Skip Headers

Oracle9*i* Database Concepts
Release 2 (9.2)
Part Number A96524-01

Home Book List Contents Index Master Index Feedback

Previous Next

23
Privileges, Roles, and Security Policies

This chapter explains how you can control users' ability to execute system operations and to access schema objects by using privileges, roles, and security policies. The chapter includes:

Introduction to Privileges

A **privilege** is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include the right to:

- Connect to the database (create a session)
- Create a table
- Select rows from another user's table
- Execute another user's stored procedure

You grant privileges to users so these users can accomplish tasks required for their job. You should grant a privilege only to a user who absolutely requires the privilege to accomplish necessary work. Excessive granting of unnecessary privileges can compromise security. A user can receive a privilege in two different ways:

- You can grant privileges to users explicitly. For example, you can explicitly grant the privilege to insert records into the `employees` table to the user `SCOTT`.
- You can also grant privileges to a role (a named group of privileges), and then grant the role to one or more users. For example, you can grant the privileges to select, insert, update, and delete records from the `employees` table to the role named `clerk`, which in turn you can grant to the users `scott` and `brian`.

Because roles allow for easier and better management of privileges, you should normally grant privileges to roles and not to specific users.

There are two distinct categories of privileges:

- System privileges

- Schema object privileges

See Also:

Oracle9i Database Administrator's Guide for a complete list of all system and schema object privileges, as well as instructions for privilege management

System Privileges

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges. There are over 60 distinct system privileges.

Grant and Revoke System Privileges

You can grant or revoke system privileges to users and roles. If you grant system privileges to roles, then you can use the roles to manage system privileges. For example, roles permit privileges to be made selectively available.

Note:

In general, you grant system privileges only to administrative personnel and application developers. End users normally do not require the associated capabilities.

Use either of the following to grant or revoke system privileges to users and roles:

- Oracle Enterprise Manager Console
- The SQL statements `GRANT` and `REVOKE`

Who Can Grant or Revoke System Privileges?

Only users who have been granted a specific system privilege with the `ADMIN OPTION` or users with the system privileges `GRANT ANY PRIVILEGE` or `GRANT ANY OBJECT PRIVILEGE` can grant or revoke system privileges to other users.

Schema Object Privileges

A **schema object privilege** is a privilege or right to perform a particular action on a specific schema object:

- Table
- View
- Sequence
- Procedure
- Function
- Package

Different object privileges are available for different types of schema objects. For example, the privilege to delete rows from the `departments` table is an object privilege.

Some schema objects, such as clusters, indexes, triggers, and database links, do not have associated object privileges. Their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the ALTER ANY CLUSTER system privilege.

A schema object and its synonym are equivalent with respect to privileges. That is, the object privileges granted for a table, view, sequence, procedure, function, or package apply whether referencing the base object by name or using a synonym.

For example, assume there is a table jward.emp with a synonym named jward.employee and the user jward issues the following statement:

```
GRANT SELECT ON emp TO swilliams;
```

The user swilliams can query jward.emp by referencing the table by name or using the synonym jward.employee:

```
SELECT * FROM jward.emp;
SELECT * FROM jward.employee;
```

If you grant object privileges on a table, view, sequence, procedure, function, or package to a **synonym** for the object, the effect is the same as if no synonym were used. For example, if jward wanted to grant the SELECT privilege for the emp table to swilliams, jward could issue either of the following statements:

```
GRANT SELECT ON emp TO swilliams;
GRANT SELECT ON employee TO swilliams;
```

If a synonym is dropped, all grants for the underlying schema object remain in effect, even if the privileges were granted by specifying the dropped synonym.

Grant and Revoke Schema Object Privileges

Schema object privileges can be granted to and revoked from users and roles. If you grant object privileges to roles, you can make the privileges selectively available. Object privileges for users and roles can be granted or revoked using the following:

- The SQL statements GRANT and REVOKE, respectively
- The Add Privilege to Role/User dialog box and the Revoke Privilege from Role/User dialog box of Oracle Enterprise Manager.

Who Can Grant Schema Object Privileges?

A user automatically has all object privileges for schema objects contained in his or her schema. A user can grant any object privilege on any schema object he or she owns to any other user or role. A user with the GRANT ANY OBJECT PRIVILEGE can grant or revoke any specified object privilege to another user with or without the GRANT OPTION of the GRANT statement. Otherwise, the grantee can use the privilege, but cannot grant it to other users.

For example, assume user SCOTT has a table named t2:

```
SQL>GRANT grant any object privilege TO U1;
SQL> connect u1/u1
Connected.
SQL> GRANT select on scott.t2 \TO U2;
SQL> SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
 WHERE TABLE_NAME = 'employees';
```

```
GRANTEE                        OWNER
------------------------------ ------------------------------
GRANTOR                        PRIVILEGE                                GRA
------------------------------ ---------------------------------------- ---
U2                             SCOTT
SCOTT                          SELECT                                   NO
```

See Also:

Oracle9i SQL Reference

Table Security

Schema object privileges for tables allow table security at the level of DML and DDL operations.

Data Manipulation Language Operations

You can grant privileges to use the `DELETE`, `INSERT`, `SELECT`, and `UPDATE` DML operations on a table or view. Grant these privileges only to users and roles that need to query or manipulate a table's data.

You can restrict `INSERT` and `UPDATE` privileges for a table to specific columns of the table. With selective `INSERT`, a privileged user can insert a row with values for the selected columns. All other columns receive `NULL` or the column's default value. With selective `UPDATE`, a user can update only specific column values of a row. Selective `INSERT` and `UPDATE` privileges are used to restrict a user's access to sensitive data.

For example, if you do not want data entry users to alter the `salary` column of the `employees` table, selective `INSERT` or `UPDATE` privileges can be granted that exclude the `salary` column. Alternatively, a view that excludes the `salary` column could satisfy this need for additional security.

See Also:

Oracle9i SQL Reference for more information about these DML operations

Data Definition Language Operations

The `ALTER`, `INDEX`, and `REFERENCES` privileges allow DDL operations to be performed on a table. Because these privileges allow other users to alter or create dependencies on a table, you should grant privileges conservatively. A user attempting to perform a DDL operation on a table may need additional system or object privileges. For example, to create a trigger on a table, the user requires both the `ALTER TABLE` object privilege for the table and the `CREATE TRIGGER` system privilege.

As with the `INSERT` and `UPDATE` privileges, the `REFERENCES` privilege can be granted on specific columns of a table. The `REFERENCES` privilege enables the grantee to use the table on which the grant is made as a parent key to any foreign keys that the grantee wishes to create in his or her own tables. This action is controlled with a special privilege because the presence of foreign keys restricts the data manipulation and table alterations that can be done to the parent key. A column-specific `REFERENCES` privilege restricts the grantee to using the named columns (which, of course, must include at least one primary or unique key of the parent table).

See Also:

Chapter 21, "Data Integrity" for more information about primary keys, unique keys, and integrity constraints

View Security

Schema object privileges for views allow various DML operations, which actually affect the base tables from which the view is derived. DML object privileges for tables can be applied similarly to views.

Privileges Required to Create Views

To create a view, you must meet the following requirements:

- You must have been granted one of the following system privileges, either explicitly or through a role:
  - The `CREATE VIEW` system privilege (to create a view in your schema)
  - The `CREATE ANY VIEW` system privilege (to create a view in another user's schema)
- You must have been explicitly granted one of the following privileges:
  - The `SELECT`, `INSERT`, `UPDATE`, or `DELETE` object privileges on all base objects underlying the view
  - The `SELECT ANY TABLE`, `INSERT ANY TABLE`, `UPDATE ANY TABLE`, or `DELETE ANY TABLE` system privileges
- Additionally, in order to grant other users access to your view, you must have received object privileges to the base objects with the `GRANT OPTION` clause or appropriate system privileges with the `ADMIN OPTION` clause. If you have not, grantees cannot access your view.

See Also:

Oracle9i SQL Reference

Increase Table Security with Views

To use a view, you require appropriate privileges only for the view itself. You do not require privileges on base objects underlying the view.

Views add two more levels of security for tables, column-level security and value-based security:

- A view can provide access to selected columns of base tables. For example, you can define a view on the `employees` table to show only the `employee_id`, `last_name`, and `manager_id` columns:

```
CREATE VIEW employees_manager AS
    SELECT last_name, employee_id, manager_id FROM employees;
```

- A view can provide value-based security for the information in a table. A `WHERE` clause in the definition of a view displays only selected rows of base tables. Consider the following two examples:

```
CREATE VIEW lowsal AS
    SELECT * FROM employees
    WHERE salary < 10000;
```

The `LOWSAL` view allows access to all rows of the `employees` table that have a salary value less than 10000. Notice that all columns of the `employees` table are accessible in the `LOWSAL` view.

```
CREATE VIEW own_salary AS
    SELECT last_name, salary
    FROM employees
    WHERE last_name = USER;
```

In the `own_salary` view, only the rows with an `last_name` that matches the current user of the view are accessible. The `own_salary` view uses the `user` pseudocolumn, whose values always refer to the current user. This view combines both column-level security and value-based security.

Procedure Security

The only **schema object privilege** for procedures, including standalone procedures and functions as well as packages, is EXECUTE. Grant this privilege only to users who need to execute a procedure or compile another procedure that calls it.

Procedure Execution and Security Domains

A user with the EXECUTE object privilege for a specific procedure can execute the procedure or compile a program unit that references the procedure. No runtime privilege check is made when the procedure is called. A user with the EXECUTE ANY PROCEDURE system privilege can execute any procedure in the database.

A user can be granted privileges through roles to execute procedures.

Additional privileges on referenced objects are required for invoker-rights procedures, but not for definer-rights procedures.

See Also:

"PL/SQL Blocks and Roles"

Definer Rights

A user of a definer-rights procedure requires only the privilege to execute the procedure and no privileges on the underlying objects that the procedure accesses, because a definer-rights procedure operates under the security domain of the user who owns the procedure, regardless of who is executing it. The procedure's owner must have all the necessary object privileges for referenced objects. Fewer privileges have to be granted to users of a definer-rights procedure, resulting in tighter control of database access.

You can use definer-rights procedures to control access to private database objects and add a level of database security. By writing a definer-rights procedure and granting only EXECUTE privilege to a user, the user can be forced to access the referenced objects only through the procedure.

At runtime, the privileges of the owner of a definer-rights stored procedure are always checked before the procedure is executed. If a necessary privilege on a referenced object has been revoked from the owner of a definer-rights procedure, then the procedure cannot be executed by the owner or any other user.

Note:

Trigger execution follows the same patterns as definer-rights procedures. The user executes a SQL statement, which that user is privileged to execute. As a result of the SQL statement, a trigger is fired. The statements within the triggered action temporarily execute under the security domain of the user that owns the trigger.

See Also:

Chapter 17, "Triggers"

Invoker Rights

An invoker-rights procedure executes with all of the invoker's privileges. Roles are enabled unless the invoker-rights procedure was called directly or indirectly by a definer-rights procedure. A user of an invoker-rights

procedure needs privileges (either directly or through a role) on objects that the procedure accesses through external references that are resolved in the invoker's schema.

The invoker needs privileges at runtime to access program references embedded in DML statements or dynamic SQL statements, because they are effectively recompiled at runtime.

For all other external references, such as direct PL/SQL function calls, the owner's privileges are checked at compile time, and no runtime check is made. Therefore, the user of an invoker-rights procedure needs no privileges on external references outside DML or dynamic SQL statements. Alternatively, the developer of an invoker-rights procedure only needs to grant privileges on the procedure itself, not on all objects directly referenced by the invoker-rights procedure.

Many packages provided by Oracle, such as most of the `DBMS_*` packages, run with invoker rights--they do not run as the owner (`SYS`) but rather as the current user. However, some exceptions exist such as the `DBMS_RLS` package.

You can create a software bundle that consists of multiple program units, some with definer rights and others with invoker rights, and restrict the program entry points *(***controlled step-in***)*. A user who has the privilege to execute an entry-point procedure can also execute internal program units indirectly, but cannot directly call the internal programs.

See Also:

- "Fine-Grained Access Control"
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for detailed documentation of the Oracle supplied packages

System Privileges Needed to Create or Alter a Procedure

To create a procedure, a user must have the `CREATE PROCEDURE` or `CREATE ANY PROCEDURE` **system privilege**. To alter a procedure, that is, to manually recompile a procedure, a user must own the procedure or have the `ALTER ANY PROCEDURE` system privilege.

The user who owns the procedure also must have privileges for schema objects referenced in the procedure body. To create a procedure, you must have been explicitly granted the necessary privileges (system or object) on all objects referenced by the procedure. You cannot have obtained the required privileges through roles. This includes the `EXECUTE` privilege for any procedures that are called inside the procedure being created.

Triggers also require that privileges to referenced objects be granted explicitly to the trigger owner. Anonymous PL/SQL blocks can use any privilege, whether the privilege is granted explicitly or through a role.

Packages and Package Objects

A user with the `EXECUTE` object privilege for a package can execute any public procedure or function in the package and access or modify the value of any public package variable. Specific `EXECUTE` privileges cannot be granted for a package's constructs. Therefore, you may find it useful to consider two alternatives for establishing security when developing procedures, functions, and packages for a database application. These alternatives are described in the following examples.

Packages and Package Objects Example 1

This example shows four procedures created in the bodies of two packages.

```
CREATE PACKAGE BODY hire_fire AS
  PROCEDURE hire(...) IS
    BEGIN
      INSERT INTO employees . . .
    END hire;
  PROCEDURE fire(...) IS
    BEGIN
      DELETE FROM employees . . .
    END fire;
END hire_fire;

CREATE PACKAGE BODY raise_bonus AS
  PROCEDURE give_raise(...) IS
    BEGIN
      UPDATE employees SET salary = . . .
    END give_raise;
  PROCEDURE give_bonus(...) IS
    BEGIN
      UPDATE employees SET bonus = . . .
    END give_bonus;
END raise_bonus;
```

Access to execute the procedures is given by granting the `EXECUTE` privilege for the package, using the following statements:

```
GRANT EXECUTE ON hire_fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

Granting `EXECUTE` privilege granted for a package provides uniform access to all package objects.

Packages and Package Objects Example 2

This example shows four procedure definitions within the body of a single package. Two additional standalone procedures and a package are created specifically to provide access to the procedures defined in the main package.

```
CREATE PACKAGE BODY employee_changes AS
  PROCEDURE change_salary(...) IS BEGIN ... END;
  PROCEDURE change_bonus(...) IS BEGIN ... END;
  PROCEDURE insert_employee(...) IS BEGIN ... END;
  PROCEDURE delete_employee(...) IS BEGIN ... END;
END employee_changes;

CREATE PROCEDURE hire
  BEGIN
    employee_changes.insert_employee(...)
  END hire;

CREATE PROCEDURE fire
  BEGIN
    employee_changes.delete_employee(...)
  END fire;

PACKAGE raise_bonus IS
  PROCEDURE give_raise(...) AS
    BEGIN
      employee_changes.change_salary(...)
    END give_raise;

  PROCEDURE give_bonus(...)
```

```
BEGIN
  employee_changes.change_bonus(...)
END give_bonus;
```

Using this method, the procedures that actually do the work (the procedures in the `employee_changes` package) are defined in a single package and can share declared global variables, cursors, on so on. By declaring top-level procedures `hire` and `fire`, and an additional package `raise_bonus`, you can grant selective `EXECUTE` privileges on procedures in the main package:

```
GRANT EXECUTE ON hire, fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

Type Security

This section describes privileges for types, methods, and objects.

System Privileges for Named Types

Oracle defines system privileges shown in Table 23-1 for named types (object types, `VARRAY`s, and nested tables):

Table 23-1 System Privileges for Named Types

Privilege	Allows you to...
`CREATE TYPE`	Create named types in your own schemas.
`CREATE ANY TYPE`	Create a named type in any schema.
`ALTER ANY TYPE`	Alter a named type in any schema.
`DROP ANY TYPE`	Drop a named type in any schema.
`EXECUTE ANY TYPE`	Use and reference a named type in any schema.

The `CONNECT` and `RESOURCE` roles include the `CREATE TYPE` system privilege. The DBA role includes all of these privileges.

Object Privileges

The only object privilege that applies to named types is `EXECUTE`. If the `EXECUTE` privilege exists on a named type, a user can use the named type to:

- Define a table
- Define a column in a relational table
- Declare a variable or parameter of the named type

The `EXECUTE` privilege permits a user to invoke the type's methods, including the type constructor. This is similar to `EXECUTE` privilege on a stored PL/SQL procedure.

Method Execution Model

Method execution is the same as any other stored PL/SQL procedure.

See Also:

"Procedure Security"

Privileges Required to Create Types and Tables Using Types

To create a type, you must meet the following requirements:

- You must have the CREATE TYPE system privilege to create a type in your schema or the CREATE ANY TYPE system privilege to create a type in another user's schema. These privileges can be acquired explicitly or through a role.
- The owner of the type must have been explicitly granted the EXECUTE object privileges to access all other types referenced within the definition of the type, or have been granted the EXECUTE ANY TYPE system privilege. The owner cannot have obtained the required privileges through roles.
- If the type owner intends to grant access to the type to other users, the owner must have received the EXECUTE privileges to the referenced types with the GRANT OPTION or the EXECUTE ANY TYPE system privilege with the ADMIN OPTION. If not, the type owner has insufficient privileges to grant access on the type to other users.

To create a table using types, you must meet the requirements for creating a table and these additional requirements:

- The owner of the table must have been explicitly granted the EXECUTE object privileges to access all types referenced by the table, or have been granted the EXECUTE ANY TYPE system privilege. The owner cannot have obtained the required privileges through roles.
- If the table owner intends to grant access to the table to other users, the owner must have received the EXECUTE privileges to the referenced types with the GRANT OPTION or the EXECUTE ANY TYPE system privilege with the ADMIN OPTION. If not, the table owner has insufficient privileges to grant access on the type to other users.

See Also:

"Table Security" for the requirements for creating a table

Privileges Required to Create Types and Tables Using Types Example

Assume that three users exist with the CONNECT and RESOURCE roles:

- user1
- user2
- user3

User1 performs the following DDL in his schema:

```
CREATE TYPE type1 AS OBJECT (
  attr1 NUMBER);

CREATE TYPE type2 AS OBJECT (
  attr2 NUMBER);

GRANT EXECUTE ON type1 TO user2;
GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION;
```

User2 performs the following DDL in his schema:

```
CREATE TABLE tab1 OF user1.type1;
CREATE TYPE type3 AS OBJECT (
  attr3 user1.type2);
CREATE TABLE tab2 (
  col1 user1.type2);
```

The following statements succeed because `user2` has `EXECUTE` privilege on `user1`'s `TYPE2` with the `GRANT OPTION`:

```
GRANT EXECUTE ON type3 TO user3;
GRANT SELECT on tab2 TO user3;
```

However, the following grant fails because `user2` does not have `EXECUTE` privilege on `user1`'s `TYPE1` with the `GRANT OPTION`:

```
GRANT SELECT ON tab1 TO user3;
```

`User3` can successfully perform the following statements:

```
CREATE TYPE type4 AS OBJECT (
  attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

Privileges on Type Access and Object Access

Existing column-level and table-level privileges for DML statements apply to both column objects and row objects. Oracle defines the privileges shown in Table 23-2 for object tables:

Table 23-2 Privileges for Object Tables

Privilege	Allows you to...
`SELECT`	Access an object and its attributes from the table
`UPDATE`	Modify the attributes of the objects that make up the table's rows
`INSERT`	Create new objects in the table
`DELETE`	Delete rows

Similar table privileges and column privileges apply to column objects. Retrieving instances does not in itself reveal type information. However, clients must access named type information in order to interpret the type instance images. When a client requests such type information, Oracle checks for `EXECUTE` privilege on the type.

Consider the following schema:

```
CREATE TYPE emp_type (
    eno NUMBER, ename CHAR(31), eaddr addr_t);
CREATE TABLE emp OF emp_t;
```

and the following two queries:

```
SELECT VALUE(emp) FROM emp;
SELECT eno, ename FROM emp;
```

For either query, Oracle checks the user's SELECT privilege for the emp table. For the first query, the user needs to obtain the emp_type type information to interpret the data. When the query accesses the emp_type type, Oracle checks the user's EXECUTE privilege.

Execution of the second query, however, does not involve named types, so Oracle does not check type privileges.

Additionally, using the schema from the previous section, user3 can perform the following queries:

```
SELECT tab1.col1.attr2 FROM user2.tab1 tab1;
SELECT attr4.attr3.attr2 FROM tab3;
```

Note that in both SELECT statements, user3 does not have explicit privileges on the underlying types, but the statement succeeds because the type and table owners have the necessary privileges with the GRANT OPTION.

Oracle checks privileges on the following events, and returns an error if the client does not have the privilege for the action:

- Pinning an object in the object cache using its REF value causes Oracle to check SELECT privilege on the containing object table.
- Modifying an existing object or flushing an object from the object cache causes Oracle to check UPDATE privilege on the destination object table.
- Flushing a new object causes Oracle to check INSERT privilege on the destination object table.
- Deleting an object causes Oracle to check DELETE privilege on the destination table.
- Pinning an object of named type causes Oracle to check EXECUTE privilege on the object.

Modifying an object's attributes in a client 3GL application causes Oracle to update the entire object. Hence, the user needs UPDATE privilege on the object table. UPDATE privilege on only certain columns of the object table is not sufficient, even if the application only modifies attributes corresponding to those columns. Therefore, Oracle does not support column level privileges for object tables.

Type Dependencies

As with stored objects such as procedures and tables, types being referenced by other objects are called dependencies. There are some special issues for types depended upon by tables. Because a table contains data that relies on the type definition for access, any change to the type causes all stored data to become inaccessible. Changes that can cause this effect are when necessary privileges required by the type are revoked or the type or dependent types are dropped. If either of these actions occur, then the table becomes invalid and cannot be accessed.

A table that is invalid because of missing privileges can automatically become valid and accessible if the required privileges are granted again. A table that is invalid because a dependent type has been dropped can never be accessed again, and the only permissible action is to drop the table.

Because of the severe effects which revoking a privilege on a type or dropping a type can cause, the SQL statements REVOKE and DROP TYPE by default implement a restrict semantics. This means that if the named type in either statement has table or type dependents, then an error is received and the statement aborts. However, if the FORCE clause for either statement is used, the statement always succeeds, and if there are depended-upon tables, they are invalidated.

See Also:

Oracle9i Database Reference for details about using the REVOKE, DROP TYPE, and FORCE clauses

Introduction to Roles

Oracle provides for easy and controlled privilege management through roles. Roles are named groups of related privileges that you grant to users or other roles. Roles are designed to ease the administration of end-user system and schema object privileges. However, roles are not meant to be used for application developers, because the privileges to access schema objects within stored programmatic constructs need to be granted directly.

These following properties of roles enable easier privilege management within a database:

Property	Description
Reduced privilege administration	Rather than granting the same set of privileges explicitly to several users, you can grant the privileges for a group of related users to a role, and then only the role needs to be granted to each member of the group.
Dynamic privilege management	If the privileges of a group must change, only the privileges of the role need to be modified. The security domains of all users granted the group's role automatically reflect the changes made to the role.
Selective availability of privileges	You can selectively enable or disable the roles granted to a user. This allows specific control of a user's privileges in any given situation.
Application awareness	The data dictionary records which roles exist, so you can design applications to query the dictionary and automatically enable (or disable) selective roles when a user attempts to execute the application by way of a given username.
Application-specific security	You can protect role use with a password. Applications can be created specifically to enable a role when supplied the correct password. Users cannot enable the role if they do not know the password.

Database administrators often create roles for a database application. The DBA grants a secure application role all privileges necessary to run the application. The DBA then grants the secure application role to other roles or users. An application can have several different roles, each granted a different set of privileges that allow for more or less data access while using the application.

The DBA can create a role with a password to prevent unauthorized use of the privileges granted to the role. Typically, an application is designed so that when it starts, it enables the proper role. As a result, an application user does not need to know the password for an application's role.

See Also:

- "Data Definition Language Statements and Roles" for information about restrictions for procedures
- *Oracle9i Application Developer's Guide - Fundamentals* for instructions for enabling roles from an application

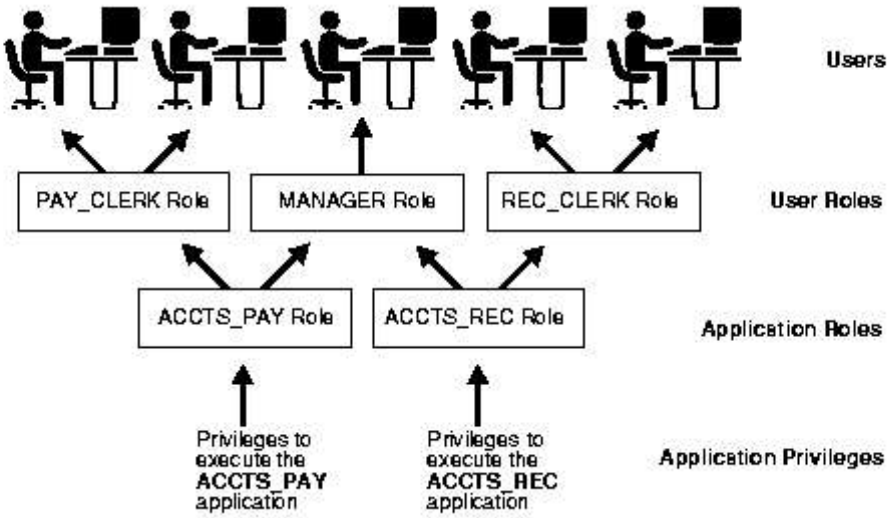
Common Uses for Roles

In general, you create a role to serve one of two purposes:

- To manage the privileges for a database application
- To manage the privileges for a user group

Figure 23-1 and the sections that follow describe the two uses of roles.

Figure 23-1 Common Uses for Roles



Text description of the illustration cncpt082.gif

Application Roles

You grant an application role all privileges necessary to run a given database application. Then, you grant the secure application role to other roles or to specific users. An application can have several different roles, with each role assigned a different set of privileges that allow for more or less data access while using the application.

User Roles

You create a user role for a group of database users with common privilege requirements. You manage user privileges by granting secure application roles and privileges to the user role and then granting the user role to appropriate users.

The Mechanisms of Roles

Database roles have the following functionality:

- A role can be granted system or schema object privileges.
- A role can be granted to other roles. However, a role cannot be granted to itself and cannot be granted circularly. For example, role `A` cannot be granted to role `B` if role `B` has previously been granted to role `A`.
- Any role can be granted to any database user.
- Each role granted to a user is, at a given time, either enabled or disabled. A user's security domain includes the privileges of all roles currently enabled for the user and excludes the privileges of any roles currently disabled for the user. Oracle allows database applications and users to enable and disable roles to provide selective availability of privileges.
- An indirectly granted role is a role granted to a role. It can be explicitly enabled or disabled for a user. However, by enabling a role that contains other roles, you implicitly enable all indirectly granted roles of the directly granted role.

Grant and Revoke Roles

You grant or revoke roles from users or other roles using the following options:

- The Grant System Privileges/Roles dialog box and Revoke System Privileges/Roles dialog box of Oracle Enterprise Manager
- The SQL statements `GRANT` and `REVOKE`

Privileges are granted to and revoked from roles using the same options. Roles can also be granted to and revoked from users using the operating system that executes Oracle, or through network services.

See Also:

Oracle9i Database Administrator's Guide for detailed instructions about role management

Who Can Grant or Revoke Roles?

Any user with the `GRANT ANY ROLE` system privilege can grant or revoke any role except a global role to or from other users or roles of the database. You should grant this system privilege conservatively because it is very powerful.

Any user granted a role with the `ADMIN OPTION` can grant or revoke that role to or from other users or roles of the database. This option allows administrative powers for roles on a selective basis.

See Also:

Oracle9i Database Administrator's Guide for information about global roles

Role Names

Within a database, each role name must be unique, and no username and role name can be the same. Unlike schema objects, roles are not contained in any schema. Therefore, a user who creates a role can be dropped with no effect on the role.

Security Domains of Roles and Users

Each role and user has its own unique security domain. A role's security domain includes the privileges granted to the role plus those privileges granted to any roles that are granted to the role.

A user's security domain includes privileges on all schema objects in the corresponding schema, the privileges granted to the user, and the privileges of roles granted to the user that are **currently enabled**. (A role can be simultaneously enabled for one user and disabled for another.) A user's security domain also includes the privileges and roles granted to the user group `PUBLIC`.

PL/SQL Blocks and Roles

The use of roles in a PL/SQL block depends on whether it is an anonymous block or a named block (stored procedure, function, or trigger), and whether it executes with definer rights or invoker rights.

Named Blocks with Definer Rights

All roles are disabled in any named PL/SQL block (stored procedure, function, or trigger) that executes with definer rights. Roles are not used for privilege checking and you cannot set roles within a definer-rights procedure.

The `SESSION_ROLES` view shows all roles that are currently enabled. If a named PL/SQL block that executes with definer rights queries `SESSION_ROLES`, the query does not return any rows.

See Also:

Oracle9i Database Reference

Anonymous Blocks with Invoker Rights

Named PL/SQL blocks that execute with invoker rights and anonymous PL/SQL blocks are executed based on privileges granted through enabled roles. Current roles are used for privilege checking within an invoker-rights PL/SQL block, and you can use dynamic SQL to set a role in the session.

See Also:

- *PL/SQL User's Guide and Reference* for an explanation of invoker and definer rights
- "Dynamic SQL in PL/SQL"

Data Definition Language Statements and Roles

A user requires one or more privileges to successfully execute a data definition language (DDL) statement, depending on the statement. For example, to create a table, the user must have the `CREATE TABLE` or `CREATE ANY TABLE` system privilege. To create a view of another user's table, the creator requires the `CREATE VIEW` or `CREATE ANY VIEW` system privilege and either the `SELECT` *`object`* privilege for the table or the `SELECT ANY TABLE` system privilege.

Oracle avoids the dependencies on privileges received by way of roles by restricting the use of specific privileges in certain DDL statements. The following rules outline these privilege restrictions concerning DDL statements:

- All system privileges and schema object privileges that permit a user to perform a DDL operation are usable when received through a role. For example:
  - System Privileges: the `CREATE TABLE`, `CREATE VIEW` and `CREATE PROCEDURE` privileges.
  - Schema Object Privileges: the `ALTER` and `INDEX` privileges for a table.

  Exception: The `REFERENCES` object privilege for a table cannot be used to define a table's foreign key if the privilege is received through a role.

- All system privileges and object privileges that allow a user to perform a DML operation that is required to issue a DDL statement are *not* usable when received through a role. For example:
  - A user who receives the `SELECT ANY TABLE` system privilege or the `SELECT` *`object`* privilege for a table through a role can use neither privilege to create a view on another user's table.

The following example further clarifies the permitted and restricted uses of privileges received through roles:

Assume that a user is:

- Granted a role that has the `CREATE VIEW` system privilege
- Granted a role that has the `SELECT` *`object`* privilege for the `employees` table, but the user is indirectly granted the `SELECT` *`object`* privilege for the `employees` table
- Directly granted the `SELECT` *`object`* privilege for the `departments` table

Given these directly and indirectly granted privileges:

- The user can issue SELECT statements on both the employees and departments tables.
- Although the user has both the CREATE VIEW and SELECT privilege for the employees table through a role, the user cannot create a usable view on the employees table, because the SELECT *object* privilege for the employees table was granted through a role. Any views created will produce errors when accessed.
- The user can create a view on the departments table, because the user has the CREATE VIEW privilege through a role and the SELECT privilege for the departments table directly.

Predefined Roles

The following roles are defined automatically for Oracle databases:

- CONNECT
- RESOURCE
- DBA
- EXP_FULL_DATABASE
- IMP_FULL_DATABASE

These roles are provided for backward compatibility to earlier versions of Oracle and can be modified in the same manner as any other role in an Oracle database.

The Operating System and Roles

In some environments, you can administer database security using the operating system. The operating system can be used to manage the granting (and revoking) of database roles and to manage their password authentication. This capability is not available on all operating systems.

See Also:

Your operating system specific Oracle documentation for details on managing roles through the operating system

Roles in a Distributed Environment

When you use roles in a distributed database environment, you must ensure that all needed roles are set as the default roles for a distributed (remote) session. You cannot enable roles when connecting to a remote database from within a local database session. For example, you cannot execute a remote procedure that attempts to enable a role at the remote site.

See Also:

Oracle9i Heterogeneous Connectivity Administrator's Guide

Fine-Grained Access Control

Fine-grained access control lets you implement security policies with functions and then associate those security policies with tables, views, or synonyms. The database server automatically enforces those security policies, no matter how the data is accessed (for example, by ad hoc queries).

You can:

- Use different policies for SELECT, INSERT, UPDATE, and DELETE.

- Use security policies only where you need them (for example, on salary information).
- Use more than one policy for each table, including building on top of base policies in packaged applications.
- Distinguish policies between different applications, by using *policy groups*. Each policy group indicates a set of policies that belong to an application.

The database administrator designates an application context, called a *driving context*, to indicate the policy group in effect. When tables, views, or synonyms are accessed, the fine-grained access control engine looks up the driving context to determine the policy group in effect and enforces all the associated policies that belong to that policy group.

The PL/SQL package `DBMS_RLS` let you administer your security policies. Using this package, you can add, drop, enable, disable, and refresh the policies you create.

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for information about package implementation
- *Oracle9i Application Developer's Guide - Fundamentals* for information and examples on establishing security policies

Dynamic Predicates

The function or package that implements the security policy you create returns a predicate (a `WHERE` condition). This predicate controls access as set out by the policy. Rewritten queries are fully optimized and shareable.

Application Context

Application context facilitates the implementation of fine-grained access control. It lets you implement security policies with functions and then associate those security policies with applications. Each application can have its own application-specific context. Users are not allowed to arbitrarily change their context (for example, through SQL*Plus).

Application contexts permit flexible, parameter-based access control, based on attributes of interest to an application. For example, context attributes for a human resources application could include "position," "organizational unit," and "country," whereas attributes for an order-entry control might be "customer number" and "sales region".

You can:

- Base predicates on context values
- Use context values within predicates, as bind variables
- Set user attributes
- Access user attributes

To define an application context:

1. Create a PL/SQL package with functions that validate and set the context for your application. You may want to use an event trigger on login to set the initial context for logged-in users.

2. Use `CREATE CONTEXT` to specify a unique context name and associate it with the PL/SQL package you created.
3. Do one of the following:
   - Reference the application context in a policy function implementing fine-grained access control.
   - Create an event trigger on login to set the initial context for a user. For example, you could query a user's employee number and set this as an "employee number" context value.
4. Reference the application context.

See Also:

- *PL/SQL User's Guide and Reference*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*
- *Oracle9i Application Developer's Guide - Fundamentals*

Secure Application Roles

Oracle provides secure application roles, which are roles that can be enabled only by authorized PL/SQL packages. This mechanism restricts the enabling of roles to the invoking application.

In previous releases, passwords were either embedded in the source code or stored in a table. Application developers no longer need to secure a role by embedding passwords inside applications. Instead, they create a secure application role and specify which PL/SQL package is authorized to enable the role. Package identity is used to determine whether there are sufficient privileges to enable the roles. The application performs authentication before enabling the role.

The application can perform customized authorization, such as checking whether the user has connected through a proxy, before enabling the role.

Note:

Because of the restriction that users cannot change security domain inside definer's right procedures, secure application roles can only be enabled inside invoker's right procedures.

Creation of Secure Application Roles

Secure application roles are created by using the `CREATE ROLE ... IDENTIFIED USING` statement. Here is an example:

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

This indicates the following:

- The role `admin_role` to be created is a secure application role.
- The role can only be enabled by any module defined inside the PL/SQL package `hr.admin`.

You must have the system privilege `CREATE ROLE` to execute this statement.

Roles that are enabled inside an Invoker's Right procedure remain in effect even after the procedure exits. Therefore, you can have a dedicated procedure that deals with enabling the role for the rest of the session to use.

See Also:

- *Oracle9i SQL Reference*
- *PL/SQL User's Guide and Reference*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*
- *Oracle9i Application Developer's Guide - Fundamentals*

Previous Next

Home Book List Contents Index Master Index Feedback