admin_role* can be enabled only by a module inside the authorized package (*hr.admin*).

Five role authorization types (2)

4. By an external source (e.g. OS or network)

   - CREATE ROLE acc_role IDENTIFIED EXTERNALLY;

   - When a user logs into the database, the operating system identifies the database roles to be enabled for the user. The OS manages/stores what roles to enable for each user.

5. By an enterprise directory service

   - CREATE ROLE supervisor IDENTIFIED GLOBALLY;

   - *supervisor* is a global role which can be authorized only to global users by an enterprise directory service.

Why use Roles?

- Two main purposes

  1. To manage the privileges for a user group (User roles)
     - DBA creates a role for a group of users with common privilege requirements. DBA grants all the required privileges to a role and then grants the role to appropriate users.

  2. To manage the privileges for an application (Application roles)
     - DBA creates a role (or a set of roles) for an application and grants it all necessary privileges to run the application. Then DBA grants the application role to appropriate users.

Application Roles

- How can we secure application roles? That is, we want application roles to be used only through the associated applications.
  - Use a password for the application role and embed the password in the application. Then the role can be enabled only by the application.
  - Associate the application role with the application (i.e., a package). Then the role can be enabled only by a module in the application.

Security Domain

- A user's security domain includes:
  - The privileges on all schema objects in his own schema
  - The privileges granted to the user
  - The privileges of roles (both directly and indirectly) granted to the user that are currently enabled.
  - The privileges and roles granted to the PUBLIC.

User Assignments

- To grant a role to a user, one needs to have the "Grant Any Role" system privilege or have been granted the role with "Admin Option".
  - GRANT ROLE clerk TO Alice;
- To revoke a role from a user, one needs to have the "Grant Any Role" system privilege or have been granted the role with "Admin Option".
  - REVOKE ROLE clerk FROM Alice;
- Users cannot revoke a role from themselves.

Permission Assignments

- To grant a privilege to a role, one just needs to be able to grant the privilege.
  - GRANT insert ON table1 TO clerk;
- To revoke a privilege from a role, one just needs to be able to revoke the privilege.
  - REVOKE insert ON table1 FROM clerk;
- No special admin privilege is required.
  - It can be a problem since one can make a role unusable by granting many roles to the role to exceed MAX_ENABLED_ROLES.
- “Grant Option” is not valid when granting an object privilege to a role.
  - To prevent the propagation of object privileges through roles.

Effective times

- Granting/revoking a role to/from users (i.e., UA)
  - After a current user session issues a “SET ROLE” statement or a new user session is created.
- Granting/revoking a privilege to/from a role (i.e., PA)
  - Immediate.
- Dropping a role
  - Immediate.
- Try with Oracle!

Default roles

- When a user logs in, all default roles are enabled.
- When a user is created, the default role setting is ALL.
- When a role is assigned to a user, the role is added to the default roles.
- The default roles can be changed.
  - ALTER USER alice DEFAULT ROLE clerk, cashier;

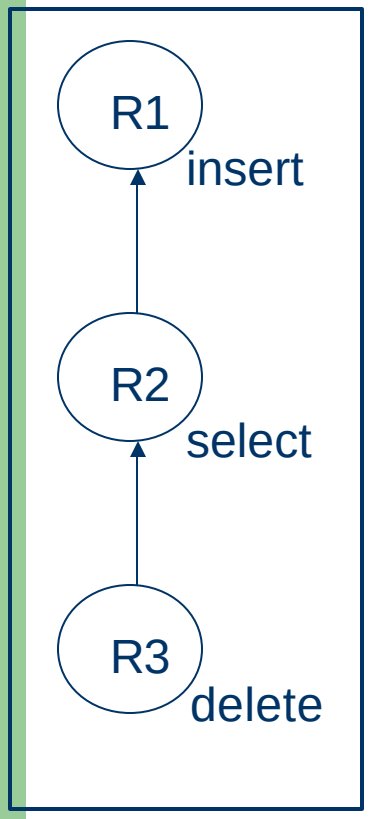
Activation/Deactivation of roles

- In order to enable a role, a user must have been granted the role.
- Three types of “Set Role” statements
  - SET ROLE clerk;
  - SET ROLE NONE;
  - SET ROLE ALL EXCEPT clerk;
  - One cannot disable roles individually.
- The number of roles that can be concurrently enabled is limited by “MAX_ENABLED_ROLES”.
  - Initialization parameter

Role hierarchy (1)

- Any role can be granted to another role.
  - A role cannot be granted to itself.
  - A role cannot be granted circularly. (e.g., a role x cannot be granted to a role y if y has been already granted to x.)
- A role granted to another role is called an indirectly granted role.
  - It can be explicitly enabled or disabled for a user.
  - Whenever a role that contains other roles is enabled, all indirectly granted roles are enabled as well.

Role hierarchy (2)



- R1 is granted to Bob;
- SET ROLE R1;
  - R1, R2, R3 are all enabled.
- SET ROLE R2;
  - R2, R3 are enabled.
- SET ROLE R3;
  - R3 is enabled.
- When a senior role is activated, all junior roles are activated.
  - {R1}, {R1, R2}, {R2} cannot be activated.

Questions