4

Configuring Privilege and Role Authorization

Privilege and role authorization controls the permissions that users have to perform day-to-day tasks.

- About Privileges and Roles
  Authorization permits users to access, process, or alter data; it also creates limitations on user access or actions.
- Privilege and Role Grants in a CDB
  The scope of a privilege and role grant in a CDB depends on where the role is being used.
- Who Should Be Granted Privileges?
  You grant privileges to users so they can accomplish tasks required for their jobs.
- How the Oracle Multitenant Option Affects Privileges
  All users, including common users, can exercise their privileges only within the current container.
- Managing Administrative Privileges
  Administrative privileges can be used for both general and specific database operations.
- Managing System Privileges
  To perform actions on schema objects, you must be granted the appropriate system privileges.
- Managing Schema Privileges
  Schema privileges enable certain system privileges to be granted on a schema.
- Administering Schema Security Policies
  To manage schema security policies for row level security, fine-grained auditing, and Oracle Data Redaction, users must be granted the appropriate system privilege.
- Managing Privileges to Enable Diagnostics
  Only users who have the `SYSDBA` administrative privilege or the `ENABLE_DIAGNOSTICS` system privilege can enable diagnostics.
- Managing Commonly and Locally Granted Privileges
  Privileges can be granted commonly for an entire CDB or application container, or granted locally to a specific PDB.
- Managing User Roles
  A user role is a named collection of privileges that you can create and assign to other users.
- Managing Common Roles and Local Roles
  A common role is a role that is created in the root; a local role is created in a PDB.
- Restricting Operations on PDBs Using PDB Lockdown Profiles
  You can use PDB lockdown profiles to restrict sets of user operations in pluggable databases (PDBs).
- Managing Object Privileges
  Object privileges enable you to perform actions on schema objects, such as tables or indexes.

- Managing Dictionary Protection for Oracle-Maintained Schemas
  Oracle-maintained schemas such as `AUDSYS` have dictionary protection to prevent users from using system privileges on these schemas.
- Table Privileges
  Object privileges for tables enable table security at the DML or DDL level of operation.
- View Privileges
  You can apply DML object privileges to views, similar to tables.
- Procedure Privileges
  The `EXECUTE` privilege enables users to run procedures and functions, either standalone or in packages.
- Type Privileges
  You can control system and object privileges for types, methods, and objects.
- Grants of User Privileges and Roles
  The `GRANT` statement provides privileges for a user to perform specific actions, such as executing a procedure.
- Revokes of Privileges and Roles from a User
  When you revoke system or object privileges, be aware of the cascading effects of revoking a privilege.
- Grants and Revokes of Privileges to and from the PUBLIC Role
  You can grant and revoke privileges and roles from the role `PUBLIC`.
- Grants of Roles Using the Operating System or Network
  Using the operating system or network to manage roles can help centralize the role management in a large enterprise.
- How Grants and Revokes Work with SET ROLE and Default Role Settings
  Privilege grants and the `SET ROLE` statement affect when and how grants and revokes take place.
- Configuring Read-Only Users
  You can override the privileges and roles that have been granted to a user by making the user a read-only user.
- User Privilege and Role Data Dictionary Views
  You can use special queries to find information about various types of privilege and role grants.

4.1 About Privileges and Roles

Authorization permits users to access, process, or alter data; it also creates limitations on user access or actions.

The limitations placed on (or removed from) users can apply to objects such as schemas, entire tables, or table rows.

A user **privilege** is the right to run a particular type of SQL statement, or the right to access an object that belongs to another user, run a PL/SQL package, and so on. The types of privileges are defined by Oracle Database.

Roles are created by users (usually administrators) to group together privileges or other roles. They are a way to facilitate the granting of multiple privileges or roles to users. In addition to granting roles to users and other roles, you can assign roles to programs by using code based access control (CBAC).

Privileges can fall into the following general categories:

- **Administrative privileges.** Administrative privileges are designed for commonly performed administrative tasks, such as performing backup and recovery operations. Oracle Database provides administrative privileges tailored to specific administrative tasks, such as the `SYSKM` administrative privilege for performing Transparent Data Encryption tasks.
- **System privileges.** System privileges enable users to perform actions on schema objects. Examples of a system privilege are the ability to create and update tables or tablespaces.
- **Roles.** A **role** groups several privileges and roles, so that they can be granted to and revoked from users simultaneously. You must enable the role for a user before the user can use it. You can embed roles by using the `SET ROLE` PL/SQL statement. See *Oracle Database SQL Language Reference*.
- **Object privileges.** Each type of object has privileges associated with it. Objects are schema objects, such as tables or indexes. Categories of object privileges are as follows:
  - **Table privileges.** These privileges enable security at the DML (data manipulation language) or DDL (data definition language) level. DML operations are `DELETE`, `INSERT`, `SELECT`, and `UPDATE` operations on tables. DDL operations are `ALTER`, `INDEX`, and `REFERENCES` operations on tables and views.
  - **View privileges.** You can apply DML object privileges to views, similar to tables.
  - **Procedure privileges.** Procedures, including standalone procedures and functions, can be granted the `EXECUTE` privilege.
  - **Type privileges.** You can grant system privileges to named types (object types, `VARRAY`s, and nested tables).
- **Read-only user and session privileges.** You can configure whether a user or session is enabled for read-write or read-only operations.

Related Topics

- Managing Administrative Privileges
  Administrative privileges can be used for both general and specific database operations.
- Managing System Privileges
  To perform actions on schema objects, you must be granted the appropriate system privileges.
- Managing Commonly and Locally Granted Privileges
  Privileges can be granted commonly for an entire CDB or application container, or granted locally to a specific PDB.
- Configuring Read-Only Users
  You can override the privileges and roles that have been granted to a user by making the user a read-only user.
- Using Code Based Access Control for Definer's Rights and Invoker's Rights
  Code based access control, used to attach database roles to PL/SQL functions, procedures, or packages, works well with invoker's rights and definer's procedures.

4.2 Privilege and Role Grants in a CDB

The scope of a privilege and role grant in a CDB depends on where the role is being used.

- About Privilege and Role Grants in a CDB
  User accounts in a CDB can grant and be granted roles and privileges. Roles and privileges in a CDB, however, are either locally or commonly granted.

- Principles of Privilege and Role Grants in a CDB
  In a CDB, every act of granting, whether local or common, occurs within a container. The container may be the CDB root, an application root, or a PDB.
- Privileges and Roles Granted Locally in a CDB
  Roles and privileges may be granted locally to users and roles *regardless* of whether the grantees, grantors, or roles being granted are local or common.
- What Makes a Privilege or Role Grant Local
  To grant a role or privilege locally, use the `GRANT` statement with the `CONTAINER=CURRENT` clause, which is the default.
- Roles and Privileges Granted Locally
  A user or role may be locally granted a privilege (`CONTAINER=CURRENT`).
- Roles and Privileges Granted Commonly in a CDB
  Privileges and common roles may be granted commonly.
- What Makes a Grant Common
  The `CONTAINER=ALL` clause specifies that the privilege or role is being granted commonly.
- Roles and Privileges Granted Commonly
  A common user account or role may be granted a privilege commonly (`CONTAINER=ALL`).
- Grants to PUBLIC in a CDB
  In a CDB, `PUBLIC` is a common role. In a PDB, privileges granted locally to `PUBLIC` enable all local and common user account to exercise these privileges in this PDB only.
- Grants of Privileges and Roles: Scenario
  In this scenario, `SYSTEM` creates common user `c##dba` and tries to give this user privileges to query a table in the `hr` schema in `hrpdb`.

4.2.1 About Privilege and Role Grants in a CDB

User accounts in a CDB can grant and be granted roles and privileges. Roles and privileges in a CDB, however, are either locally or commonly granted.

A privilege or role granted locally is exercisable only in the PDB in which it was granted. A privilege or role granted commonly is exercisable in every existing and future PDB in the container—either the CDB or an application container—in which it was granted.

Users and roles may be common or local. However, a privilege is *in itself* neither common nor local. If a user grants a privilege locally using the `CONTAINER=CURRENT` clause, then the grantee has a privilege exercisable only in the current container. If a user connects to either the CDB root or an application root, and if this user grants a privilege commonly using the `CONTAINER=ALL` clause, then the grantee has this privilege in any existing or future PDB within the current container.

4.2.2 Principles of Privilege and Role Grants in a CDB

In a CDB, every act of granting, whether local or common, occurs within a container. The container may be the CDB root, an application root, or a PDB.

If the current container is the CDB root, then granting commonly means granting to all containers in the CDB. If the current container is an application root, however, then granting commonly means granting to all PDBs in the current application container.

The basic principles of granting are as follows:

- Both common and local phenomena may grant and be granted locally.

- Only common phenomena may grant or be granted commonly.

Local users, roles, and privileges are restricted to a particular PDB. Thus, local users may not grant roles and privileges commonly, and local roles and privileges may not be granted commonly.

The following sections describe the implications of the preceding principles.

4.2.3 Privileges and Roles Granted Locally in a CDB

Roles and privileges may be granted locally to users and roles *regardless* of whether the grantees, grantors, or roles being granted are local or common.

The following table explains the valid possibilities for locally granted roles and privileges.

Table 4-1 Local Grants

Phenomenon	May Grant Locally	May Be Granted Locally	May Receive a Role or Privilege Granted Locally
Common User	Yes	N/A	Yes
Local User	Yes	N/A	Yes
Common Role	N/A	Yes (but privileges in this role are available to the grantee only in the container in which the role was granted, regardless of whether the privileges were granted to the role locally or commonly)	Yes
Local Role	N/A	Yes (but privileges in this role are available to the grantee only in the container in which the role was granted and created)	Yes
Privilege	N/A	Yes	N/A

4.2.4 What Makes a Privilege or Role Grant Local

To grant a role or privilege locally, use the `GRANT` statement with the `CONTAINER=CURRENT` clause, which is the default.

Specifically, a role or privilege is granted locally only when the following criteria are met:

- The grantor has the necessary privileges to grant the specified role or privileges.

  For system privileges and roles, the grantor must have the `ADMIN OPTION` for the role or privilege being granted. For object privileges, the grantor must have the `GRANT OPTION` for the privilege being granted.
- The grant applies to only one container.

  By default, the `GRANT` statement includes the `CONTAINER=CURRENT` clause, which indicates that the privilege or role is granted locally.

Example 4-1 Granting a Privilege Locally

In this example, both `SYSTEM` and `c##hr_admin` are common users. The example connects to `hrpdb` as `SYSTEM` (which has administrator privileges), and then locally grants read privileges on the `employees` table to `c##hr_admin`. This grant applies *only* to `c##hr_admin` within `hrpdb`, not within any other PDBs.

```
CONNECT SYSTEM@hrpdb
Enter password: password
Connected.

GRANT READ ON employees TO c##hr_admin CONTAINER=CURRENT;
```

4.2.5 Roles and Privileges Granted Locally

A user or role may be locally granted a privilege (`CONTAINER=CURRENT`).

For example, a `READ ANY TABLE` privilege granted locally to a local or common user in `hrpdb` applies only to this user in this PDB.

A user or role may be locally granted a role (`CONTAINER=CURRENT`). A common role may receive a privilege granted locally. For example, the common role `c##dba` may be granted the `READ ANY TABLE` privilege locally in `hrpdb`. If the `c##cdb` common role has local privileges, then these privileges apply *only* in the container in which the role is granted. In this example, a common user who has the `c##cdba` role does not, because of a privilege granted locally to this role in `hrpdb`, have the right to exercise this privilege in any PDB other than `hrpdb`.

4.2.6 Roles and Privileges Granted Commonly in a CDB

Privileges and common roles may be granted commonly.

User accounts or roles may be granted roles and privileges commonly only if the grantees and grantors are both *common*. If a role is being granted commonly, then the role itself must be common. The following table explains the possibilities for common grants.

Table 4-2 Common Grants

Phenomenon	May Grant Commonly	May Be Granted Commonly	May Receive Roles and Privileges Granted Commonly
Common User Account	Yes	N/A	Yes
Local User Account	No	N/A	No
Common Role	N/A	Yes[1]	Yes
Local Role	N/A	No	No
Privilege	N/A	Yes	N/A

1 Privileges that were granted commonly to a common role are available to the grantee across all containers. In addition, any privilege granted locally to a common role is available to the grantee only in the container in which that privilege was granted to the common role.

4.2.7 What Makes a Grant Common

The `CONTAINER=ALL` clause specifies that the privilege or role is being granted commonly.

A role or privilege is granted commonly when the following criteria are met:

- The grantor is a common user.

  The user that performs the grant is either common to the CDB itself, or common to a specific application container.

- The grantee is a common user or common role.

  The recipient of the grant is either common to the CDB itself, or common to a specific application container.

- The grantor has the necessary privileges to grant the specified role or privileges.

  For system privileges and roles, the grantor must have the `ADMIN OPTION` for the role or privilege being granted. For object privileges, the grantor must have the `GRANT OPTION` for the privilege being granted.

- The grant applies to all PDBs within the container (either CDB or application container) in which the grant occurred.

  The `GRANT` statement includes a `CONTAINER=ALL` clause specifying that the privilege or role is granted commonly.

- If a role is being granted, then it must be common, and if an object privilege is being granted, then the object on which the privilege is granted must be common.

Example 4-2 Granting a Privilege Commonly

In this example, both `SYSTEM` and `c##hr_admin` are common users. `SYSTEM` connects to the CDB root, and then grants the `CREATE ANY TABLE` privilege commonly to `c##hr_admin`. In this case, `c##hr_admin` can now create a table in any PDB in the CDB.

```
CONNECT SYSTEM@root
Enter password: password
Connected.

GRANT CREATE ANY TABLE TO c##hr_admin CONTAINER=ALL;
```

4.2.8 Roles and Privileges Granted Commonly

A common user account or role may be granted a privilege commonly (`CONTAINER=ALL`).

Within the context of either the CDB root or an application root, the privilege is granted to this common user account or role in all existing and future PDBs within the current container. For example, if `SYSTEM` connects to the CDB root and grants a `SELECT ANY TABLE` privilege commonly to CDB common user account `c##dba`, then the `c##dba` user has this privilege in all PDBs in the CDB. A role or privilege granted commonly cannot be revoked locally.

A user or role may receive a common role granted commonly. A common role may receive a privilege granted locally. Thus, a common user can be granted a common role, and this role may contain locally granted privileges.

For example, the common role `c##admin` may be granted the `SELECT ANY TABLE` privilege that is local to `hrpdb`. Locally granted privileges in a common role apply *only* in the container in

which the privilege was granted. Thus, the common user with the c##admin role does not have the right to exercise an hrpdb-contained privilege in salespdb or any PDB other than hrpdb.

4.2.9 Grants to PUBLIC in a CDB

In a CDB, PUBLIC is a common role. In a PDB, privileges granted locally to PUBLIC enable all local and common user account to exercise these privileges in this PDB only.

Every privilege and role granted to Oracle-supplied users and roles is granted commonly except for system privileges granted to PUBLIC, which are granted locally. This exception exists because you may want to revoke some grants included by default in Oracle Database, such as EXECUTE on the SYS.UTL_FILE package.

Assume that local user account hr exists in hrpdb. This user locally grants the SELECT privilege on hr.employees to PUBLIC. Common and local users in hrpdb may exercise the privilege granted to PUBLIC. User accounts in salespdb or any other PDB do not have the privilege to query hr.employees in hrpdb.

Privileges granted commonly to PUBLIC enable all local users to exercise the granted privilege in their respective PDBs and enable all common users to exercise this privilege in the PDBs to which they have access. Oracle recommends that users do not commonly grant privileges and roles to PUBLIC.

4.2.10 Grants of Privileges and Roles: Scenario

In this scenario, SYSTEM creates common user c##dba and tries to give this user privileges to query a table in the hr schema in hrpdb.

The scenario shows how the CONTAINER clause affects grants of roles and privileges. The first column shows operations in CDB$ROOT. The second column shows operations in hrpdb.

Table 4-3 Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation

t1	`SQL> CONNECT SYSTEM@root`
	`Enter password: *******`
	`Connected.`
		n/a	Common user SYSTEM connects to the root container.

t2	`SQL> CREATE USER c##dba`
	`IDENTIFIED BY password`
	`CONTAINER=ALL;`
		n/a	SYSTEM creates common user c##dba. The clause CONTAINER=ALL makes the user a common user.

Table 4-3 (Cont.) Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation
t3	`SQL> GRANT CREATE SESSION TO c##dba;`	n/a	`SYSTEM` grants the `CREATE SESSION` system privilege to `c##dba`. Because the clause `CONTAINER=ALL` is absent, this privilege is granted locally and thus applies *only* to the root, which is the current container.
t4	`SQL> CREATE ROLE c##admin CONTAINER=ALL;`	n/a	`SYSTEM` creates a common role named `c##admin`. The clause `CONTAINER=ALL` makes the role a common role.
t5	`SQL> GRANT SELECT ANY TABLE TO c##admin;` `Grant succeeded.`	n/a	`SYSTEM` grants the `SELECT ANY TABLE` privilege to the `c##admin` role. The absence of the `CONTAINER=ALL` clause makes the privilege local to the root. Thus, this common role contains a privilege that is exercisable only in the root.
t6	`SQL> GRANT c##admin TO c##dba;` `SQL> EXIT;`	n/a	`SYSTEM` grants the `c##admin` role to `c##dba`. Because the `CONTAINER=ALL` clause is absent, the role applies *only* to the current container, even though it is a common role. If `c##dba` connects to a PDB, then `c##dba` does not have this role.

t7	n/a	`SQL> CONNECT c##dba@hrpdb`
		`Enter password: *******`
		`ERROR: ORA-01045: user c##dba lacks CREATE SESSION privilege; logon denied`
			`c##dba` fails to connect to `hrpdb` because the grant at t3 was local to the root.

t8	n/a	`SQL> CONNECT SYSTEM@hrpdb`
		`Enter password: *******`
		`Connected.`
			`SYSTEM` connects to `hrpdb`.

Table 4-3 (Cont.) Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation

t9	n/a	`SQL> GRANT CONNECT, RESOURCE TO c##dba;`
		`Grant succeeded.`
		`SQL> EXIT`
			`SYSTEM` grants the `CONNECT` and `RESOURCE` roles to common user `c##dba`. Because the clause `CONTAINER=ALL` is absent, the grant is local to `hrpdb`.

t10	n/a	`SQL> CONNECT c##dba@hrpdb`
		`Enter password: *******`
		`Connected.`
			Common user `c##dba` connects to `hrpdb`.

t11	n/a	`SQL> SELECT COUNT(*)`
		`FROM hr.employees;`
		`select * from hr.employees`
		`*`
		`ERROR at line 1:`
		`ORA-00942: table or view does not exist`
			The query of `hr.employees` still returns an error because `c##dba` does not have select privileges on tables in `hrpdb`. The `SELECT ANY TABLE` privilege granted locally at t5 is restricted to the root and thus does not apply to `hrpdb`.

t12	`SQL> CONNECT SYSTEM@root`
	`Enter password: *******`
	`Connected.`
		n/a	Common user `SYSTEM` connects to the root container.

t13	`SQL> GRANT SELECT ANY TABLE TO c##admin CONTAINER=ALL;` `Grant succeeded.`	n/a	`SYSTEM` grants the `SELECT ANY TABLE` privilege to the `c##admin` role. The presence of `CONTAINER=ALL` means the privilege is being granted commonly.

t14	n/a	`SQL> SELECT COUNT(*) FROM hr.employees;`
		`select * from hr.employees`
		`*`
		`ERROR at line 1:`
		`ORA-00942: table or view does not exist`
			A query of `hr.employees` still returns an error. The reason is that at t6 the `c##admin` common role was granted to `c##dba` in the root only.

Table 4-3 (Cont.) Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation
t15	`SQL> GRANT c##admin TO c##dba CONTAINER=ALL;` `Grant succeeded.`	n/a	`SYSTEM` grants the common role named `c##admin` to `c##dba`, specifying `CONTAINER=ALL`. Now user `c##dba` has the role in *all* containers, not just the root.

t17	n/a	`SQL> SELECT COUNT(*) FROM hr.employees;`
		
		`COUNT(*)`
		`----------`
		`107`
			The query succeeds.

4.3 Who Should Be Granted Privileges?

You grant privileges to users so they can accomplish tasks required for their jobs.

You should grant a privilege only to a user who requires that privilege to accomplish the necessary work. Excessive granting of unnecessary privileges can compromise security. For example, you never should grant `SYSDBA` or `SYSOPER` administrative privilege to users who do not perform administrative tasks.

You can grant privileges to a user in two ways:

- **You can grant privileges to users explicitly.** For example, you can explicitly grant to user `psmith` the privilege to insert records into the `employees` table.
- **You can 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 users `psmith` and `robert`.

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

See Also:

- Guidelines for Securing User Accounts and Privileges for best practices to follow when granting privileges
- *Oracle Database Vault Administrator's Guide* if you are concerned about excessive privilege grants
- *Oracle Database SQL Language Reference* for the complete list of system privileges and their descriptions

4.4 How the Oracle Multitenant Option Affects Privileges

All users, including common users, can exercise their privileges only within the current container.

However, a user connected to the root can perform certain operations that affect other pluggable databases (PDBs). These operations include `ALTER PLUGGABLE DATABASE`, `CREATE USER`, `CREATE ROLE`, and `ALTER USER`. The common user must possess the commonly granted privileges that enable these operations. A common user connected to the root can see metadata pertaining to PDBs by way of the container data objects (for example, multitenant container database (CDB) views and `V$` views) in the root, provided that the common user has been granted privileges required to access these views and their `CONTAINER_DATA` attribute has been set to allow seeing data about various PDBs. The common user cannot query tables or views in a PDB.

Common users cannot exercise their privileges across other PDBs. They must first switch to the PDB that they want, and then exercise their privileges from there. To switch to a different container, the common user must have the `SET CONTAINER` privilege. The `SET CONTAINER` privilege must be granted either commonly or in the container to which the user is attempting to switch. Alternatively, the common user can start a new database session whose initial current container is the container this user wants, relying on the `CREATE SESSION` privilege in that PDB.

Be aware that commonly granted privileges may interfere with the security configured for individual PDBs. For example, suppose an application PDB database administrator wants to prevent any user in the PDB from modifying a particular application common object. A privilege (such as `UPDATE`) granted commonly to `PUBLIC` or to a common user or common role on the object would circumvent the PDB database administrator's intent.

Related Topics

- Enabling Common Users to View CONTAINER_DATA Object Information
  Common users can view information about `CONTAINER_DATA` objects in the root or for data in specific PDBs.

4.5 Managing Administrative Privileges

Administrative privileges can be used for both general and specific database operations.

- About Administrative Privileges
  For better separation of duty, Oracle Database provides administrative privileges that are tailored for commonly performed specific administrative tasks.
- Grants of Administrative Privileges to Users
  As with all powerful privileges, grant administrative privileges to only trusted users.
- SYSDBA and SYSOPER Privileges for Standard Database Operations
  The `SYSDBA` and `SYSOPER` administrative privileges enable you to perform standard database operations.
- Forcing oracle Users to Enter a Password When Logging in as SYSDBA
  You can force an `oracle` user to enter a password when the user logs in to an Oracle database using the `SYSDBA` administrative privilege.
- SYSBACKUP Administrative Privilege for Backup and Recovery Operations
  The `SYSBACKUP` administrative privilege is used to perform backup and recovery operations from either Oracle Recovery Manager (RMAN) and or through SQL*Plus.

- SYSDG Administrative Privilege for Oracle Data Guard Operations
  You can log in as user SYSDG with the SYSDG administrative privilege to perform Data Guard operations.
- SYSKM Administrative Privilege for Transparent Data Encryption
  The SYSKM administrative privilege enables the SYSKM user to manage Transparent Data Encryption (TDE) wallet operations.
- SYSRAC Administrative Privilege for Oracle Real Application Clusters
  The SYSRAC administrative privilege is used by the Oracle Real Application Clusters (Oracle RAC) Clusterware agent.

4.5.1 About Administrative Privileges

For better separation of duty, Oracle Database provides administrative privileges that are tailored for commonly performed specific administrative tasks.

These tasks include operations for backup and recovery, Oracle Data Guard, and encryption key management for Transparent Data Encryption (TDE).

You can find the administrative privileges that a user has by querying the V$PWFILE_USERS dynamic view, which lists users in the password file.

In previous releases, you needed to have the SYSDBA administrative privilege to perform these tasks. To support backward compatibility, you still can use the SYSDBA privilege for these tasks, but Oracle recommends that you use the administrative privileges described in this section.

Users who have been granted administrative privileges can be altered to be schema-only accounts.

The use of administrative privileges is mandatorily audited.

Related Topics

- Auditing Administrative Users
  You can create unified audit policies to capture the actions of administrative user accounts, such as SYS.

4.5.2 Grants of Administrative Privileges to Users

As with all powerful privileges, grant administrative privileges to only trusted users.

However, be aware that there is a restriction for users whose names have non-ASCII characters (for example, the umlaut in the name HÜBER). You can grant administrative privileges to these users, but if the Oracle database instance is down, the authentication using the granted privilege is not supported if the user name has non-ASCII characters. If the database instance is up, then the authentication is supported.

4.5.3 SYSDBA and SYSOPER Privileges for Standard Database Operations

The SYSDBA and SYSOPER administrative privileges enable you to perform standard database operations.

These database operations can include tasks such as database startups and shutdowns, creating the server parameter file (SPFILE), or altering the database archive log. You can grant the SYSDBA and SYSOPER administrative privileges to application common users (but not to CDB common users).

By default, the underlying schemas for SYSDBA and SYSOPER are dictionary protected. This protection prevents other users from using system privileges (including ANY privileges) on these schemas. In addition, you cannot create objects in these schemas.

You can find if a user has been granted an administrative privilege on a local (PDB) level, for a CDB root, or for an application root by querying the SCOPE column of the V$PWFILE_USERS dynamic view.

You can grant the SYSDBA or SYSOPER administrative privilege to users who have been created with no authentication.

4.5.4 Forcing oracle Users to Enter a Password When Logging in as SYSDBA

You can force an oracle user to enter a password when the user logs in to an Oracle database using the SYSDBA administrative privilege.

1. Edit the $ORACLE_HOME/network/admin/sqlnet.ora file.
2. Set the SQLNET.AUTHENTICATION_SERVICES parameter as follows:

   ```
   sqlnet.authentication_services=none
   ```

   If SQLNET.AUTHENTICATION_SERVICES is not set, then it defaults to ALL.

4.5.5 SYSBACKUP Administrative Privilege for Backup and Recovery Operations

The SYSBACKUP administrative privilege is used to perform backup and recovery operations from either Oracle Recovery Manager (RMAN) and or through SQL*Plus.

By default, the underlying schema for SYSBACKUP is dictionary protected. This protection prevents other users from using system privileges (including ANY privileges) on this schema. In addition, you cannot create objects in this schema.

To connect to the database as SYSBACKUP using a password, you must create a password file for it.

You cannot grant the SYSBACKUP administrative privilege to users who have been created with no authentication.

This privilege enables you to perform the following operations:

- STARTUP
- SHUTDOWN
- ALTER DATABASE
- ALTER SYSTEM
- ALTER SESSION
- ALTER TABLESPACE
- CREATE CONTROLFILE
- CREATE ANY DIRECTORY

- CREATE ANY TABLE
- CREATE ANY CLUSTER
- CREATE PFILE
- CREATE RESTORE POINT (including GUARANTEED restore points)
- CREATE SESSION
- CREATE SPFILE
- DROP DATABASE
- DROP TABLESPACE
- DROP RESTORE POINT (including GUARANTEED restore points)
- FLASHBACK DATABASE
- RESUMABLE
- UNLIMITED TABLESPACE
- SELECT ANY DICTIONARY
- SELECT ANY TRANSACTION
- SELECT
  - X$ tables (that is, the fixed tables)
  - V$ and GV$ views (that is, the dynamic performance views)
  - APPQOSSYS.WLM_CLASSIFIER_PLAN
  - SYSTEM.LOGSTDBY$PARAMETERS
- DELETE/INSERT
  - SYS.APPLY$_SOURCE_SCHEMA
  - SYSTEM.LOGSTDBY$PARAMETERS
- EXECUTE
  - SYS.DBMS_BACKUP_RESTORE
  - SYS.DBMS_RCVMAN
  - SYS.DBMS_DATAPUMP
  - SYS.DBMS_IR
  - SYS.DBMS_PIPE
  - SYS.SYS_ERROR
  - SYS.DBMS_TTS
  - SYS.DBMS_TDB
  - SYS.DBMS_PLUGTS
  - SYS.DBMS_PLUGTSP
- SELECT_CATALOG_ROLE

In addition, the SYSBACKUP privilege enables you to connect to the database even if the database is not open.

Related Topics

- *Oracle Database Administrator's Guide*
- *Oracle Database Backup and Recovery User's Guide*

4.5.6 SYSDG Administrative Privilege for Oracle Data Guard Operations

You can log in as user `SYSDG` with the `SYSDG` administrative privilege to perform Data Guard operations.

By default, the underlying schema for `SYSDG` is dictionary protected. This protection prevents other users from using system privileges (including `ANY` privileges) on this schema. In addition, you cannot create objects in this schema.

You can use this privilege with either Data Guard Broker or the `DGMGRL` command-line interface. In order to connect to the database as `SYSDG` using a password, you must create a password file for it.

You cannot grant the `SYSYSDG` administrative privilege to users who have been created with no authentication.

The `SYSDG` privilege enables the following operations:

- `STARTUP`
- `SHUTDOWN`
- `ALTER DATABASE`
- `ALTER SESSION`
- `ALTER SYSTEM`
- `CREATE RESTORE POINT` (including `GUARANTEED` restore points)
- `CREATE SESSION`
- `DROP RESTORE POINT` (including `GUARANTEED` restore points)
- `FLASHBACK DATABASE`
- `SELECT ANY DICTIONARY`
- `SELECT`
  - `X$` tables (that is, the fixed tables)
  - `V$` and `GV$` views (that is, the dynamic performance views)
  - `APPQOSSYS.WLM_CLASSIFIER_PLAN`
- `DELETE`
  - `APPQOSSYS.WLM_CLASSIFIER_PLAN`
- `EXECUTE`
  - `SYS.DBMS_DRS`

In addition, the `SYSDG` privilege enables you to connect to the database even if it is not open.

Related Topics

- *Oracle Database Administrator's Guide*
- *Oracle Data Guard Concepts and Administration*

4.5.7 SYSKM Administrative Privilege for Transparent Data Encryption

The SYSKM administrative privilege enables the SYSKM user to manage Transparent Data Encryption (TDE) wallet operations.

By default, the underlying schema for SYSKM is dictionary protected. This protection prevents other users from using system privileges (including ANY privileges) on this schema. In addition, you cannot create objects in this schema

In order to connect to the database as SYSKM using a password, you must create a password file for it.

You cannot grant the SYSKM administrative privilege to users who have been created with no authentication.

The SYSKM administrative privilege enables the following operations:

- ADMINISTER KEY MANAGEMENT
- CREATE SESSION
- SELECT (only when database is open)
  - SYS.V$ENCRYPTED_TABLESPACES
  - SYS.V$ENCRYPTION_WALLET
  - SYS.V$WALLET
  - SYS.V$ENCRYPTION_KEYS
  - SYS.V$CLIENT_SECRETS
  - SYS.DBA_ENCRYPTION_KEY_USAGE

In addition, the SYSKM privilege enables you to connect to the database even if it is not open.

Related Topics

- *Oracle Database Administrator's Guide*
- *Oracle Database Advanced Security Guide*

4.5.8 SYSRAC Administrative Privilege for Oracle Real Application Clusters

The SYSRAC administrative privilege is used by the Oracle Real Application Clusters (Oracle RAC) Clusterware agent.

By default, the underlying schema for SYSRAC is dictionary protected. This protection prevents other users from using system privileges (including ANY privileges) on this schema. In addition, you cannot create objects in this schema.

The SYSRAC administrative privilege provides only the minimal privileges necessary for performing day-to-day Oracle RAC operations. For example, this privilege is used for Oracle RAC utilities such as SRVCTL.

You cannot grant the SYSRAC administrative privilege to users who have been created with no authentication.

The SYSRAC administrative privilege enables the following operations:

- STARTUP

- SHUTDOWN
- ALTER DATABASE MOUNT
- ALTER DATABASE OPEN
- ALTER DATABASE OPEN READ ONLY
- ALTER DATABASE CLOSE NORMAL
- ALTER DATABASE DISMOUNT
- ALTER SESSION SET EVENTS
- ALTER SESSION SET _NOTIFY_CRS
- ALTER SESSION SET CONTAINER
- ALTER SYSTEM REGISTER
- ALTER SYSTEM SET *local_listener|remote_listener|listener_networks*

In addition to these privileges, the SYSRAC user will have access to the following views:

- V$PARAMETER
- V$DATABASE
- V$PDBS
- CDB_SERVICE$
- DBA_SERVICES
- V$ACTIVE_SERVICES
- V$SERVICES

The SYSRAC user is also granted the EXECUTE privilege for the following PL/SQL packages:

- DBMS_DRS
- DBMS_SERVICE
- DBMS_SERVICE_PRVT
- DBMS_SESSION
- DBMS_HA_ALERTS_PRVT
- Dequeue messaging SYS.SYS$SERVICE_METRICS

Related Topics

- *Oracle Database Administrator's Guide*
- *Oracle Real Application Clusters Administration and Deployment Guide*

4.6 Managing System Privileges

To perform actions on schema objects, you must be granted the appropriate system privileges.

- About System Privileges
  A system privilege is the right to perform an action or to perform actions on schema objects.

- Who Can Grant or Revoke System Privileges?
  Only two types of users can grant system privileges to other users or revoke those privileges from them.
- Why Is It Important to Restrict System Privileges?
  System privileges are very powerful, so only grant them to trusted users. You should also secure the data dictionary and `SYS` schema objects.
- Grants and Revokes of System Privileges
  You can grant or revoke system privileges to users and roles.
- About ANY Privileges and the PUBLIC Role
  System privileges that use the `ANY` keyword enable you to set privileges for an entire category of objects in the database.

4.6.1 About System Privileges

A system privilege is the right to perform an action or to perform actions on schema objects.

For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.

There are many different kinds of system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations. *Remember that system privileges are very powerful.* Only grant them when necessary to roles and trusted users of the database. To find the system privileges that have been granted to a user, you can query the `DBA_SYS_PRIVS` data dictionary view.

If you want to restrict a system privilege to a specific schema, then you can do so by granting it as a schema privilege. A schema privilege enables you to grant a specific system privilege on a schema without having to perform a grant on every object within the schema.

System privileges such as `SELECT ANY TABLE` do not work on `SYS` objects or other objects that are owned by schemas that are marked as `DICTIONARY PROTECTED`.

Related Topics

- How Commonly Granted System Privileges Work
  Users can exercise system privileges only within the PDB in which they were granted.
- Managing Schema Privileges
  Schema privileges enable certain system privileges to be granted on a schema.
- *Oracle Database SQL Language Reference*

4.6.2 Who Can Grant or Revoke System Privileges?

Only two types of users can grant system privileges to other users or revoke those privileges from them.

These users are as follows:

- Users who were granted a specific system privilege with the `ADMIN OPTION`
- Users with the system privilege `GRANT ANY PRIVILEGE`

For this reason, only grant these privileges to trusted users.

4.6.3 Why Is It Important to Restrict System Privileges?

System privileges are very powerful, so only grant them to trusted users. You should also secure the data dictionary and SYS schema objects.

- About the Importance of Restricting System Privileges
  System privileges are very powerful, so by default the database is configured to prevent typical (non-administrative) users from exercising the ANY system privileges.
- User Access to Objects in the SYS Schema
  Users with explicit object privileges or those who connect with administrative privileges (SYSDBA) can access objects in the SYS schema.

4.6.3.1 About the Importance of Restricting System Privileges

System privileges are very powerful, so by default the database is configured to prevent typical (non-administrative) users from exercising the ANY system privileges.

For example, users are prevented from exercising ANY system privileges such as UPDATE ANY TABLE on the data dictionary.

Related Topics

- Guidelines for Securing User Accounts and Privileges
  Oracle provides guidelines to secure user accounts and privileges.

4.6.3.2 User Access to Objects in the SYS Schema

Users with explicit object privileges or those who connect with administrative privileges (SYSDBA) can access objects in the SYS schema.

The following table lists roles that you can grant to users who need access to objects in the SYS schema.

Table 4-4 Roles to Allow Access to SYS Schema Objects

Role	Description
SELECT_CATALOG_ROLE	Grant this role to allow users SELECT privileges on data dictionary views.
EXECUTE_CATALOG_ROLE	Grant this role to allow users EXECUTE privileges for packages and procedures in the data dictionary.

Additionally, you can grant the SELECT ANY DICTIONARY system privilege to users who require access to tables created in the SYS schema. This system privilege allows query access to any object in the SYS schema, including tables created in that schema. It must be granted individually to each user requiring the privilege. It is not included in GRANT ALL PRIVILEGES, but it can be granted through a role.

> **Note:**
>
> You should grant these roles and the `SELECT ANY DICTIONARY` system privilege with extreme care, because the integrity of your system can be compromised by their misuse.

4.6.4 Grants and Revokes of 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 exercise system privileges. For example, roles permit privileges to be made selectively available. Ensure that you follow separation of duty guidelines for securing roles.

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

- `GRANT` and `REVOKE` SQL statements
- Oracle Enterprise Manager Cloud Control

Related Topics

- Guidelines for Securing Roles
  Oracle provides guidelines for role management.
- User Privilege and Role Data Dictionary Views
  You can use special queries to find information about various types of privilege and role grants.

4.6.5 About ANY Privileges and the PUBLIC Role

System privileges that use the `ANY` keyword enable you to set privileges for an entire category of objects in the database.

For example, the `CREATE ANY PROCEDURE` system privilege permits a user to create a procedure anywhere in the database. The behavior of an object created by users with the `ANY` privilege is not restricted to the schema in which it was created. For example, if user `JSMITH` has the `CREATE ANY PROCEDURE` privilege and creates a procedure in the schema `JONES`, then the procedure will run as `JONES`. However, `JONES` may not be aware that the procedure `JSMITH` created is running as `JONES`. If `JONES` has `DBA` privileges, letting `JSMITH` run a procedure as `JONES` could pose a security violation.

The `PUBLIC` role is a special role that every database user account automatically has when the account is created. By default, it has no privileges granted to it, but it does have numerous grants, mostly to Java objects. You cannot drop the `PUBLIC` role, and a manual grant or revoke of this role has no meaning, because the user account will always assume this role. Because all database user accounts assume the `PUBLIC` role, it does not appear in the `DBA_ROLES` and `SESSION_ROLES` data dictionary views.

You can grant privileges to the `PUBLIC` role, but remember that this makes the privileges available to every user in the Oracle database. For this reason, be careful about granting privileges to the `PUBLIC` role, particularly powerful privileges such as the `ANY` privileges and system privileges. For example, if `JSMITH` has the `CREATE PUBLIC SYNONYM` system privilege, `JSMITH` could redefine an interface that they know everyone else uses, and then point to it with

the PUBLIC SYNONYM that JSMITH created. Instead of accessing the correct interface, users would access the interface of JSMITH, which could possibly perform illegal activities such as stealing the login credentials of users.

These types of privileges are very powerful and could pose a security risk if given to the wrong person. Be careful about granting privileges using ANY or PUBLIC. As with all privileges, you should follow the principles of "least privilege" when granting these privileges to users.

Related Topics

- Guidelines for Securing a Database Installation and Configuration
  Oracle provides guidelines to secure the database installation and configuration.

4.7 Managing Schema Privileges

Schema privileges enable certain system privileges to be granted on a schema.

- About Managing Schema Privileges
  When a schema privilege is granted on a schema, the grantee has the system privilege on all the objects in the schema on which the grant has been made.
- Privileges That Are Excluded from Schema Privilege Grants
  Many administrative and system privileges cannot be used in schema privilege grants.
- Granting a Schema Privilege
  You can use the GRANT statement to grant a schema privilege to a user or a role.
- Revoking a Schema Privilege
  You can use the REVOKE statement to revoke a schema privilege from a user or a role.

4.7.1 About Managing Schema Privileges

When a schema privilege is granted on a schema, the grantee has the system privilege on all the objects in the schema on which the grant has been made.

The system privilege applies to both current and future objects in the schema. For example, suppose you grant the CREATE ANY TABLE system privilege to user psmith for use on the HR schema. User psmith is then able to create tables in the HR schema and not in any other schema for which psmith does not have permission. You can grant the schema privilege to either users or roles. Schema privilege grants can be used on a wide range of system privileges, though not all. In addition, you cannot use schema privileges on the SYS schema. Because this grant provides powerful privileges to the grantee, ensure that you grant the schema privilege to trusted users only.

Granting users schema privileges has the following benefits:

- Granting schema privileges instead of system privileges allows use of the principle of least privilege. Granting a system privilege could be unnecessarily permissive, because it allows the same privilege on any object in any schema in the database, whereas by granting only a schema privilege to a user or role, the user or role would be granted the least privilege necessary to accomplish their task. Hence, this approach makes the database more secure.
- This type of privilege grant makes the granting of privileges much easier. Rather than having to grant the system or object privilege individually to a user, an administrator can grant the privilege to the schema so that all objects within the schema are accessible to the user.

To grant or revoke schema privileges, you must have the `GRANT ANY SCHEMA PRIVILEGE` or the `GRANT ANY PRIVILEGE` system privilege.

The `ANY` system privileges that you can include in the schema grants cover operations such as creation, altering, executing, dropping of objects.

The *Oracle Database SQL Language Reference* provides a list of the available system privileges that you can grant as schema privileges.

To find information about schema privilege grants, query the following data dictionary views:

- `DBA_SCHEMA_PRIVS`
- `ROLE_SCHEMA_PRIVS`
- `USER_SCHEMA_PRIVS`
- `SESSION_SCHEMA_PRIVS`
- `V$ENABLEDSCHEMAPRIVS`

Related Topics

- Privileges That Are Excluded from Schema Privilege Grants
  Many administrative and system privileges cannot be used in schema privilege grants.
- Administering Schema Security Policies
  To manage schema security policies for row level security, fine-grained auditing, and Oracle Data Redaction, users must be granted the appropriate system privilege.
- Data Dictionary Views to Find Information about Privilege and Role Grants
  Oracle Database provides data dictionary views that describe privilege and role grants.

4.7.2 Privileges That Are Excluded from Schema Privilege Grants

Many administrative and system privileges cannot be used in schema privilege grants.

The following administrative privileges are excluded from schema privilege grants:

- `SYSDBA`
- `SYSOPER`
- `SYSASM`
- `SYSBACKUP`
- `SYSDG`
- `SYSKM`

The following table lists system privileges that are excluded from schema privilege grants.

Table 4-5 System Privileges Excluded from Schema Privileges

System Privilege Type	Privilege
Advisor framework	• `ADVISOR` • `ADMINISTER SQL TUNING SET`
Application context	• `CREATE ANY CONTEXT` • `DROP ANY CONTEXT`

Table 4-5 (Cont.) System Privileges Excluded from Schema Privileges

System Privilege Type	Privilege
Application continuity	• KEEP DATE TIME • KEEP SYSGUID
Database change notification	• CHANGE NOTIFICATION
Database links	• CREATE DATABASE LINK • CREATE PUBLIC DATABASE LINK • DROP PUBLIC DATABASE LINK
Database triggers	• ADMINISTER DATABASE TRIGGER
Debugging	• DEBUG CONNECT SESSION
Dictionary protection	• SELECT ANY DICTIONARY • ANALYZE ANY DICTIONARY
Directories	• CREATE ANY DIRECTORY • DROP ANY DIRECTORY • READ • WRITE
Editions	• CREATE ANY EDITION • DROP ANY EDITION
Exports and imports	• EXPORT FULL DATABASE • IMPORT FULL DATABASE
Flashback	• FLASHBACK ARCHIVE ADMINISTER • SELECT ANY TRANSACTION
Key management	• ADMINISTER KEY MANAGEMENT
Logminer	• LOGMINING
Plan management	• ADMINISTER SQL MANAGEMENT OBJECT
Pluggable databases	• CREATE PLUGGABLE DATABASE • SET CONTAINER
Profiles	• CREATE PROFILE • ALTER PROFILE • DROP PROFILE
Public synonyms	• CREATE PUBLIC SYNONYM • DROP PUBLIC SYNONYM
Recycle bin	• PURGE DBA_RECYCLEBIN
Resource management	• ADMINISTRATE RESOURCE MANAGER
Resumable space allocation	• RESUMABLE
Roles	• CREATE ROLE • DROP ANY ROLE • GRANT ANY ROLE • ALTER ANY ROLE

Table 4-5 (Cont.) System Privileges Excluded from Schema Privileges

System Privilege Type	Privilege
Rollback segment	• CREATE ROLLBACK SEGMENT • ALTER ROLLBACK SEGMENT • DROP ROLLBACK SEGMENT
Sessions	• CREATE SESSION • ALTER SESSION • RESTRICT SESSION
Stored outlines	• CREATE ANY OUTLINE • ALTER ANY OUTLINE • DROP ANY OUTLINE
System	• ALTER DATABASE • ALTER SYSTEM • AUDIT SYSTEM • ALTER RESOURCE COST
Tablespaces	• CREATE TABLESPACE • ALTER TABLESPACE • MANAGE TABLESPACE • DROP TABLESPACE • UNLIMITED TABLESPACE
Transactions	• FORCE TRANSACTION • FORCE ANY TRANSACTION
Users	• CREATE USER • BECOME USER • ALTER USER • DROP USER

4.7.3 Granting a Schema Privilege

You can use the `GRANT` statement to grant a schema privilege to a user or a role.

1. Log in to the CDB root or to a PDB as a user who has been granted the `GRANT ANY SCHEMA PRIVILEGE` or `GRANT ANY PRIVILEGE` system privilege.
2. To find the available schema privileges that you can grant, see *Oracle Database SQL Language Reference*
3. Grant the schema privilege to the user or role.

   For example, suppose you grant the `SELECT ANY TABLE` system privilege to user `psmith` for use on the `HR` schema. User `psmith` is then able to select from existing and future tables that are created in the `HR` schema.

   ```
   GRANT SELECT ANY TABLE ON SCHEMA HR TO psmith;
   ```

   If you have the `GRANT ANY SCHEMA PRIVILEGE WITH ADMIN OPTION` privilege, then you can do two additional types of grants:

   - Grant `GRANT ANY SCHEMA PRIVILEGE` to another user.

- Grant a schema privilege `WITH ADMIN OPTION`, so that the user can grant the schema privilege to another user.

4.7.4 Revoking a Schema Privilege

You can use the `REVOKE` statement to revoke a schema privilege from a user or a role.

1. Log in to the CDB root or to a PDB as a user who has been granted the `GRANT ANY SCHEMA PRIVILEGE` system privilege with `WITH ADMIN OPTION`.
2. To find the schema privileges that have been granted to the user or role, run a query similar to the following:

   For example:

   ```
   SELECT PRIVILEGE, SCHEMA FROM DBA_SCHEMA_PRIVS
   WHERE GRANTEE = 'PSMITH';
   ```

   Output similar to the following appears:

   ```
   PRIVILEGE             SCHEMA
   --------------------- ------------
   SELECT ANY TABLE      HR
   ```

3. Revoke the schema privileges from the user or role.

   For example, to revoke the `SELECT ANY TABLE` schema privilege from user `psmith`:

   ```
   REVOKE SELECT ANY TABLE ON SCHEMA HR FROM psmith;
   ```

4.8 Administering Schema Security Policies

To manage schema security policies for row level security, fine-grained auditing, and Oracle Data Redaction, users must be granted the appropriate system privilege.

- About Administering Schema System Security Policies
  Security policies for row level security, fine-grained auditing, and Oracle Data Redaction require special schema-related system privileges.
- Granting an Administrator Schema Security Policy
  You can use the `GRANT` statement to grant a schema system privilege to a user or role.
- Revoking an Administrator Security Policy
  You can use the `REVOKE` statement to revoke a schema system privilege from a user or role.

4.8.1 About Administering Schema System Security Policies

Security policies for row level security, fine-grained auditing, and Oracle Data Redaction require special schema-related system privileges.

The system privileges and their corresponding PL/SQL packages that the user must be granted are as follows:

- `ADMINISTER ROW LEVEL SECURITY POLICY` system privilege, for use with the `DBMS_RLS` PL/SQL package
- `ADMINISTER FINE GRAINED AUDIT POLICY` system privilege, for use with the `DBMS_FGA` PL/SQL package
- `ADMINISTER REDACTION POLICY` system privilege, for use with the `DBMS_REDACT` PL/SQL package

You must grant the system privilege to the user in addition to the other required privileges that are needed for the security policy, such as the `EXECUTE` privilege on any PL/SQL packages. You can grant the system privilege in either of the following ways:

- If the security policy is to apply to all non-`SYS` schemas across the database, then use the following syntax:

  ```
  GRANT system_privilege TO grantee;
  ```

- If the security policy is to be restricted to a specific schema, then use this syntax:

  ```
  GRANT system_privilege ON SCHEMA schema TO grantee;
  ```

4.8.2 Granting an Administrator Schema Security Policy

You can use the `GRANT` statement to grant a schema system privilege to a user or role.

1. Log in to the CDB root or to a PDB as a user who has been granted the `GRANT ANY SCHEMA PRIVILEGE` system privilege with `WITH ADMIN OPTION`.
2. Grant the user the `EXECUTE` privilege on the PL/SQL package (and any other necessary privileges) to administer the security policy.

   For example, for a user who is responsible for creating row level security policies:

   ```
   GRANT EXECUTE ON DBMS_RLS TO preston;
   ```

3. Grant the user the schema system privilege.

   For example, to restrict row level security policies to the `HR` schema:

   ```
   GRANT ADMINISTER ROW LEVEL SECURITY POLICY ON SCHEMA HR TO preston;
   ```

   To enable the user to create the policy in any non-`SYS` schema in the database:

   ```
   GRANT ADMINISTER ROW LEVEL SECURITY POLICY TO preston;
   ```

4.8.3 Revoking an Administrator Security Policy

You can use the `REVOKE` statement to revoke a schema system privilege from a user or role.

1. Log in to the CDB root or to a PDB as a user who has been granted the `GRANT ANY SCHEMA PRIVILEGE` system privilege with `WITH ADMIN OPTION`.
2. To find the system privileges that have been granted to the user or role, run a query similar to the following:

For example:

```
SELECT PRIVILEGE FROM DBA_SYS_PRIVS_ALL WHERE GRANTEE = 'PRESTON';
```

Output similar to the following appears:

```
PRIVILEGE
----------------------------------
ADMINISTER ROW LEVEL SECURITY POLICY
```

3. Revoke the system privilege from the user or role.

   For example:

   ```
   REVOKE ADMINISTER ROW LEVEL SECURITY POLICY ON SCHEMA HR FROM preston;
   ```

   Or:

   ```
   REVOKE ADMINISTER ROW LEVEL SECURITY POLICY FROM preston;
   ```

4. Revoke any other privileges as necessary, such as the `EXECUTE` privilege on the associated PL/SQL package.

   For example:

   ```
   REVOKE EXECUTE ON DBMS_RLS FROM preston;
   ```

4.9 Managing Privileges to Enable Diagnostics

Only users who have the `SYSDBA` administrative privilege or the `ENABLE_DIAGNOSTICS` system privilege can enable diagnostics.

The kinds of diagnostics that you can restrict control of include the following: debug-events (events++, error-numbers) and debug-actions through `ALTER SESSION` and `ALTER SYSTEM` operations.

- To control the ability of users to perform these types of diagnostics, set the `DIAGNOSTICS_CONTROL` initialization parameter in the initialization file.

  `DIAGNOSTICS_CONTROL` values are as follows:

  - `ERROR`: If a user who does not have the `SYSDBA` or `ENABLE DIAGNOSTICS` privilege attempts to enable a diagnostic, then the attempt will fail and an `ORA-01031: insufficient privileges` error appears.
  - `WARNING`: A user who does not have the `SYSDBA` or `ENABLE DIAGNOSTICS` privilege will be able to enable a diagnostic, but a warning message is written to an alert log. The warning message is similar to the following:

    ```
    User 'USERNAME' has set the following debug-event(s) on the event-group
    'session':

    1357 trace name context forever, level 2
    ```

    In this message, the `session` keyword is used if the user run an `ALTER SESSION` statement. If the user runs an `ALTER SYSTEM` statement, then the keyword is `system`.

  - `IGNORE`: The user can perform the diagnostic task without any error messages appearing. This setting is the default.

4.10 Managing Commonly and Locally Granted Privileges

Privileges can be granted commonly for an entire CDB or application container, or granted locally to a specific PDB.

- About Commonly and Locally Granted Privileges
  Both common users and local users can grant privileges to one another.
- How Commonly Granted System Privileges Work
  Users can exercise system privileges only within the PDB in which they were granted.
- How Commonly Granted Object Privileges Work
  Object privileges on common objects applies to the object as well as all associated links on this common object.
- Granting or Revoking Privileges to Access a PDB
  You can grant and revoke privileges for PDB access.
- Example: Granting a Privilege to a Common User
  You must use the `GRANT` statement in the root to grant privileges to a common user.
- Enabling Common Users to View CONTAINER_DATA Object Information
  Common users can view information about `CONTAINER_DATA` objects in the root or for data in specific PDBs.

4.10.1 About Commonly and Locally Granted Privileges

Both common users and local users can grant privileges to one another.

Privileges by themselves are neither common nor local. How the privileges are applied depends on whether the privilege is granted commonly or granted locally.

For commonly granted privileges:

- A privilege that is granted commonly can be used in every existing and future container.
- Only common users can grant privileges commonly, and only if the grantee is common.
- A common user can grant privileges to another common user or to a common role.
- The grantor must be connected to the root and must specify `CONTAINER=ALL` in the `GRANT` statement.
- Both system and object privileges can be commonly granted. (Object privileges become actual only with regard to the specified object.)
- When a common user connects to or switches to a given container, this user's ability to perform various activities (such as creating a table) is controlled by privileges granted commonly as well as privileges granted locally in the given container.
- Do not grant privileges to `PUBLIC` commonly.

For locally granted privileges:

- A privilege granted locally can be used only in the container in which it was granted. When the privilege is granted in the root, it applies only to the root.
- Both common users and local users can grant privileges locally.
- A common user and a local user can grant privileges to other common or local roles.

- The grantor must be connected to the container and must specify `CONTAINER=CURRENT` in the `GRANT` statement.
- Any user can grant a privilege locally to any other user or role (both common and local) or to the `PUBLIC` role.

Related Topics

- *Oracle Multitenant Administrator's Guide*
- How the PUBLIC Role Works in a Multitenant Environment
  All privileges that Oracle grants to the `PUBLIC` role are granted locally.

4.10.2 How Commonly Granted System Privileges Work

Users can exercise system privileges only within the PDB in which they were granted.

For example, if a system privilege is locally granted to a common user `c##hr_admin` in the PDB `hr_pdb`, user `c##hr_admin` can exercise that privilege only while connected to PDB `hr_pdb`.

System privileges can apply in the root and in all existing and future PDBs if the following requirements are met:

- The system privilege grantor is a common user and the grantee is a common user, a common role, or the `PUBLIC` role. Do not commonly grant system privileges to the `PUBLIC` role, because this in effect makes the system privilege available to all users.
- The system privilege grantor possesses the `ADMIN OPTION` for the commonly granted privilege
- The `GRANT` statement must contain the `CONTAINER=ALL` clause.

The following example shows how to commonly grant a privilege to the common user `c##hr_admin`.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT CREATE ANY TABLE TO c##hr_admin CONTAINER=ALL;
```

4.10.3 How Commonly Granted Object Privileges Work

Object privileges on common objects applies to the object as well as all associated links on this common object.

These links include all metadata links, data links (previously called object links), or extended data links that are associated with it in the root and in all PDBs belonging to the container (including future PDBs) if certain requirements are met.

These requirements are as follows:

- The object privilege grantor is a common user and the grantee is a common user, a common role, or the `PUBLIC` role.
- The object privilege grantor possesses the commonly granted `GRANT OPTION` for the privilege
- The `GRANT` statement contains the `CONTAINER=ALL` clause.

The following example shows how to grant an object privilege to the common user c##hr_admin so that they can select from the DBA_PDBS view in the CDB root or in any of the associated PDBs that they can access.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT SELECT ON DBA_OBJECTS TO c##hr_admin
CONTAINER=ALL;
```

Related Topics

- *Oracle Multitenant Administrator's Guide*
- How the PUBLIC Role Works in a Multitenant Environment
  All privileges that Oracle grants to the PUBLIC role are granted locally.

4.10.4 Granting or Revoking Privileges to Access a PDB

You can grant and revoke privileges for PDB access.

To grant or revoke a privilege in a PDB, include the CONTAINER clause in the GRANT or REVOKE statement.
Setting CONTAINER to ALL applies the privilege to all existing and future containers; setting it to CURRENT applies the privilege to the local container only. Omitting the CONTAINER clause applies the privilege to the local container. If you issue the GRANT statement from the root and omit the CONTAINER clause, then the privilege is applied locally.

Related Topics

- *Oracle Database SQL Language Reference*

4.10.5 Example: Granting a Privilege to a Common User

You must use the GRANT statement in the root to grant privileges to a common user.

Example 4-3 shows how to commonly grant the CREATE TABLE privilege to common user c##hr_admin so that this user can use this privilege in all existing and future containers.

Example 4-3 Granting a Privilege in a Multitenant Environment

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT CREATE TABLE TO c##hr_admin CONTAINER=ALL;
```

4.10.6 Enabling Common Users to View CONTAINER_DATA Object Information

Common users can view information about CONTAINER_DATA objects in the root or for data in specific PDBs.

- Viewing Data About the Root, CDB, and PDBs While Connected to the Root
  You can restrict view information for the X$ table and the V$, GV$ and CDB_* views when common users perform queries.

- Enabling Common Users to Query Data in Specific PDBs
  You can enable common users to access data pertaining to specific PDBs by adjusting the users' CONTAINER_DATA attribute.

4.10.6.1 Viewing Data About the Root, CDB, and PDBs While Connected to the Root

You can restrict view information for the X$ table and the V$, GV$ and CDB_* views when common users perform queries.

The X$ table and these views contain information about the application root and its associated application PDBs or, if you are connected to the CDB root, the entire CDB.
Restricting this information is useful when you do not want to expose sensitive information about other PDBs. To enable this functionality, Oracle Database provides these tables and views as container data objects. You can find if a specific table or view is a container data object by querying the TABLE_NAME, VIEW_NAME, and CONTAINER_DATA columns of the USER_|DBA_|ALL_VIEWS|TABLES dictionary views.

To find information about the default (user-level) and object-specific CONTAINER_DATA attributes:

1. In SQL*Plus or SQL Developer, log in to the root.
2. Query the CDB_CONTAINER_DATA data dictionary view.

   For example:

```
COLUMN USERNAME FORMAT A13
COLUMN DEFAULT_ATTR FORMAT A7
COLUMN OWNER FORMAT A11
COLUMN OBJECT_NAME FORMAT A11
COLUMN ALL_CONTAINERS FORMAT A3
COLUMN CONTAINER_NAME FORMAT A10
COLUMN CON_ID FORMAT A6

SELECT USERNAME, DEFAULT_ATTR, OWNER, OBJECT_NAME,
       ALL_CONTAINERS, CONTAINER_NAME, CON_ID
FROM   CDB_CONTAINER_DATA
ORDER BY OBJECT_NAME;

USERNAME    DEFAULT OWNER OBJECT_NAME ALL CONTAINERS CON_ID
----------- ------- ----- ----------- --- ---------- ------
C##HR_ADMIN N       SYS   V$SESSION   N   CDB$ROOT        1
C##HR_ADMIN N       SYS   V$SESSION   N   SALESPDB        1
C##HR_ADMIN Y                         N   HRPDB           1
C##HR_ADMIN Y                         N   CDB$ROOT        1
DBSNMP      Y                         Y                   1
SYSTEM      Y                         Y                   1
```

Related Topics

- *Oracle Database Reference*

4.10.6.2 Enabling Common Users to Query Data in Specific PDBs

You can enable common users to access data pertaining to specific PDBs by adjusting the users' `CONTAINER_DATA` attribute.

To enable common users to access data about specific PDBs:

- Issue the `ALTER USER` statement in the root.

Example 4-4 Setting the CONTAINER_DATA Attribute

This example shows how to issue the `ALTER USER` statement to enable the common user `c##hr_admin` to view information pertaining to the `CDB$ROOT`, `SALES_PDB`, and `HRPDB` containers in the `V$SESSION` view (assuming this user can query that view).

```
CONNECT SYSTEM
Enter password: password
Connected.

ALTER USER c##hr_admin
SET CONTAINER_DATA = (CDB$ROOT, SALESPDB, HRPDB)
FOR V$SESSION CONTAINER=CURRENT;
```

In this specification:

- `SET CONTAINER_DATA` lists containers, data pertaining to which can be accessed by the user.
- `FOR V$SESSION` specifies the `CONTAINER_DATA` dynamic view, which common user `c##hr_admin` will query.
- `CONTAINER = CURRENT` must be specified because when you are connected to the root, `CONTAINER=ALL` is the default for the `ALTER USER` statement, but modification of the `CONTAINER_DATA` attribute must be restricted to the root.

If you want to enable user `c##hr_admin` to view information that pertains to the `CDB$ROOT`, `SALES_PDB`, `HRPDB` containers in all `CONTAINER_DATA` objects that this user can access, then omit `FOR V$SESSION`. For example:

```
ALTER USER c##hr_admin
SET CONTAINER_DATA = (CDB$ROOT, SALESPDB, HRPDB)
CONTAINER=CURRENT;
```

Related Topics

- *Oracle Database SQL Language Reference*

4.11 Managing User Roles

A user role is a named collection of privileges that you can create and assign to other users.

- About User Roles
  User roles are useful in a variety of situations, such as restricting DDL usage.
- Predefined Roles in an Oracle Database Installation
  Oracle Database provides a set of predefined roles to help in database administration.

- Creating a Role
  You can create a role that is authenticated with or without a password. You also can create external or global roles.
- Specifying the Type of Role Authorization
  You can configure a role to be authorized through different sources, such the database or an external source.
- Granting and Revoking Roles
  You can grant or revoke privileges to and from roles, and then grant these roles to users or to other roles.
- Dropping Roles
  Dropping a role affects the security domains of users or roles who had been granted the role.
- Restricting SQL*Plus Users from Using Database Roles
  You should restrict SQL*Plus users from using database roles, which helps to safeguard the database from intruder attacks.
- Role Privileges and Secure Application Roles
  A secure application role can be enabled only by an authorized PL/SQL package or procedure.

4.11.1 About User Roles

User roles are useful in a variety of situations, such as restricting DDL usage.

- What Are User Roles?
  A user **role** is a named group of related privileges that you can grant as a group to users or other roles.
- The Functionality of Roles
  Roles are useful for quickly and easily granting permissions to users.
- Properties of Roles and Why They Are Advantageous
  Roles have special properties that make their management very easy, such reduced privilege administration.
- Typical Uses of Roles
  In general, you create a role to manage privileges.
- Common Uses of Application Roles
  You can use application roles to control privileges to use applications.
- Common Uses of User Roles
  You can create a user role for a group of database users with common privilege grant requirements.
- How Roles Affect the Scope of a User's Privileges
  Each role and user has its own unique security domain.
- How Roles Work in PL/SQL Blocks
  Role behavior in a PL/SQL block is determined by the type of block and by definer's rights or invoker's rights.
- How Roles Aid or Restrict DDL Usage
  A user requires one or more privileges to successfully run a DDL statement, depending on the statement.
- How Operating Systems Can Aid Roles
  In some environments, you can administer database security using the operating system.

- How Roles Work in a Distributed Environment
  In a distributed database environment, all necessary roles must be set as the default role for a distributed (remote) session.

4.11.1.1 What Are User Roles?

A user **role** is a named group of related privileges that you can grant as a group to users or other roles.

Managing and controlling privileges is easier when you use **roles**.

Within a database, each role name must be unique, different from all user names and all other role names. 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.

Related Topics

- Managing Common Roles and Local Roles
  A common role is a role that is created in the root; a local role is created in a PDB.

4.11.1.2 The Functionality of Roles

Roles are useful for quickly and easily granting permissions to users.

Although you can use Oracle Database-defined roles, you have more control and continuity if you create your own roles that contain only the privileges pertaining to your requirements. Oracle may change or remove the privileges in an Oracle Database-defined role.

Roles have the following functionality:

- A role can be granted system or object privileges.
- 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 Database allows database applications and users to enable and disable roles to provide selective availability of 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 `role1` cannot be granted to role `role2` if role `role2` has previously been granted to role `role1`.
- If a role is not password authenticated or a secure application role, then you can grant the role indirectly to the user. An indirectly granted role is a role granted to the user through another role that has already been granted to this user. For example, suppose you grant user `psmith` the `role1` role. Then you grant the `role2` and `role3` roles to the `role1` role. Roles `role2` and `role3` are now under `role1`. This means `psmith` has been indirectly granted the roles `role2` and `role3`, in addition to the direct grant of `role1`. Enabling the direct `role1` for `psmith` enables the indirect roles `role2` and `role3` for this user as well.
- Optionally, you can make a directly granted role a default role. You enable or disable the default role status of a directly granted role by using the `DEFAULT ROLE` clause of the `ALTER USER` statement. Ensure that the `DEFAULT ROLE` clause refers only to roles that have been directly granted to the user. To find the directly granted roles for a user, query the `DBA_ROLE_PRIVS` data dictionary view. This view does not include the user's indirectly granted roles. To find roles that are granted to other roles, query the `ROLE_ROLE_PRIVS` view.

- If the role is password authenticated or a secure application role, then you cannot grant it indirectly to the user, nor can you make it a default role. You only can grant this type of role directly to the user. Typically, you enable password authenticated or secure application roles by using the `SET ROLE` statement.

4.11.1.3 Properties of Roles and Why They Are Advantageous

Roles have special properties that make their management very easy, such reduced privilege administration.

Table 4-6 describes the properties of roles that enable easier privilege management within a database.

Table 4-6 Properties of Roles and Their Description

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 must be granted to each member of the group.
Dynamic privilege management	If the privileges of a group must change, then 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 run the application by way of a given user name.
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. You should grant a secure application role all privileges necessary to run the application. You then can grant 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 role.

Related Topics

- How Roles Aid or Restrict DDL Usage
  A user requires one or more privileges to successfully run a DDL statement, depending on the statement.

4.11.1.4 Typical Uses of Roles

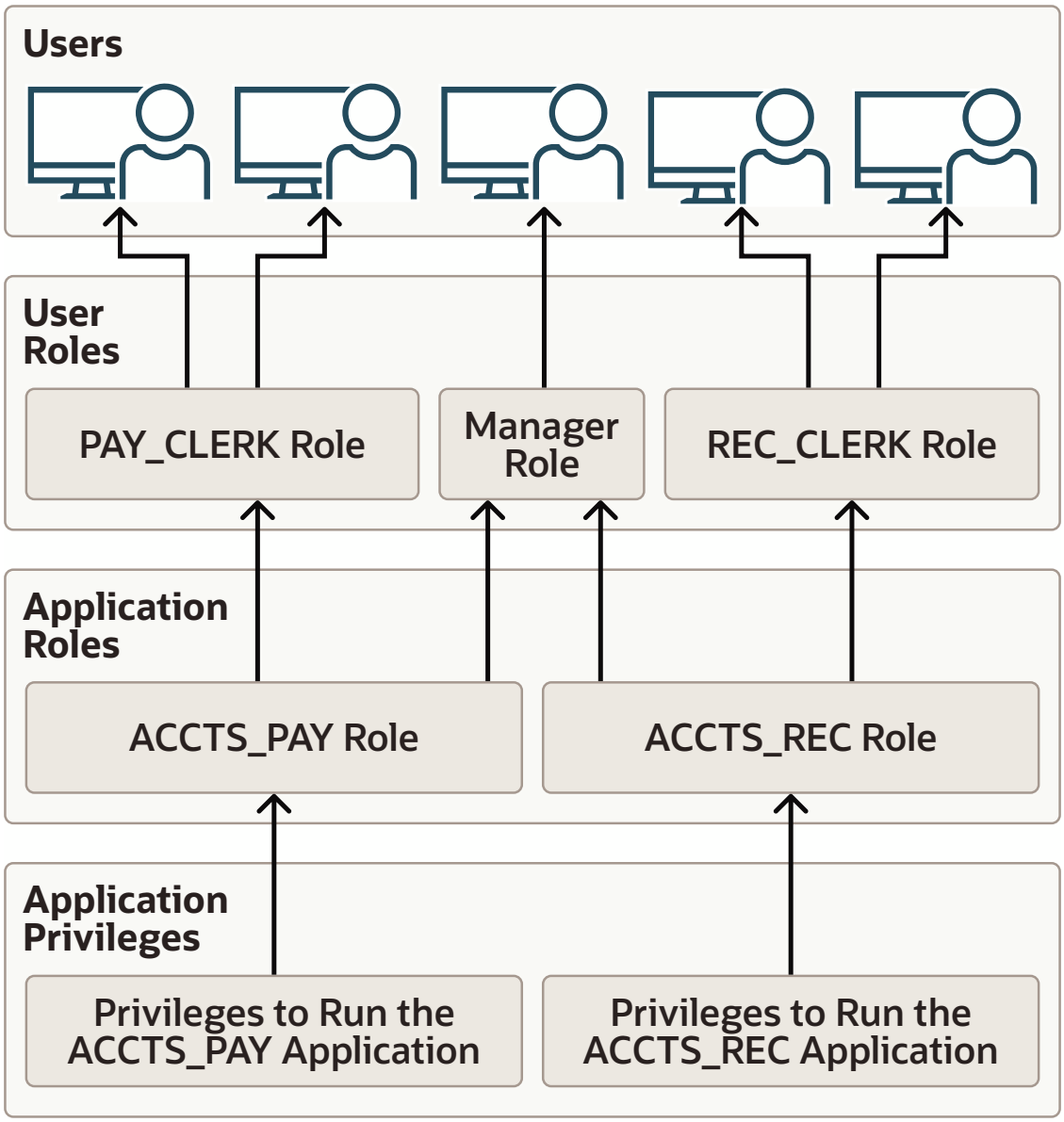
In general, you create a role to manage privileges.

Reasons are as follows:

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

The following diagram describes the two uses of roles.

Figure 4-1 Common Uses for Roles



Related Topics

- Common Uses of Application Roles
  You can use application roles to control privileges to use applications.
- Common Uses of User Roles
  You can create a user role for a group of database users with common privilege grant requirements.

4.11.1.5 Common Uses of Application Roles

You can use application roles to control privileges to use applications.

You should grant an application role all privileges necessary to run a given database application. Then, 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.

4.11.1.6 Common Uses of User Roles

You can create a user role for a group of database users with common privilege grant requirements.

You can manage user privileges by granting secure application roles and privileges to the user role and then granting the user role to appropriate users.

4.11.1.7 How Roles Affect the Scope of a User's Privileges

Each role and user has its own unique security domain.

The security domain of a role includes the privileges granted to the role plus those privileges granted to any roles that are granted to the role.

The security domain of a user 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.) This domain also includes the privileges and roles granted to the role `PUBLIC`. The `PUBLIC` role represents all users in the database.

4.11.1.8 How Roles Work in PL/SQL Blocks

Role behavior in a PL/SQL block is determined by the type of block and by definer's rights or invoker's rights.

- Roles Used in Named Blocks with Definer's Rights
  All roles are disabled in any named PL/SQL block that runs with definer's rights.
- Roles Used in Named Blocks with Invoker's Rights and Anonymous PL/SQL Blocks
  Named PL/SQL blocks that run with invoker's rights and anonymous PL/SQL blocks are run based on privileges granted through enabled roles.

4.11.1.8.1 Roles Used in Named Blocks with Definer's Rights

All roles are disabled in any named PL/SQL block that runs with definer's rights.

Examples of named PL/SQL blocks are stored procedures, functions, and triggers.

Roles are not used for privilege checking and you cannot set roles within a definer's rights procedure.

The `SESSION_ROLES` data dictionary view shows all roles that are currently enabled and if a PL/SQL block runs with definer’s rights. If a named PL/SQL block that runs with definer's rights queries `SESSION_ROLES`, then the query does not return any rows.

4.11.1.8.2 Roles Used in Named Blocks with Invoker's Rights and Anonymous PL/SQL Blocks

Named PL/SQL blocks that run with invoker's rights and anonymous PL/SQL blocks are run based on privileges granted through enabled roles.

Current roles are used for privilege checking within an invoker's rights PL/SQL block. You can use dynamic SQL to set a role in the session.

Related Topics

- *Oracle Database PL/SQL Language Reference*Invokers Rights and Definers Rights (AUTHID Property)

4.11.1.9 How Roles Aid or Restrict DDL Usage

A user requires one or more privileges to successfully run a 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 a table that belongs to another user, the creator must have 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 Database avoids the dependencies on privileges received by way of roles by restricting the use of specific privileges in certain DDL statements. The following rules describe these privilege restrictions concerning DDL statements:

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

    You cannot use the `REFERENCES` object privilege for a table to define the foreign key of a table 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. The security domain does not contain roles when a `CREATE VIEW` statement is used. For example, a user who is granted the `SELECT ANY TABLE` system privilege or the `SELECT` *`object`* privilege for a table through a role cannot use either of these privileges to create a view on a table that belongs to another user. This is because views are definer's rights objects, so when creating them you cannot use any privileges (neither system privileges or object privileges) granted to you through a role. If the privilege is granted directly to you, then you can use the privilege. However, if the privilege is revoked at a later time, then the view definition becomes invalid ("contains errors") and must recompiled before it can be used again.

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
- Directly granted a role that has 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 view on the `employees` table, because the `SELECT` *`object`* privilege for the `employees` table was granted through a role.
- 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.

4.11.1.10 How Operating Systems Can Aid Roles

In some environments, you can administer database security using the operating system.

The operating system can be used to grant and revoke database roles and to manage their password authentication. This capability is not available on all operating systems.

> **See Also:**
>
> Your operating system-specific Oracle Database documentation for details about managing roles through the operating system

4.11.1.11 How Roles Work in a Distributed Environment

In a distributed database environment, all necessary roles must be set as the default role for a distributed (remote) session.

These roles cannot be enabled when the user connects to a remote database from within a local database session. For example, the user cannot run a remote procedure that attempts to enable a role at the remote site.

> **See Also:**
>
> *Oracle Database Heterogeneous Connectivity User's Guide*

4.11.2 Predefined Roles in an Oracle Database Installation

Oracle Database provides a set of predefined roles to help in database administration.

These predefined role are automatically defined for Oracle databases when you run the standard scripts (such as `catalog.sql` and `catproc.sql`) that are part of database creation, and they are considered common roles. If you install other options or products, then other predefined roles may be created. You can find roles that are created and maintained by Oracle by querying the `ROLE` and `ORACLE_MAINTAINED` columns of the `DBA_ROLES` data dictionary view. If the output for `ORACLE_MAINTAINED` is `Y`, then you must not modify the role except by running the script that was used to create it.

Table 4-7 Oracle Database Predefined Roles

Predefined Role	Description
ACCHK_READ	Provides privileges to use Application Continuity Protection Check (ACCHK), which includes the ability to query the following data dictionary views: • DBA_ACCHK_EVENTS • DBA_ACCHK_EVENTS_SUMMARY • DBA_ACCHK_STATISTICS • DBA_ACCHK_STATISTICS_SUMMARY Database administrators and PDB administrators grant this role to developers to read their results from ACCHK.
ADM_PARALLEL_EXECUTE_TASK	Provides privileges to update table data in parallel by using the DBMS_PARALLEL_EXECUTE PL/SQL package.
AQ_ADMINISTRATOR_ROLE	Provides privileges to administer Advanced Queuing. Includes ENQUEUE ANY QUEUE, DEQUEUE ANY QUEUE, and MANAGE ANY QUEUE, SELECT privileges on Advanced Queuing tables and EXECUTE privileges on Advanced Queuing packages.
AQ_USER_ROLE	De-supported, but kept mainly for release 8.0 compatibility. Provides EXECUTE privileges on the DBMS_AQ and DBMS_AQIN packages.
AUDIT_ADMIN	Provides privileges to create unified and fine-grained audit policies, use the AUDIT and NOAUDIT SQL statements, view audit data, and manage the audit trail administration
AUDIT_VIEWER	Provides privileges to view and analyze audit data
AUTHENTICATEDUSER	Used by the XDB protocols to define any user who has logged in to the system.
AVTUNE_PKG_ROLE	Is granted by default to the DBMS_AVTUNE package so that it can do its job. The DBMS_AVTUNE package is granted the role so that is has those privileges when it executes and the user does not need to have them.
BDSQL_ADMIN	Provides privileges to use the DBMS_BDSQL PL/SQL package
BDSQL_USER	Provides privileges to use Oracle Big Data SQL
CAPTURE_ADMIN	Provides the privileges necessary to create and manage privilege analysis policies.
CDB_DBA	Provides the privileges required for administering a CDB, such as SET CONTAINER, SELECT ON PDB_PLUG_IN_VIOLATIONS, and SELECT ON CDB_LOCAL_ADMIN_PRIVS. If your site requires additional privileges, then you can create a role (either common or local) to cover these privileges, and then grant this role to the CDB_DBA role.
CONNECT	Provides the CREATE SESSION system privilege. This role is provided for compatibility with previous releases of Oracle Database. You can determine the privileges encompassed by this role by querying the DBA_SYS_PRIVS data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database.
CTXAPP	Provides privileges to create Oracle Text indexes and index preferences, and to use PL/SQL packages. This role should be granted to Oracle Text users.
DATAPUMP_EXP_FULL_DATABASE	Provides privileges to export data from an Oracle database using Oracle Data Pump. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users.

Table 4-7 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
DATAPUMP_IMP_FULL_DATABASE	Provides privileges to import data into an Oracle database using Oracle Data Pump. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users.
DB_DEVELOPER_ROLE	Provides most of the system privileges, object privileges, predefined roles, PL/SQL package privileges, and tracing privileges that an application developer needs.
DBA	Provides a large number of system privileges, including the ANY privileges (such as the DELETE ANY TABLE and GRANT ANY PRIVILEGE privileges). This role is provided for compatibility with previous releases of Oracle Database. You can find the privileges that are encompassed by this role by querying the DBA_SYS_PRIVS data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database.
DBJAVASCRIPT	Provided privileges for a schema to run JavaScript code, using the Nashorn engine of 12.2 Oracle JVM. Desupported.
DBMS_MDX_INTERNAL	Supports the DBMS_MDX_ODBO PL/SQL package. For internal use only.
DGPDB_ROLE	Grants privileges to the Oracle Data Guard account DGPDB_INT, which is an internal account
DV_ACCTMGR	Provides privileges to manage user accounts in an Oracle Database Vault environment
DV_ADMIN	Provides privileges to use the Oracle Database Vault PL/SQL packages
DV_AUDIT_CLEANUP	Provides privileges for purge operations in an Oracle Database Vault environment
DV_DATAPUMP_NETWORK_LINK	Provides privileges for performing Oracle Data Pump import operations in an Oracle Database Vault environment
DV_GOLDENGATE_ADMIN	Provides privileges to configure Oracle GoldenGate in an Oracle Database Vault environment
DV_GOLDENGATE_REDO_ACCESS	Provides privileges to use the Oracle GoldenGate TRANLOGOPTIONS DBLOGREADER method to access redo logs in an Oracle Database Vault environment
DV_MONITOR	Enables the Oracle Enterprise Manager Cloud Control agent to monitor Oracle Database Vault for attempted violations and configuration issues with realm or command rule definitions
DV_OWNER	Provides privileges to manage the Oracle Database Vault roles and its configuration
DV_PATCH_ADMIN	Provides privileges to perform patch operations in an Oracle Database Vault environment
DV_POLICY_OWNER	Provides privileges to manage to a limited degree Oracle Database Vault policies
DV_SECANALYST	Provides privileges to analyze Oracle Database Vault reports and monitor Oracle Database Vault
DV_STREAMS_ADMIN	Required for configuring Oracle Streams, which is deprecated, in an Oracle Database Vault environment
DV_XSTREAM_ADMIN	Required for configuring Oracle XStreams in an Oracle Database Vault environment
DBFS_ROLE	Provides access to the DBFS (the Database Filesystem) packages and objects.

Table 4-7 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
EJBCLIENT	Provides privileges to connect to EJBs from a Java stored procedure.
EXECUTE_CATALOG_ROLE	Provides EXECUTE privileges on objects in the data dictionary.
EXP_FULL_DATABASE	Provides the privileges required to perform full and incremental database exports using the Export utility (later replaced with Oracle Data Pump). It includes these privileges: SELECT ANY TABLE, BACKUP ANY TABLE, EXECUTE ANY PROCEDURE, EXECUTE ANY TYPE, ADMINISTER RESOURCE MANAGER, and INSERT, DELETE, and UPDATE on the tables SYS.INCVID, SYS.INCFIL, and SYS.INCEXP. Also includes the following roles: EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE. This role is provided for convenience in using the export and import utilities. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users.
GATHER_SYSTEM_STATISTICS	Provides privileges to update system statistics, which are collected using the DBMS_STATS.GATHER_SYSTEM_STATISTICS procedure
GDS_CATALOG_SELECT	Provides the read privilege to the Global Data Services (GDS) and sharding catalog tables that are owned by GSMADMIN_INTERNAL. This role was created primarily for Oracle Enterprise Manager support of GDS and shrading, but users can use it to run their own reports using GDS metadata.
GLOBAL_AQ_USER_ROLE	Provides privileges to establish a connection to an LDAP server, for use with Oracle Database Advanced Queuing
GRAPH_ADMINISTRATOR	Provides privileges to perform operations on the graph server (PGX) using the Java API (as compared to running start and stop operations as an OS user)
GRAPH_DEVELOPER	Provides privileges to create, publish, modify, query, and view graphs using the Java API or SQLcl or the graph visualization application
GRAPH_USER	Provides privileges to query and view graphs using the Java API or SQLcl or the graph visualization application
GSMADMIN_ROLE	Should be granted to Global Data Services (GDS) and sharding administrators, so that they can administer a GDS or sharding configuration
GSMCATUSER_ROLE	Granted only the Oracle delivered account GSMCATUSER for internal use
GSMROOTUSER_ROLE	Granted only to Oracle delivered account GSMROOTUSER for internal use
GSMUSER_ROLE	Granted only to Oracle delivered account GSMUSER for internal use
GSM_POOLADMIN_ROLE	Valid for GDS only (not for sharding). Should be granted to GDS pool administrators so that they can administer their GDS pool
HS_ADMIN_EXECUTE_ROLE	Provides the EXECUTE privilege for users who want to use the Heterogeneous Services (HS) PL/SQL packages
HS_ADMIN_ROLE	Provides privileges to both use the Heterogeneous Services (HS) PL/SQL packages and query the HS-related data dictionary views
HS_ADMIN_SELECT_ROLE	Provides privileges to query the Heterogeneous Services data dictionary views
IMP_FULL_DATABASE	Provides the privileges required to perform full database imports using the Import utility (later replaced with Oracle Data Pump). Includes an extensive list of system privileges (use view DBA_SYS_PRIVS to view privileges) and the following roles: EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE. This role is provided for convenience in using the export and import utilities. **Caution:** This is a very powerful role because it provides a user access to any data in any schema in the database. Use caution when granting this role to users.

Table 4-7 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
JAVADEBUGPRIV	Provides privileges to run the Oracle Database Java applications debugger
JAVAIDPRIV	Deprecated for this release
JAVASYSPRIV	Provides major permissions to use Java2, including updating Oracle JVM-protected packages
JAVAUSERPRIV	Provides limited permissions to use Java2
JAVA_ADMIN	Provides administrative permissions to update policy tables for Oracle Database Java applications
JMXSERVER	Provides privileges to start and maintain a JMX agent in a database session
LBAC_DBA	Provides permissions to use the SA_SYSDBA PL/SQL package
LOGSTDBY_ADMINISTRATOR	Provides administrative privileges to manage the SQL Apply (logical standby database) environment
OEM_ADVISOR	Provides privileges to create, drop, select (read), load (write), and delete a SQL tuning set through the DBMS_SQLTUNE PL/SQL package, and to access to the Advisor framework using the ADVISOR PL/SQL package
OEM_MONITOR	Provides privileges needed by the Management Agent component of Oracle Enterprise Manager to monitor and manage the database
OGG_APPLY	Provides privileges to manage Oracle GoldenGate Replicat
OGG_APPLY_PROCREP	Provides privileges for using Oracle GoldenGate procedural replication
OGG_CAPTURE	Provides privileges to use Oracle GoldenGate Extract
OGG_CAPTURE_SHARED	Provides privileges for managing GoldenGate Shared Capture
OLAP_DBA	Provides administrative privileges to create dimensional objects in different schemas for Oracle OLAP
OLAP_USER	Provides application developers privileges to create dimensional objects in their own schemas for Oracle OLAP
OLAP_XS_ADMIN	Provides privileges to administer security for Oracle OLAP
OPTIMIZER_PROCESSING_RATE	Provides privileges to run the GATHER_PROCESSING_RATE, SET_PROCESSING_RATE, and DELETE_PROCESSING_RATE procedures in the DBMS_STATS package. These procedures manage the processing rate of a system for automatic degree of parallelism (Auto DOP). Auto DOP uses these processing rates to determine the optimal degree of parallelism for a SQL statement.
OSAK_ADMIN_ROLE	Provides privileges for an Oracle SQL Access to Kafka (OSAK) administrator to configure, register, and manage Kafka clusters
PDB_DBA	Granted automatically to the local user that is created when you create a new PDB from the seed PDB. No privileges are provided with this role.
PGX_SERVER_GET_INFO	Provides privileges to find status information on the property graph (PGX) instance using the Admin API
PGX_SERVER_MANAGE	Provides privileges to manage the PGX instance
PGX_SESSION_ADD_PUBLISHED_GRAPH	Provides privileges to create a new graph in PGX by loading from the database using a configuration file, using the CREATE PROPERTY GRAPH statement in PGQL, creating a sub-graph from another graph, or using the GraphBuilder
PGX_SESSION_COMPILE_ALGORITHM	Provides privileges to compile algorithms using the PGX Algorithm API
PGX_SESSION_CREATE	Provides privileges to create a new PGX session using the ServerInstance.createSession API

Table 4-7 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
PGX_SESSION_GET_PUBLISHED_GRAPH	Provides privileges to query and view graphs published by another user to the public namespace
PGX_SESSION_MODIFY_MODEL	Provides privileges to create, train, and store an ML model using PgxML
PGX_SESSION_NEW_GRAPH	Provides privileges to create a new graph in PGX by loading from the database using a configuration file, using the CREATE PROPERTY GRAPH statement in PGQL, creating a sub-graph from another graph, or using the GraphBuilder
PGX_SESSION_READ_MODEL	Provides privileges to load and use an ML model using PgxML
PPLB_ROLE	Granted only to the Oracle Data Guard account DGPDB_INT for internal use. This role enables the DGPDB_INT account to access the pre-plugin backup tables when plugging new PDBs. Do not grant this role to any users or other roles.
PROVISIONER	Provides privileges to register and update global callbacks for Oracle Database Real Application sessions and to provision principals.
RDFCTX_ADMIN	Provides privileges for using the Semantic (Text) search feature of Resource Description Framework (RDF) graphs
RECOVERY_CATALOG_OWNER	Provides the following privileges for owner of the recovery catalog: • ADMINISTER DATABASE • ALTER SESSION • CREATE ANY CONTEXT • CREATE ANY SYNONYM • CREATE ANY TRIGGER • CREATE CLUSTER • CREATE DATABASE LINK • CREATE PROCEDURE • CREATE SEQUENCE • CREATE SESSION • CREATE SYNONYM • CREATE TABLE • CREATE TRIGGER • CREATE VIEW • DROP ANY SYNONYM • EXECUTE ON DBMS_RLS • QUERY REWRITE
RECOVERY_CATALOG_OWNER_VPD	Provides privileges for recovery catalog management.
RECOVERY_CATALOG_USER	Provides privileges for recovery catalog management.

Table 4-7 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
RESOURCE	Provides the following resource-related system privileges: • CREATE ANALYTIC VIEW • CREATE ATTRIBUTE DIMENSION • CREATE CLUSTER • CREATE HIERARCHY • CREATE INDEXTYPE • CREATE MATERIALIZED VIEW • CREATE OPERATOR • CREATE PROCEDURE • CREATE PROPERTY GRAPH • CREATE SEQUENCE • CREATE SYNONYM • CREATE TABLE • CREATE TRIGGER • CREATE TYPE • CREATE VIEW Be aware that RESOURCE no longer provides the UNLIMITED TABLESPACE system privilege. This role is provided for compatibility with previous releases of Oracle Database. You can determine the privileges encompassed by this role by querying the DBA_SYS_PRIVS data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database.
SAGA_ADM_ROLE	Provides the ability to invoke APIs from the DBMS_SAGA_ADM package. This role is required for saga administrators for the initial setup and provides full access to the DBMS_SAGA_ADM API.
SAGA_CONNECT_ROLE	Provided to the remote database link user when the Oracle saga framework is in use.
SAGA_PARTICIPANT_ROLE	Required for saga participant services. Saga primitives can only be invoked by a user that has the SAGA_PARTICIPANT role granted to it.
SCHEDULER_ADMIN	Allows the grantee to run the procedures of the DBMS_SCHEDULER package. It includes all of the job scheduler system privileges and is included in the DBA role.
SELECT_CATALOG_ROLE	Provides SELECT privilege on objects in the data dictionary.
SHARDED_SCHEMA_OWNER	Provides privileges for sharded schema owners to perform sharding administrative tasks on their own schema
SODA_APP	Provides privileges to use the SODA APIs, in particular, to create, drop, and list document collections.
SQL_FIREWALL_ADMIN	Provides the following privileges to administer SQL Firewall: • ADMINISTER SQL FIREWALL system privilege • EXECUTE privilege on the DBMS_SQL_FIREWALL PL/SQL package • SELECT privilege for the DBA_SQL_FIREWALL_* data dictionary views
SQL_FIREWALL_VIEWER	Provides the SELECT privilege for the SQL Firewall DBA_SQL_FIREWALL_* data dictionary views

Table 4-7 (Cont.) Oracle Database Predefined Roles

Predefined Role	Description
WM_ADMIN_ROLE	Provides administrative privileges for Oracle Workspace Manager. This enables users to run any DBMS_WM procedures on all version enabled tables, workspaces, and savepoints regardless of their owner. It also enables the user to modify the system parameters specific to Workspace Manager.
XDBADMIN	Allows the grantee to register an XML schema globally, as opposed to registering it for use or access only by its owner. It also lets the grantee bypass access control list (ACL) checks when accessing Oracle XML DB Repository (deprecated).
XDB_SET_INVOKER	Allows the grantee to define invoker's rights handlers and to create or update the resource configuration for XML repository triggers. By default, Oracle Database grants this role to the DBA role but not to the XDBADMIN role.
XDB_WEBSERVICES	Allows the grantee to access Oracle Database Web services over HTTPS. However, it does not provide the user access to objects in the database that are public. To allow public access, you need to grant the user the XDB_WEBSERVICES_WITH_PUBLIC role. For a user to use these Web services, SYS must enable the Web service servlets.
XDB_WEBSERVICES_OVER_HTTP	Allows the grantee to access Oracle Database Web services over HTTP. However, it does not provide the user access to objects in the database that are public. To allow public access, you need to grant the user the XDB_WEBSERVICES_WITH_PUBLIC role.
XDB_WEBSERVICES_WITH_PUBLIC	Allows the grantee access to public objects through Oracle Database Web services.
XSTREAM_APPLY	Provides privileges to manage XStream In
XSTREAM_CAPTURE	Provides privileges to manage XStream Out
XS_CACHE_ADMIN	In Oracle Database Real Application Security, enables the grantee to manage the mid-tier cache. It is required for caching the security policy at the mid-tier level for the checkAcl (authorization) method of the XSAccessController class. Grant this role to the application connection user or the Real Application Security dispatcher.
XS_NAMESPACE_ADMIN	In Oracle Database Real Application Security, enables the grantee to manage and manipulate the namespace and attribute for a session. Grant this role to the Real Application Security session user.
XS_RESOURCE	In Oracle Database Real Application Security, enables the grantee to manage objects in the attached schema, through the XS_ACL PL/SQL package. This package creates procedures to create and manage access control lists (ACLs). It contains the ADMIN SEC POLICY privilege. It is similar to the Oracle Database RESOURCE role.
XS_SESSION_ADMIN	In Oracle Database Real Application Security, enables the grantee to manage the life cycle of a session, including the ability to create, attach, detach, and destroy the session. Grant this role to the application connection user or Real Application Security dispatcher.

> **Note:**
>
> Each installation should create its own roles and assign only those privileges that are needed, thus retaining detailed control of the privileges in use. This process also removes any need to adjust existing roles, privileges, or procedures whenever Oracle Database changes or removes roles that Oracle Database defines. For example, the `CONNECT` role now has only one privilege: `CREATE SESSION`.

4.11.3 Creating a Role

You can create a role that is authenticated with or without a password. You also can create external or global roles.

- About the Creation of Roles
  You can create a role by using the `CREATE ROLE` statement.
- Creating a Role That Is Authenticated With a Password
  You can create a password authenticated role by using the `IDENTIFIED BY` clause.
- Creating a Role That Has No Password Authentication
  You can create a role that does not require a password by omitting the `IDENTIFIED BY` clause.
- Creating a Role That Is External or Global
  External or global roles allow services that are outside the database to associate database roles to authenticated users.
- Altering a Role
  The `ALTER ROLE` statement can modify the authorization method for a role.

4.11.3.1 About the Creation of Roles

You can create a role by using the `CREATE ROLE` statement.

To create the role, you must have the `CREATE ROLE` system privilege. Typically, only security administrators have this system privilege. After you create a role, the role has no privileges associated with it. Your next step is to grant either privileges or other roles to the new role.

You must give each role that you create a unique name among existing user names and role names of the database. Roles are not contained in the schema of any user. In a database that uses a multi-byte character set, Oracle recommends that each role name contain at least one single-byte character. If a role name contains only multi-byte characters, then the encrypted role name and password combination is considerably less secure. See Guideline 1 in Guidelines for Securing Passwords for password guidelines.

You can use the `IDENTIFIED BY` clause to authorize the role with a password. This clause specifies how the user must be authorized before the role can be enabled for use by a specific user to which it has been granted. If you do not specify this clause, or if you specify `NOT IDENTIFIED`, then no authorization is required when the role is enabled. Roles can be specified to be authorized by the following:

- The database using a password
- An application using a specified package
- Externally by the operating system, network, or other external source
- Globally by an enterprise directory service

As an alternative to creating password-protected roles, Oracle recommends that you use secure application roles instead.

Note the following restrictions about the creation of roles:

- A role and a user cannot have the same name.
- The role name cannot start with the value of the `COMMON_USER_PREFIX` parameter (which defaults to `C##`) unless this role is a CDB common role.

Related Topics

- Role Privileges and Secure Application Roles
  A secure application role can be enabled only by an authorized PL/SQL package or procedure.
- Creating Secure Application Roles to Control Access to Applications
  A secure application role is only enabled through its associated PL/SQL package or procedure.
- Rules for Creating Common Roles
  When you create a common role, you must follow special rules.

4.11.3.2 Creating a Role That Is Authenticated With a Password

You can create a password authenticated role by using the `IDENTIFIED BY` clause.

- To create a password-authenticated role, use the `CREATE ROLE` statement with the `IDENTIFIED BY` clause.

For example:

```
CREATE ROLE clerk IDENTIFIED BY password;
```

> **Note:**
>
> - You can enable password-protected roles in a proxy session. Both secure application roles and password-protected roles provide a secure method for enabling a role in a session. Oracle recommends using secure password roles instead of password-protected roles where the password has to be maintained and transmitted over insecure channels or if more than one person needs to know the password. Password-protected roles in a proxy session are suitable for situations where automation is used to set the role.
> - If you set the `SQLNET.ALLOWED_LOGON_VERSION_SERVER` parameter is to `11` or higher, then you must recreate roles that have been created with the `IDENTIFIED BY` clause.

Related Topics

- Role Privileges and Secure Application Roles
  A secure application role can be enabled only by an authorized PL/SQL package or procedure.
- Management of Case Sensitivity for Secure Role Passwords
  Oracle Database ensures that the passwords for secure roles are case sensitive.

4.11.3.3 Creating a Role That Has No Password Authentication

You can create a role that does not require a password by omitting the `IDENTIFIED BY` clause.

- Use the `CREATE ROLE` statement with no clauses to create a role that has no password authentication.

For example:

```
CREATE ROLE salesclerk;
```

4.11.3.4 Creating a Role That Is External or Global

External or global roles allow services that are outside the database to associate database roles to authenticated users.

Database external roles are associated with operating system and RADIUS groups. This way, database user authorization can be managed externally from the database.

An external user must be authorized by an external service, such as an operating system or a third-party service, before the external user can enable the role.

Global roles are used by globally authenticated users, using centrally managed users or Oracle Enterprise User Security. A global user must be authorized to use the role by the enterprise directory service before the role is enabled at login time.

Oracle recommends that you migrate to using Centrally Managed Users (CMU). This feature enables you to directly connect with Microsoft Active Directory without an intervening directory service for enterprise user authentication and authorization to the database. If your Oracle Database is in the cloud, you can also choose to move to one of the newer integrations with a cloud identity provider.

- To create a role that is to be authorized externally, include the `IDENTIFIED EXTERNALLY` clause in the `CREATE ROLE` statement.
  For example:

  ```
  CREATE ROLE clerk_external IDENTIFIED EXTERNALLY;
  ```

- To create a role to be authorized globally, use the `CREATE ROLE` statement.

  For example:

  ```
  CREATE ROLE clerk_global IDENTIFIED GLOBALLY;
  ```

You can authorize roles globally to a user through a directory service mapping such as with centrally managed users.

Related Topics

- Grants of Roles Using the Operating System or Network
  Using the operating system or network to manage roles can help centralize the role management in a large enterprise.
- Configuring RADIUS Authentication
  RADIUS is a client/server security protocol widely used to enable remote authentication and access.
- Mapping a Directory Group to a Global Role
  Database global roles mapped to directory groups give member users additional privileges and roles above what they have been granted through their login schemas.
- *Oracle Database Enterprise User Security Administrator's Guide*

4.11.3.5 Altering a Role

The `ALTER ROLE` statement can modify the authorization method for a role.

To alter the authorization method for a role, you must have the `ALTER ANY ROLE` system privilege or have been granted the role with `ADMIN` option.

Remember that you can only directly grant secure application roles or password-authenticated roles to a user. Be aware that if you create a common role in the root, you cannot change it to a local role.

- To alter a role, use the `ALTER ROLE` statement.

  For example, to alter the `clerk` role to specify that the user must be authorized by an external source before enabling the role:

  ```
  ALTER ROLE clerk IDENTIFIED EXTERNALLY;
  ```

4.11.4 Specifying the Type of Role Authorization

You can configure a role to be authorized through different sources, such the database or an external source.

- Authorizing a Role by Using the Database
  You can protect a role authorized by the database by assigning the role a password.
- Authorizing a Role by Using an Application
  An application role can be enabled only by applications that use an authorized PL/SQL package.
- Authorizing a Role by Using an External Source
  Oracle Database supports the use of external roles but with certain limitations.
- Authorizing a Role by Using the Operating System
  Oracle Database supports role authentication through the operating system but with certain limitations.
- Authorizing a Role by Using a Network Client
  Oracle Database supports role authentication by a network client but you must be aware of security risks.
- Authorizing a Global Role by an Enterprise Directory Service
  A global role enables a global user to be authorized only by an enterprise directory service.

4.11.4.1 Authorizing a Role by Using the Database

You can protect a role authorized by the database by assigning the role a password.

If you are granted a role protected by a password, then you can enable or disable the role by supplying the proper password for the role in the `SET ROLE` statement. You cannot authenticate a password-authenticated role on logon, even if the role is a member of your list of default roles. You must explicitly enable it with the `SET ROLE` statement using the required password.

1. Use the `CREATE ROLE` statement with the `IDENTIFIED BY` clause to create the password-authenticated role.

   For example:

   ```
   CREATE ROLE hr_clerk IDENTIFIED BY password;
   ```

When the role is enabled, the password must be supplied.

2. Use the `SET ROLE` statement to set the password-authenticated role.

   The following example shows how to set a password-authenticated role by using the `SET ROLE` statement.

   ```
   SET ROLE hr_clerk IDENTIFIED BY password;
   ```

Related Topics

- Guidelines for Securing Passwords
  Oracle provides guidelines for securing passwords in a variety of situations.

4.11.4.2 Authorizing a Role by Using an Application

An application role can be enabled only by applications that use an authorized PL/SQL package.

Application developers do not need to secure a role by embedding passwords inside applications. Instead, they can create an application role (secure application role) and specify which PL/SQL package is authorized to enable the role.

- To create a role enabled by an authorized PL/SQL package, use the `IDENTIFIED USING package_name` clause in the `CREATE ROLE` SQL statement.

For example, to indicate that the role `admin_role` is an application role and the role can only be enabled by any module defined inside the PL/SQL package `hr.admin`:

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

Related Topics

- Role Privileges and Secure Application Roles
  A secure application role can be enabled only by an authorized PL/SQL package or procedure.
- Creating Secure Application Roles to Control Access to Applications
  A secure application role is only enabled through its associated PL/SQL package or procedure.

4.11.4.3 Authorizing a Role by Using an External Source

Oracle Database supports the use of external roles but with certain limitations.

You can define an external role locally in the database, but you cannot grant the external role to global users, to global roles, or to any other roles in the database. You can create roles that are authorized by the operating system or network clients.

- To authorize a role by using an external source, use the `CREATE ROLE` statement with the `IDENTIFIED EXTERNALLY` clause.

For example:

```
CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;
```

4.11.4.4 Authorizing a Role by Using the Operating System

Oracle Database supports role authentication through the operating system but with certain limitations.

Role authentication through the operating system is useful only when the operating system is able to dynamically link operating system privileges with applications.

When a user starts an application, the operating system grants an operating system privilege to the user. The granted operating system privilege corresponds to the role associated with the application. At this point, the application can enable the application role. When the application is terminated, the previously granted operating system privilege is revoked from the operating system account of the user.

- If a role is authorized by the operating system, then configure information for each user at the operating system level. This operation is operating system dependent.

If roles are granted by the operating system, then you do not need to have the operating system authorize them also.

Related Topics

- Grants of Roles Using the Operating System or Network
  Using the operating system or network to manage roles can help centralize the role management in a large enterprise.

4.11.4.5 Authorizing a Role by Using a Network Client

Oracle Database supports role authentication by a network client but you must be aware of security risks.

If users connect to the database over Oracle Net, then by default, the operating system cannot authenticate their roles. This includes connections through a shared server configuration, as this connection requires Oracle Net. This restriction is the default because a remote user could impersonate another operating system user over a network connection. Oracle recommends that you set `REMOTE_OS_ROLES` to `FALSE`, which is the default.

- If you are not concerned with this security risk and want to use operating system role authentication for network clients, then set the initialization parameter `REMOTE_OS_ROLES` in the database initialization parameter file to `TRUE`.

The `REMOTE_OS_ROLES` initialization parameter is deprecated in Oracle Database 23ai

The change takes effect the next time you start the instance and mount the database.

4.11.4.6 Authorizing a Global Role by an Enterprise Directory Service

A global role enables a global user to be authorized only by an enterprise directory service.

You define the global role locally in the database by granting privileges and roles to it, but you cannot grant the global role itself to any user or other role in the database. When a global user attempts to connect to the database, the enterprise directory is queried to obtain any global roles associated with the user. Global roles are one component of enterprise user security. A global role only applies to one database, but you can grant it to an enterprise role defined in the enterprise directory. An enterprise role is a directory structure that contains global roles on multiple databases and can be granted to enterprise users.

> **Note:**
>
> Enterprise User Security (EUS) is deprecated with Oracle Database 23ai. Oracle recommends that you migrate to using Centrally Managed Users (CMU). This feature enables you to directly connect with Microsoft Active Directory without an intervening directory service for enterprise user authentication and authorization to the database. If your Oracle Database is in the cloud, you can also choose to move to one of the newer integrations with a cloud identity provider.

- To create a global role to be authorized by an enterprise directory service, use the `CREATE ROLE` statement with the `IDENTIFIED GLOBALLY` clause.

For example:

```
CREATE ROLE supervisor IDENTIFIED GLOBALLY;
```

Related Topics

- *Oracle Database Enterprise User Security Administrator's Guide*

4.11.5 Granting and Revoking Roles

You can grant or revoke privileges to and from roles, and then grant these roles to users or to other roles.

- About Granting and Revoking Roles
  You can grant system or object privileges to a role, and grant any role to any database user or to another role.
- Who Can Grant or Revoke Roles?
  The `GRANT ANY ROLE` system privilege enables users to grant or revoke any role except global roles to or from other users or roles.
- Granting and Revoking Roles to and from Program Units
  You can grant roles to function, procedure, and PL/SQL package program units.

4.11.5.1 About Granting and Revoking Roles

You can grant system or object privileges to a role, and grant any role to any database user or to another role.

However, a role cannot be granted to itself, nor can the role be granted circularly, that is, role `X` cannot be granted to role `Y` if role `Y` has previously been granted to role `X`.

To provide selective availability of privileges, Oracle Database permits applications and users to enable and disable roles. Each role granted to a user is, at any given time, either enabled or disabled. The security domain of a user includes the privileges of all roles currently enabled for the user and excludes the privileges of any roles currently disabled for the user.

A role granted to a role is called an indirectly granted role. You can explicitly enable or disable it for a user. However, whenever you enable a role that contains other roles, you implicitly enable all indirectly granted roles of the directly granted role.

You grant roles by using the `GRANT` statement, and revoke them by using the `REVOKE` statement. Privileges are granted to and revoked from roles using the same statements.

You cannot grant a secure role (that is, an `IDENTIFIED BY` role, `IDENTIFIED USING` role, or `IDENTIFIED EXTERNALLY` role) to either another secure role or to a non-secure role. You can use the `SET ROLE` statement to enable the secure role for the session.

4.11.5.2 Who Can Grant or Revoke Roles?

The `GRANT ANY ROLE` system privilege enables users to grant or revoke any role except global roles to or from other users or roles.

A global role is managed in a directory, such as Oracle Internet Directory, but its privileges are contained within a single database. By default, the `SYS` or `SYSTEM` user has the `GRANT ANY ROLE` privilege. 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 to be granted on a selective basis.

Related Topics

- *Oracle Database Enterprise User Security Administrator's Guide*

4.11.5.3 Granting and Revoking Roles to and from Program Units

You can grant roles to function, procedure, and PL/SQL package program units.

The role then becomes enabled during the execution of the program unit, but not during the compilation of the program unit. This enables you to temporarily escalate privileges in the PL/SQL code without granting the role directly to the user. It also increases security for applications and helps to enforce the principle of least privilege.

- Use the `GRANT` or `REVOKE` statement to grant or revoke a role to a program unit.

The following example shows how to grant the same role to the PL/SQL package `checkstats_pkg`:

```
GRANT clerk_admin TO package psmith.checkstats_pkg;
```

This example shows how to revoke the `clerk_admin` role from the PL/SQL package `checkstats_pkg`:

```
REVOKE clerk_admin FROM package psmith.checkstats_pkg;
```

The following example shows how to grant the role `clerk_admin` to the procedure `psmith.check_stats_proc`.

```
GRANT clerk_admin TO PROCEDURE psmith.checkstats_proc;
```

Related Topics

- Using Code Based Access Control for Definer's Rights and Invoker's Rights
  Code based access control, used to attach database roles to PL/SQL functions, procedures, or packages, works well with invoker's rights and definer's procedures.

4.11.6 Dropping Roles

Dropping a role affects the security domains of users or roles who had been granted the role.

That is, the security domains of all users and roles that were granted to the dropped role are changed to reflect the absence of the dropped role privileges.

All indirectly granted roles of the dropped role are also removed from affected security domains. Dropping a role automatically removes the role from all user default role lists.

Because the existence of objects is not dependent on the privileges received through a role, tables and other objects are not dropped when a role is dropped.

To drop a role, you must have the `DROP ANY ROLE` system privilege or have been granted the role with the `ADMIN` option.

- To drop a role, use the `DROP ROLE` statement.

For example, to drop the role `CLERK`:

```
DROP ROLE clerk;
```

4.11.7 Restricting SQL*Plus Users from Using Database Roles

You should restrict SQL*Plus users from using database roles, which helps to safeguard the database from intruder attacks.

- Potential Security Problems of Using Ad Hoc Tools
  Ad hoc tools can pose problems if malicious users have access to such tools.
- How the PRODUCT_USER_PROFILE System Table Can Limit Roles
  The `SYSTEM` schema `PRODUCT_USER_PROFILE` table can disable SQL and SQL*Plus commands in the SQL*Plus environment for each user.
- How Stored Procedures Can Encapsulate Business Logic
  Stored procedures encapsulate privileges use with business logic so that privileges are only exercised in the context of a well-formed business transaction.

4.11.7.1 Potential Security Problems of Using Ad Hoc Tools

Ad hoc tools can pose problems if malicious users have access to such tools.

Prebuilt database applications explicitly control the potential actions of a user, including the enabling and disabling of user roles while using the application. By contrast, ad hoc query tools such as SQL*Plus, permit a user to submit any SQL statement (which may or may not succeed), including enabling and disabling a granted role.

Potentially, an application user can exercise the privileges attached to that application to issue destructive SQL statements against database tables by using an ad hoc tool.

For example, consider the following scenario:

- The Vacation application has a corresponding `vacation` role.
- The `vacation` role includes the privileges to issue `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements against the `emp_tab` table.
- The Vacation application controls the use of privileges obtained through the `vacation` role.

Now, consider a user who has been granted the `vacation` role. Suppose that, instead of using the Vacation application, the user runs SQL*Plus. At this point, the user is restricted only by the privileges granted to the user explicitly or through roles, including the `vacation` role. Because SQL*Plus is an ad hoc query tool, the user is not restricted to a set of predefined actions, as with designed database applications. The user can query or modify data in the `emp_tab` table as they choose.

4.11.7.2 How the PRODUCT_USER_PROFILE System Table Can Limit Roles

The SYSTEM schema PRODUCT_USER_PROFILE table can disable SQL and SQL*Plus commands in the SQL*Plus environment for each user.

SQL*Plus, not the Oracle Database, enforces this security. You can even restrict access to the GRANT, REVOKE, and SET ROLE commands to control user ability to change their database privileges.

The PRODUCT_USER_PROFILE table enables you to list roles that you do not want users to activate with an application. You can also explicitly disable the use of various commands, such as SET ROLE.

For example, you could create an entry in the PRODUCT_USER_PROFILE table to:

- Disallow the use of the clerk and manager roles with SQL*Plus
- Disallow the use of SET ROLE with SQL*Plus

Suppose user Marla connects to the database using SQL*Plus. Marla has the clerk, manager, and analyst roles. As a result of the preceding entry in PRODUCT_USER_PROFILE, Marla is only able to exercise the analyst role with SQL*Plus. Also, when Ginny attempts to issue a SET ROLE statement, this user is explicitly prevented from doing so because of the entry in the PRODUCT_USER_PROFILE table prohibiting use of SET ROLE.

Be aware that the PRODUCT_USER_PROFILE table does not completely guarantee security, for multiple reasons. (PRODUCT_USER_PROFILE was desupported in Oracle Database release 19c.) In the preceding example, while SET ROLE is disallowed with SQL*Plus, if Marla had other privileges granted to them directly, then they could exercise these using SQL*Plus.

Related Topics

- *SQL*Plus User's Guide and Reference*

4.11.7.3 How Stored Procedures Can Encapsulate Business Logic

Stored procedures encapsulate privileges use with business logic so that privileges are only exercised in the context of a well-formed business transaction.

For example, an application developer can create a procedure to update the employee name and address in the employees table, which enforces that the data can only be updated in normal business hours.

In addition, rather than grant a human resources clerk the UPDATE privilege on the employees table, a security administrator may grant the privilege on the procedure only. Then, the human resources clerk can exercise the privilege only in the context of the procedures, and cannot update the employees table directly.

4.11.8 Role Privileges and Secure Application Roles

A secure application role can be enabled only by an authorized PL/SQL package or procedure.

The PL/SQL package itself reflects the security policies that are necessary to control access to the application.

This method of role creation restricts the enabling of this type of role to the invoking application. For example, the application can perform authentication and customized authorization, such as checking whether the user has connected through a proxy.

This type of role strengthens security because passwords are not embedded in application source code or stored in a table. This way, the actions the database performs are based on the implementation of your security policies, and these definitions are stored in one place, the database, rather than in your applications. If you need to modify the policy, you do so in one place without having to modify your applications. No matter how users connect to the database, the result is always the same, because the policy is bound to the role.

To enable the secure application role, you must run its underlying package by invoking it directly from the application when the user logs in, before the user exercises the privileges granted by the secure application role. You cannot use a logon trigger to enable a secure application role, nor can you have this type of role be a default role.

When you enable the secure application role, Oracle Database verifies that the authorized PL/SQL package is on the calling stack, that is, it verifies that the authorized PL/SQL package is issuing the command to enable the role.

You can use secure application roles to ensure the existence of a database connection. Because a secure application role is a role implemented by a package, the package can validate that users can connect to the database through a middle tier or from a specific IP address. In this way, the secure application role prevents users from accessing data outside an application. They are forced to work within the framework of the application privileges that they have been granted.

Related Topics

- Creating Secure Application Roles to Control Access to Applications
  A secure application role is only enabled through its associated PL/SQL package or procedure.

4.12 Managing Common Roles and Local Roles

A common role is a role that is created in the root; a local role is created in a PDB.

- About Common Roles and Local Roles
  Database roles can be specific to a PDB or used throughout the entire system container or application container.
- Common Roles in a CDB
  A common role exists either in the CDB root or an application root, and applies to every PDB within the root container (either the CDB or the application container).
- How Common Roles Work
  Common roles are visible in the root and in every PDB of a container within which they are defined.
- How the PUBLIC Role Works in a Multitenant Environment
  All privileges that Oracle grants to the `PUBLIC` role are granted locally.
- Privileges Required to Create, Modify, or Drop a Common Role
  Only common users who have the commonly granted `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE` privileges can create, alter, or drop common roles.
- Rules for Creating Common Roles
  When you create a common role, you must follow special rules.
- Creating a Common Role
  You can use the `CREATE ROLE` statement to create a common role.
- Rules for Creating Local Roles
  To create a local role, you must follow special rules.

- Local Roles in a CDB
  A **local role** exists only in a single PDB, and is thus completely independent of local roles in any other PDBs.
- Creating a Local Role
  You can use the `CREATE ROLE` statement to create a role.
- Role Grants and Revokes for Common Users and Local Users
  Role grants and revokes apply only to the scope of access of the common user or the local user.

4.12.1 About Common Roles and Local Roles

Database roles can be specific to a PDB or used throughout the entire system container or application container.

A common role is a role whose identity and (optional) password are created in the root of a container and will be known in the root and in all existing and future PDBs belonging to that container.

A local role exists in only one PDB and can only be used within this PDB. It does not have any commonly granted privileges.

Note the following:

- Common users can both create and grant common roles to other common and local users.
- You can grant a role (local or common) to a local user or role only locally.
- If you grant a common role locally, then the privileges of that common role apply only in the container where the role is granted.
- Local users cannot create common roles, but they can grant them to common and other local users.
- The `CONTAINER = ALL` clause is the default when you create a common role in the CDB root or an application root.
- Every Oracle-supplied role is common, for example, the predefined `DBA` role. In Oracle-supplied scripts, every privilege or role granted to Oracle-supplied users and roles is granted commonly, with one exception: system privileges are granted locally to the common role `PUBLIC`.

Related Topics

- Predefined Roles in an Oracle Database Installation
  Oracle Database provides a set of predefined roles to help in database administration.

4.12.2 Common Roles in a CDB

A common role exists either in the CDB root or an application root, and applies to every PDB within the root container (either the CDB or the application container).

Common roles are useful for cross-container operations, ensuring that a common user has a role in every PDB. Every common role is one of the following types:

- Oracle-supplied

  All Oracle-supplied roles, such as `DBA` and `PUBLIC`, are common to the CDB.
- User-created

Create a common role by executing `CREATE ROLE ... CONTAINER=ALL` in either the CDB root or application root, which determines the container to which the role is common. The standard naming conventions apply. Additionally, the names of CDB common roles must begin with the characters specified by the `COMMON_USER_PREFIX` initialization parameter, which are `c##` or `C##` by default.

The scope of the role is the scope of the root within which it is defined. If you define the role in `CDB$ROOT`, then its scope is the entire CDB. If you define the role within application root, then its scope is the application container.

4.12.3 How Common Roles Work

Common roles are visible in the root and in every PDB of a container within which they are defined.

A privilege can be granted commonly to a common role if:

- The grantor is a common user.
- The grantor possesses the commonly granted `ADMIN OPTION` for the privilege that is being granted.
- The `GRANT` statement contains the `CONTAINER=ALL` clause.

If the common role contains locally granted privileges, then these privileges apply only within the PDB in which they were granted to the common role. A local role cannot be granted commonly.

For example, suppose the CDB common user `c##hr_mgr` has been commonly granted the `DBA` role. This means that user `c##hr_mgr` can use the privileges associated with the `DBA` role in the root and in every PDB in the container. However, if the CDB common user `c##hr_mgr` has only been locally granted the `DBA` role for the `hr_pdb` PDB, then this user can only use the `DBA` role's privileges in the `hr_pdb` PDB.

4.12.4 How the PUBLIC Role Works in a Multitenant Environment

All privileges that Oracle grants to the `PUBLIC` role are granted locally.

This feature enables you to revoke privileges or roles that have been granted to the `PUBLIC` role individually in each PDB as needed. If you must grant any privileges to the `PUBLIC` role, then grant them locally. Never grant privileges to `PUBLIC` commonly.

Related Topics

- About Commonly and Locally Granted Privileges
  Both common users and local users can grant privileges to one another.

4.12.5 Privileges Required to Create, Modify, or Drop a Common Role

Only common users who have the commonly granted `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE` privileges can create, alter, or drop common roles.

Common users can also create local roles, but these roles are available only in the PDB in which they were created.

4.12.6 Rules for Creating Common Roles

When you create a common role, you must follow special rules.

The rules are as follows:

- **Ensure that you are in the correct root.** For the creation of common roles, you must be in the correct root, either the CDB root or the application root. You cannot create common roles from a PDB. To check if you are in the correct root, run one of the following:
  - To confirm that you are in the CDB root, you can issue the `show_con_name` command. The output should be `CDB$ROOT`.
  - To confirm that you are in an application root, verify that the following query returns `YES`:

    ```
    SELECT APPLICATION_ROOT FROM V$PDBS WHERE CON_ID=SYS_CONTEXT('USERENV',
    'CON_ID');
    ```

  - **Ensure that the name that you give the common role starts with the value of the COMMON_USER_PREFIX parameter (which defaults to C##).** Note that this requirement does not apply to the names of existing Oracle-supplied roles, such as `DBA` or `RESOURCE`.
- **Optionally, set the CONTAINER clause to ALL.** As long as you are in the root, if you omit the `CONTAINER = ALL` clause, then by default the role is created as a common role for the CDB root or the application root.

4.12.7 Creating a Common Role

You can use the `CREATE ROLE` statement to create a common role.

1. Connect to the root of the CDB or the application container in which you want to create the common role.

   For example:

   ```
   CONNECT SYSTEM
   Enter password: password
   Connected.
   ```

2. Run the `CREATE ROLE` statement with the `CONTAINER` clause set to `ALL`.

   For example:

   ```
   CREATE ROLE c##sec_admin IDENTIFIED BY password CONTAINER=ALL;
   ```

Related Topics

- Creating a Role
  You can create a role that is authenticated with or without a password. You also can create external or global roles.
- Creating a Common Role in Enterprise Manager
  Common roles can be used to assign common privileges to common users.

4.12.8 Rules for Creating Local Roles

To create a local role, you must follow special rules.

These rules are as follows:

- You must be connected to the PDB in which you want to create the role, and have the `CREATE ROLE` privilege.
- The name that you give the local role must not start with the value of the `COMMON_USER_PREFIX` parameter (which defaults to `C##`).
- You can include `CONTAINER=CURRENT` in the `CREATE ROLE` statement to specify the role as a local role. If you are connected to a PDB and omit this clause, then the `CONTAINER=CURRENT` clause is implied.
- You cannot have common roles and local roles with the same name. However, you can use the same name for local roles in different PDBs. To find the names of existing roles, query the `CDB_ROLES` and `DBA_ROLES` data dictionary views.

4.12.9 Local Roles in a CDB

A **local role** exists only in a single PDB, and is thus completely independent of local roles in any other PDBs.

A local role can only contain roles and privileges that apply within the container in which the role exists. For example, if you create the local role `pdbadmin` in `hrpdb`, then the scope of this role is restricted to this PDB.

PDBs in the same CDB, or in the same application container, may contain local roles with the same name. For example, the user-created role `pdbadmin` may exist in both `hrpdb` and `salespdb`. However, these roles are completely independent of each other.

4.12.10 Creating a Local Role

You can use the `CREATE ROLE` statement to create a role.

1. Connect to the PDB in which you want to create the local role.

   For example:

   ```
   CONNECT sec_admin@pdb_name
   Enter password: password
   Connected.
   ```

   To find the available PDBs in a CDB, log in to the CDB root container and then query the `PDB_NAME` column of the `DBA_PDBS` data dictionary view. To check the current container, run the `show con_name` command.

2. Run the `CREATE ROLE` statement with the `CONTAINER` clause set to `CURRENT`.

   For example:

   ```
   CREATE ROLE sec_admin CONTAINER=CURRENT;
   ```

4.12.11 Role Grants and Revokes for Common Users and Local Users

Role grants and revokes apply only to the scope of access of the common user or the local user.

Common users can grant and revoke common roles to and from other common users. A local user can grant a common role to any user in a PDB, including common users, but this grant applies only within the PDB.

The following example shows how to grant the common user `c##sec_admin` the `AUDIT_ADMIN` common role for use in all containers.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT AUDIT_ADMIN TO c##sec_admin CONTAINER=ALL;
```

Similarly, the next example shows how local user `aud_admin` can grant the common user `c##sec_admin` the `AUDIT_ADMIN` common role for use within the `hrpdb` PDB.

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.

GRANT AUDIT_ADMIN TO c##sec_admin CONTAINER=CURRENT;
```

This example shows how a local user `aud_admin` can revoke a role from another user in a PDB. If you omit the `CONTAINER` clause, then `CURRENT` is implied.

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.

REVOKE sec_admin FROM psmith CONTAINER=CURRENT;
```

Related Topics

- Revoking Common Privilege Grants in Enterprise Manager
  You can revoke common privilege grants from the root.

4.13 Restricting Operations on PDBs Using PDB Lockdown Profiles

You can use PDB lockdown profiles to restrict sets of user operations in pluggable databases (PDBs).

- About PDB Lockdown Profiles
  A PDB lockdown profile is a named set of features that controls a group of operations.
- How PDB Lockdown Profiles Work
  PDB lockdown profiles are designed to restrict access at different levels for features that use shared identities.
- PDB_OS_CREDENTIAL Initialization Parameter
  When the database accesses an external procedure with the `extproc` agent, the `PDB_OS_CREDENTIAL` initialization parameter determines the identity of the operating system user employed when interacting with the operating system from a PDB.
- Features That Benefit from PDB Lockdown Profiles
  Features that use shared identities benefit from PDB lockdown profiles.

- PDB Lockdown Profile Inheritance
  PDB lockdown profiles have inheritance behaviors between the CDB root, the application root, and their associated PDBs.
- Default PDB Lockdown Profiles
  Oracle Database provides a set of default PDB lockdown profiles that you can customize for your site requirements.
- Creating a PDB Lockdown Profile
  To create a PDB lockdown profile, you must have the `CREATE LOCKDOWN PROFILE` system privilege.
- Enabling or Disabling a PDB Lockdown Profile
  To enable or disable a PDB lockdown profile, use the `PDB_LOCKDOWN` initialization parameter
- Dropping a PDB Lockdown Profile
  To drop a PDB lockdown profile, you must have the `DROP LOCKDOWN PROFILE` system privilege and be logged into the CDB or application root.

4.13.1 About PDB Lockdown Profiles

A PDB lockdown profile is a named set of features that controls a group of operations.

A PDB lockdown profile restricts the features and options available to users in a PDB. The `PDB_OS_CREDENTIAL` initialization parameter can specify a unique operating system user for a PDB to limit operating system access. Also, when the `PATH_PREFIX` and `CREATE_FILE_DEST` clauses are specified during PDB creation, they limit file system access.

In some cases, you can enable or disable operations individually. For example, a PDB lockdown profile can contain settings to disable specific clauses that come with the `ALTER SYSTEM` statement.

PDB lockdown profiles restrict user access to the functionality the features provided, similar to resource limits that are defined for users. As the name suggests, you use PDB lockdown profiles in a CDB, for an application container, or for a PDB or application PDB. You can create custom profiles to accommodate the requirements of your site. PDB profiles enable you to define custom security policies for an application. In addition, you can create a lockdown profile that is based on another profile, called a **base profile**. You can configure this profile to be dynamically updated when the base profile is modified, or configure it to be static (unchanging) when the base profile is updated. Lockdown profiles are designed for both Oracle Cloud and on-premises environments.

The general procedure for creating a PDB lockdown profile is to first create it in the CDB root or the application root using the `CREATE LOCKDOWN PROFILE` statement, and then use the `ALTER LOCKDOWN PROFILE` statement to add the restrictions.

To enable a PDB lockdown profile, you can use the `ALTER SYSTEM` statement to set the `PDB_LOCKDOWN` parameter. You can find information about existing PDB lockdown profiles by connecting to CDB or application root and querying the `DBA_LOCKDOWN_PROFILES` data dictionary view. A local user can find the contents of a PDB lockdown parameter by querying the `V$LOCKDOWN_RULES` dynamic data dictionary view.

4.13.2 How PDB Lockdown Profiles Work

PDB lockdown profiles are designed to restrict access at different levels for features that use shared identities.

A use case for might be the creation of lockdown profiles at high, medium, and low levels. The high level might greatly restrict access, whereas the low level might enable access.

When logged in to the CDB root or application root, create a lockdown profile by issuing the `CREATE LOCKDOWN PROFILE` statement, which supports the following optional clauses:

- `FROM` *`static_base_profile`* creates a new lockdown profile by using the values from an existing profile. Any subsequent changes to the existing profile will not affect the new profile.
- `INCLUDING` *`dynamic_base_profile`* creates a new lockdown profile by using the values from an existing profile, except that this new lockdown profile inherits the `DISABLE STATEMENT` rules that comprise the base profile, and any subsequent changes to the base profile.

The user issuing the statement must have the `CREATE LOCKDOWN PROFILE` system privilege in the current container. You can add and remove restrictions with the `ALTER LOCKDOWN PROFILE` statement. The user must issue the `ALTER` statement in the CDB root or application root and must have the have `ALTER LOCKDOWN PROFILE` system privilege in the current container.

Specify a lockdown profile by using the `PDB_LOCKDOWN` initialization parameter. This parameter determines whether the PDB lockdown profile applies to a given PDB. You can set this parameter at the following levels:

- PDB

  The profile applies only to the PDB in which it is set.
- Application container

  The profile applies to all application PDBs in the application container. The value can be modified only by an application common user who has application common `SYSDBA` or common `ALTER SYSTEM` privileges or a CDB common user who has common `SYSDBA` or common `ALTER SYSTEM` privileges.
- CDB

  The profile applies to all PDBs. A common user who has common `SYSDBA` or common `ALTER SYSTEM` privileges can override a CDB-wide setting for a specific PDB.

If the `PDB_LOCKDOWN` parameter in a PDB is set to the name of a lockdown profile different from the container for this PDB (CDB or application container), then a set of rules govern the interaction between restrictions.

Example 4-5 Creating a PDB Lockdown Profile

In this example, you connect to the CDB root as a common user with the `CREATE LOCKDOWN PROFILE` privilege. You create a profile called medium that disables all `ALTER SYSTEM` statements except for `ALTER SYSTEM FLUSH SHARED POOL`:

```
CREATE LOCKDOWN PROFILE medium;
ALTER LOCKDOWN PROFILE medium DISABLE STATEMENT=('ALTER SYSTEM');
ALTER LOCKDOWN PROFILE medium ENABLE STATEMENT=('ALTER SYSTEM')
CLAUSE=('FLUSH SHARED POOL');
```

You can connect as the same common user to each PDB that requires this profile, and then use `ALTER SYSTEM` to set the `PDB_LOCKDOWN` initialization parameter to `medium`. For example, you could set `PDB_LOCKDOWN` to `medium` for `hrpdb`, but not `salespdb`.

The following example creates a `medium2` profile from `medium`:

```
CREATE LOCKDOWN PROFILE medium2 FROM medium;
```

4.13.3 PDB_OS_CREDENTIAL Initialization Parameter

When the database accesses an external procedure with the `extproc` agent, the `PDB_OS_CREDENTIAL` initialization parameter determines the identity of the operating system user employed when interacting with the operating system from a PDB.

Using an operating system user described by a credential whose name is specified as a value of the `PDB_OS_CREDENTIAL` initialization parameter can ensure that operating system interactions are performed as a less powerful user. In this way, the feature protects data belonging to one PDB from being accessed by users connected to another PDB. A credential is an object that is created using the `CREATE_CREDENTIAL` procedure in the `DBMS_CREDENTIAL` package.

The Oracle operating system user is usually a highly privileged user. Using this account for operating system interactions is not recommended. Also, using the same OS user for operating system interactions from different PDBs might compromise data belonging to a given PDB.

4.13.4 Features That Benefit from PDB Lockdown Profiles

Features that use shared identities benefit from PDB lockdown profiles.

A potential for elevation of privileges exists when PDBs share an identity. For example, identity can be shared at a network level, or when PDBs access common objects or connect through database links. To increase security, a CDB administrator may want to compartmentalize access, thereby restricting the operations that a user can perform in a PDB.

When identities are shared between PDBs, elevated privileges may exist. You can use lockdown profiles to prevent this elevation of privileges. Identities can be shared in the following situations:

- At the operating system level, when the database interacts with operating system resources such as files or processes
- At the network level, when the database communicates with other systems, and network identity is important
- Inside the database, as PDBs access or create common objects or they communicate across container boundaries using features such as database links

The features that use shared identifies and that benefit from PDB lockdown profiles are in several categories.

- **Network access features.** These are operations that use the network to communicate outside the PDB. For example, the PL/SQL packages `UTL_TCP`, `UTL_HTTP`, `UTL_MAIL`, `UTL_SNMP`, `UTL_INADDR`, and `DBMS_DEBUG_JDWP` perform these kinds of operations. Currently, ACLs are used to control this kind of access to share network identity.
- **Common user or object access.** These are operations in which a local user in the PDB can proxy through common user accounts or access objects in a common schema. These kinds of operations include adding or replacing objects in a common schema, granting privileges to common objects, accessing common directory objects, granting the `INHERIT PRIVILEGES` role to a common user, and manipulating a user proxy to a common user.

- **Operating System access.** For example, you can restrict access to the `UTL_FILE` or `DBMS_FILE_TRANSFER` PL/SQL packages.
- **Connections.** For example, you can restrict common users from connecting to the PDB or you can restrict a local user who has the `SYSOPER` administrative privilege from connecting to a PDB that is open in restricted mode.
- **Administrative features.** For example, you can restrict the use of `ALTER SYSTEM`, `ALTER SESSION`, and `ALTER DATABASE`.
- **Database options.** For example, you can use lockdown profiles to disable access to database options such as Oracle Partitioning or Oracle Database Advanced Queuing.

4.13.5 PDB Lockdown Profile Inheritance

PDB lockdown profiles have inheritance behaviors between the CDB root, the application root, and their associated PDBs.

- The inheritance path between PDBs and their respective roots is as follows:
  - The `PDB_LOCKDOWN` parameter setting in a CDB PDB takes precedence over the `PDB_LOCKDOWN` parameter setting in the CDB root. Similarly, the `PDB_LOCKDOWN` setting in an application PDB takes precedence over a `PDB_LOCKDOWN` setting in the application root.
  - If a CDB PDB (or an application PDB) does not have the `PDB_LOCKDOWN` parameter set, then the PDB inherits the settings of the `PDB_LOCKDOWN` parameter in the CDB root (or the application root).
  - If the application root does not have the `PDB_LOCKDOWN` parameter set, then the application root inherits the settings of the `PDB_LOCKDOWN` parameter in the CDB root.
- If the `PDB_LOCKDOWN` parameter in a CDB PDB or an application PDB is set to a CDB lockdown profile, then the PDB ignores any lockdown profiles that are set by the `PDB_LOCKDOWN` parameter in the CDB root or the application root.
- PDB lockdown parameters can inherit rules that are stipulated in an application lockdown profile, including the disable rules that come from a CDB lockdown profile that was set in its nearest ancestor (that is, an application root or the CDB root). This applies in the case of when a `PDB_LOCKDOWN` parameter in an application PDB is set to an application lockdown profile while the `PDB_LOCKDOWN` parameter in the application root or the CDB root is set to a CDB lockdown profile.
- Sometimes a conflict arises between the rules that comprise a CDB lockdown profile and an application lockdown profile. In this case, the rules in the CDB lockdown profile take precedence. For example, the setting for an `OPTION_VALUE` clause in the CDB lockdown profile takes precedence over the setting for the `OPTION_VALUE` clause in an application lockdown profile.

4.13.6 Default PDB Lockdown Profiles

Oracle Database provides a set of default PDB lockdown profiles that you can customize for your site requirements.

By default, most of these profiles are empty. They are designed to be a placeholder or template for you to configure, depending on your deployment requirements.

Detailed information about these profiles is as follows:

- `PRIVATE_DBAAS` incorporates restrictions that are suitable for private Cloud Database-as-a-Service (DBaaS) deployments. These restrictions are:
  - Must have the same database administrator for each PDB
  - Different users permitted to connect to the database
  - Different applications permitted

  `PRIVATE_DBAAS` permits users to connect to the PDBs but prevents them from using Oracle Database administrative features.
- `SAAS` incorporates restrictions that are suitable for Software-as-a-Service (SaaS) deployments. These restrictions are:
  - Must have the same database administrator for each PDB
  - Different users permitted to connect to the database
  - Must use the same application

  The `SAAS` lockdown profile is more restrictive than the `PRIVATE_DBAAS` profile. Users can be different, but the application code is the same; users are prevented from directly connecting and must connect only through the application; and users are not granted the ability to perform any administrative features.
- `PUBLIC_DBAAS` incorporates restrictions that are suitable for public Cloud Database-as-a-Service (DBaaS) deployments. The restrictions are as follows:
  - Different DBAs in each PDB
  - Different users
  - Different applications

  The `PUBLIC_DBAAS` lockdown profile is the most restrictive of the lockdown profiles.

4.13.7 Creating a PDB Lockdown Profile

To create a PDB lockdown profile, you must have the `CREATE LOCKDOWN PROFILE` system privilege.

After you create the lockdown profile, you can add restrictions before enabling it.

1. Connect to the CDB root or the application root as a user who has the `CREATE LOCKDOWN PROFILE` system privilege.

   For example, to connect to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `CREATE LOCKDOWN PROFILE` statement to create the profile by using the following syntax:

   ```
   CREATE LOCKDOWN PROFILE profile_name
   [FROM static_base_profile | INCLUDING dynamic_base_profile];
   ```

   In this specification:

   - *profile_name* is the name that you assign the lockdown profile. You can find existing names by querying the `PROFILE_NAMES` column of the `DBA_LOCKDOWN_PROFILES` data dictionary view.

- `FROM static_base_profile` creates a new lockdown profile by using the values from an existing profile. Any subsequent changes to the base profile will not affect the new profile.
- `INCLUDING dynamic_base_profile` also creates a new lockdown profile by using the values from an existing base profile, except that this new lockdown profile will inherit the `DISABLE STATEMENT` rules that comprise the base profile, as well as any subsequent changes to the base profile. If rules that are explicitly added to the new profile conflict with the rules in the base profile, then the rules in the base profile take precedence. For example, an `OPTION_VALUE` clause in the base profile takes precedence over the `OPTION_VALUE` clause in the new profile.

The following two PDB lockdown profile statements demonstrate how the inheritance works:

```
CREATE LOCKDOWN PROFILE hr_prof INCLUDING PRIVATE_DBAAS;
CREATE LOCKDOWN PROFILE hr_prof2 FROM hr_prof;
```

In the first statement, `hr_prof` inherits any changes made to the `PRIVATE_DBAAS` base profile. If a new statement is enabled for `PRIVATE_DBAAS`, then it is enabled for `hr_prof`. In the second statement, in contrast, when `hr_prof` changes, then `hr_prof2` does *not* change because it is independent of its base profile.

3. Run the `ALTER LOCKDOWN PROFILE` statement to provide restrictions for the profile.

   For example:

```
ALTER LOCKDOWN PROFILE hr_prof DISABLE STATEMENT  = ('ALTER SYSTEM');
ALTER LOCKDOWN PROFILE hr_prof ENABLE STATEMENT = ('ALTER SYSTEM') clause
= ('flush shared_pool');
ALTER LOCKDOWN PROFILE hr_prof DISABLE FEATURE = ('XDB_PROTOCOLS');
```

   In the preceding example:

   - `DISABLE STATEMENT = ('ALTER SYSTEM')` disables the use of all `ALTER SYSTEM` statements for the PDB.
   - `ENABLE STATEMENT = ('ALTER SYSTEM') clause = ('flush shared_pool')` enables only the use of the `FLUSH_SHARED_POOL` clause for `ALTER SYSTEM`.
   - `DISABLE FEATURE = ('XDB_PROTOCOLS')` prohibits the use of the XDB protocols (FTP, HTTP, HTTPS) by this PDB

   After you create a PDB lockdown profile, you are ready to enable it by using the `ALTER SYSTEM SET PDB_LOCKDOWN` SQL statement.

4.13.8 Enabling or Disabling a PDB Lockdown Profile

To enable or disable a PDB lockdown profile, use the `PDB_LOCKDOWN` initialization parameter

You can use `ALTER SYSTEM SET PDB_LOCKDOWN` to enable a lockdown profile in any of the following contexts:

- CDB (affects all PDBs)
- Application root (affects all application PDBs in the container)
- Application PDB

- PDB

> **Note:**
>
> It is not necessary to restart the instance to enable the profile. When the `ALTER SYSTEM SET PDB_LOCKDOWN` statement completes, the profile rules take effect immediately.

When you set `PDB_LOCKDOWN` in the CDB root, every PDB and application root inherits this setting unless `PDB_LOCKDOWN` is set at the container level. To disable lockdown profiles, set `PDB_LOCKDOWN` to null. If you set this parameter to null in the CDB root, then lockdown profiles are disabled for all PDBs except those that explicitly set a profile within the PDB.

A CDB common user who has been commonly granted the `SYSDBA` administrative privilege or the `ALTER SYSTEM` system privilege can set `PDB_LOCKDOWN` only to a lockdown profile that was created in the CDB root. An application common user with the application common `SYSDBA` administrative privilege or the `ALTER SYSTEM` system privilege can set `PDB_LOCKDOWN` only to a lockdown profile created in an application root.

1. Log in to the desired container as a user who has the commonly granted `ALTER SYSTEM` or commonly granted `SYSDBA` privilege.

   For example, to enable the profile for all PDBs, log in to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `ALTER SYSTEM SET PDB_LOCKDOWN` statement.

   For example, the following statement enables the lockdown profile named `hr_prof` for all PDBs:

   ```
   ALTER SYSTEM SET PDB_LOCKDOWN = hr_prof;
   ```

   The following statement resets the `PDB_LOCKDOWN` parameter:

   ```
   ALTER SYSTEM RESET PDB_LOCKDOWN;
   ```

   This variation of the preceding statement includes the `SCOPE` clause::

   ```
   ALTER SYSTEM RESET PDB_LOCKDOWN SCOPE = BOTH;
   ```

   The following statement disables all lockdown profiles in the CDB except those that are explicitly set at the PDB level:

   ```
   ALTER SYSTEM SET PDB_LOCKDOWN = '' SCOPE = BOTH;
   ```

   To find the names of PDB lockdown profiles, query the `PROFILE_NAME` column of the `DBA_LOCKDOWN_PROFILES` data dictionary view.

3. Optionally, review information about the profiles by querying `DBA_LOCKDOWN_PROFILES`.

For example, run the following query:

```
SET LINESIZE 150
COL PROFILE_NAME FORMAT a20
COL RULE FORMAT a20
COL CLAUSE FORMAT a25

SELECT PROFILE_NAME, RULE, CLAUSE, STATUS FROM CDB_LOCKDOWN_PROFILES;
```

Sample output appears below:

```
PROFILE_NAME      RULE                 CLAUSE                    STATUS
----------------- -------------------- ------------------------- -------
HR_PROF           XDB_PROTOCOLS                                  DISABLE
HR_PROF           ALTER SYSTEM                                   DISABLE
HR_PROF           ALTER SYSTEM         FLUSH SHARED_POOL         ENABLE
HR_PROF2                                                        EMPTY
PRIVATE_DBAAS                                                   EMPTY
PUBLIC_DBAAS                                                    EMPTY
SAAS                                                            EMPTY
```

4.13.9 Dropping a PDB Lockdown Profile

To drop a PDB lockdown profile, you must have the `DROP LOCKDOWN PROFILE` system privilege and be logged into the CDB or application root.

You can find the names of existing PDB lockdown profiles by querying the `DBA_LOCKDOWN_PROFILES` data dictionary view.

1. Connect to the CDB root or the application root as a user who has the `DROP LOCKDOWN PROFILE` system privilege.

   For example, to connect to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `DROP LOCKDOWN_PROFILE` statement.

   For example:

   ```
   DROP LOCKDOWN PROFILE hr_prof2;
   ```

3. Optionally, review the current list of profiles by querying `DBA_LOCKDOWN_PROFILES`.

   For example, run the following query:

   ```
   SET LINESIZE 150
   COL PROFILE_NAME FORMAT a20
   COL RULE FORMAT a20
   COL CLAUSE FORMAT a25

   SELECT PROFILE_NAME, RULE, CLAUSE, STATUS FROM CDB_LOCKDOWN_PROFILES;
   ```

Sample output appears below:

```
PROFILE_NAME      RULE                 CLAUSE                       STATUS
----------------- -------------------- ---------------------------- -------
HR_PROF           XDB_PROTOCOLS                                     DISABLE
HR_PROF           ALTER SYSTEM                                      DISABLE
HR_PROF           ALTER SYSTEM         FLUSH SHARED_POOL            ENABLE
PRIVATE_DBAAS                                                       EMPTY
PUBLIC_DBAAS                                                        EMPTY
SAAS                                                                EMPTY
```

4.14 Managing Object Privileges

Object privileges enable you to perform actions on schema objects, such as tables or indexes.

- About Object Privileges
  An object privilege grants permission to perform a particular action on a specific schema object.
- Who Can Grant Object Privileges?
  A user automatically has all object privileges for schema objects contained in their schema.
- Grants and Revokes of Object Privileges
  You can grant privileges to or revoke privileges from objects either directly to a user or through roles.
- READ and SELECT Object Privileges
  The `READ` and `SELECT` privileges provide different layers of query privileges.
- Object Privilege Use with Synonyms
  The `CREATE SYNONYM` statement create synonyms for database objects.
- Sharing Application Common Objects
  Database objects can be configured so that their metadata links, data links, and extended data links can be shared in the application root.

4.14.1 About Object Privileges

An object privilege grants permission to perform a particular action on a specific schema object.

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

Some examples of object privileges include the right to:

- Use an edition
- Update a table
- Select rows from another user's table
- Run a stored procedure of another user

If you want to restrict privilege grants to all objects within a specific schema, then you can do so by granting the user or role a schema privilege for the schema. A schema privilege enables

you to perform one grant that will apply to all objects of a specific type within the schema. For example, a grant of the `CREATE ANY TABLE` privilege for the schema enables the user to create any tables within that schema.

Related Topics

- How Commonly Granted Object Privileges Work
  Object privileges on common objects applies to the object as well as all associated links on this common object.
- Managing Schema Privileges
  Schema privileges enable certain system privileges to be granted on a schema.
- *Oracle Database SQL Language Reference*

4.14.2 Who Can Grant Object Privileges?

A user automatically has all object privileges for schema objects contained in their schema.

A user with the `GRANT ANY OBJECT PRIVILEGE` system privilege can grant any specified object privilege to another user with or without the `WITH GRANT OPTION` clause of the `GRANT` statement. A user with the `GRANT ANY OBJECT PRIVILEGE` privilege can also use that privilege to revoke any object privilege that was granted either by the object owner or by some other user with the `GRANT ANY OBJECT PRIVILEGE` privilege.

If the grantee does not have the `GRANT ANY OBJECT PRIVILEGE` privilege or had been granted the privilege without the `WITH GRANT OPTION` clause of the `GRANT` statement, then this user cannot grant the privilege to other users.

The `WITH GRANT OPTION` can be used only with object privilege grants to users. It cannot be used for object privilege grants to roles.

Related Topics

- *Oracle Database SQL Language Reference*

4.14.3 Grants and Revokes of Object Privileges

You can grant privileges to or revoke privileges from objects either directly to a user or through roles.

- About Granting and Revoking Object Privileges
  Object privileges can be granted to and revoked from users and roles.
- How the ALL Clause Grants or Revokes All Available Object Privileges
  Each type of object has different privileges associated with it, which can be controlled by the `ALL` clause.

4.14.3.1 About Granting and Revoking Object Privileges

Object privileges can be granted to and revoked from users and roles.

If you grant object privileges to roles, then you can make the privileges selectively available To grant object privileges, you can use the `GRANT` statement; to revoke object privileges, you can use the `REVOKE` statement.

4.14.3.2 How the ALL Clause Grants or Revokes All Available Object Privileges

Each type of object has different privileges associated with it, which can be controlled by the `ALL` clause.

You can specify `ALL [PRIVILEGES]` to grant or revoke all available object privileges for an object. `ALL` is not a privilege. Rather, it is a shortcut, or a way of granting or revoking all object privileges with one `GRANT` and `REVOKE` statement. If all object privileges are granted using the `ALL` shortcut, then individual privileges can still be revoked.

Similarly, you can revoke all individually granted privileges by specifying `ALL`. However, if you `REVOKE ALL`, and revoking causes integrity constraints to be deleted (because they depend on a `REFERENCES` privilege that you are revoking), then you must include the `CASCADE CONSTRAINTS` option in the `REVOKE` statement.

Example 4-6 revokes all privileges on the orders table in the `HR` schema using `CASCADE CONSTRAINTS`.

Example 4-6 Revoking All Object Privileges Using CASCADE CONSTRAINTS

```
REVOKE ALL
 ON ORDERS FROM HR
 CASCADE CONSTRAINTS;
```

4.14.4 READ and SELECT Object Privileges

The `READ` and `SELECT` privileges provide different layers of query privileges.

- About Managing READ and SELECT Object Privileges
  You can grant users either the `READ` or the `SELECT` object privilege.
- Enabling Users to Use the READ Object Privilege to Query Any Table in the Database
  The `READ ANY TABLE` system privilege provides the `READ` object privilege for querying any table in the database.
- Restrictions on the READ and READ ANY TABLE Privileges
  There are special restrictions on the `READ` and `READ ANY TABLE` privileges.

4.14.4.1 About Managing READ and SELECT Object Privileges

You can grant users either the `READ` or the `SELECT` object privilege.

The grant of these privileges depend on the level of access that you want to allow the user.

Follow these guidelines:

- If you want the user only to be able to query tables, views, materialized views, or synonyms, then you should grant the `READ` object privilege. For example:

  ```
  GRANT READ ON HR.EMPLOYEES TO psmith;
  ```

- If you want the user to be able to perform the following actions in addition to performing the query, then you should grant the user the `SELECT` object privilege:
  - `LOCK TABLE table_name IN EXCLUSIVE MODE;`
  - `SELECT ... FROM table_name FOR UPDATE;`

  For example:

```
GRANT SELECT ON HR.EMPLOYEES TO psmith;
```

In either case, user `psmith` would use a `SELECT` statement to perform query.

Related Topics

- Auditing the READ ANY TABLE and SELECT ANY TABLE Privileges
  The `CREATE AUDIT POLICY` statement can audit the `READ ANY TABLE` and `SELECT ANY TABLE` privileges.

4.14.4.2 Enabling Users to Use the READ Object Privilege to Query Any Table in the Database

The `READ ANY TABLE` system privilege provides the `READ` object privilege for querying any table in the database.

- To enable a user to have the `READ` object privilege for any table in the database, grant the user the `READ ANY TABLE` system privilege.

For example:

```
GRANT READ ANY TABLE TO psmith;
```

As with the `READ` object privilege, the `READ ANY TABLE` system privilege does not enable users to lock tables in exclusive mode nor select tables for update operations. Conversely, the `SELECT ANY TABLE` system privilege enables users to lock the rows of a table, or lock the entire table, through a `SELECT ... FOR UPDATE` statement, in addition to querying any table.

4.14.4.3 Restrictions on the READ and READ ANY TABLE Privileges

There are special restrictions on the `READ` and `READ ANY TABLE` privileges.

These privileges are as follows:

- The `READ` object privilege has no effect on the requirements of the `SQL92_SECURITY` standard. If the `SQL92_SECURITY` initialization parameter has been set to `TRUE`, then its requirement that users must be granted the `SELECT` object privilege in addition to `UPDATE` or `DELETE` in order to run the `UPDATE` or `DELETE` statements is not relaxed to require that `READ` is sufficient instead of `SELECT`.
- If Oracle Database Vault is enabled, remember that the `SQL92_SECURITY` initialization parameter is automatically set to `TRUE`. Hence, `UPDATE` and `DELETE` statements will fail if the user has only been granted the `READ` object privilege or the `READ ANY TABLE` system privilege. In this case, you must grant the user the `SELECT` object privilege or, if the user is a trusted user, the `SELECT ANY TABLE` system privilege.

4.14.5 Object Privilege Use with Synonyms

The `CREATE SYNONYM` statement create synonyms for database objects.

You can create synonyms for the following objects: tables, views, sequences, operators, procedures, stored functions, packages, materialized views, Java class schema objects, user-defined object types, or other synonyms.

If you grant users the privilege to use the synonym, then the object privileges granted on the underlying objects apply whether the user references the base object by name or by using the synonym.

For example, suppose user OE creates the following synonym for the CUSTOMERS table:

```
CREATE SYNONYM customer_syn FOR CUSTOMERS;
```

Then OE grants the READ privilege on the customer_syn synonym to user HR.

```
GRANT READ ON customer_syn TO HR;
```

User HR then tries either of the following queries:

```
SELECT COUNT(*) FROM OE.customer_syn;

SELECT COUNT(*) FROM OE.CUSTOMERS;
```

Both queries will yield the same result:

```
  COUNT(*)
----------
       319
```

Be aware that when you grant the synonym to another user, the grant applies to the underlying object that the synonym represents, not to the synonym itself. For example, if user HR queries the ALL_TAB_PRIVS data dictionary view for their privileges, this user will learn the following:

```
SELECT TABLE_SCHEMA, TABLE_NAME, PRIVILEGE
FROM ALL_TAB_PRIVS
WHERE TABLE_SCHEMA = 'OE';

TABLE_SCHEMA  TABLE_NAME  PRIVILEGE
------------  ----------  ------------------
OE            CUSTOMER    READ
OE            OE          INHERIT PRIVILEGES
```

The results show that in addition to other privileges, the user has the READ privilege for the underlying object of the customer_syn synonym, which is the OE.CUSTOMER table.

At this point, if user OE then revokes the READ privilege on the customer_syn synonym from HR, here are the results if HR checks their privileges again:

```
TABLE_SCHEMA  TABLE_NAME  PRIVILEGE
------------  ----------  ------------------
OE            OE          INHERIT PRIVILEGES
```

User HR no longer has the READ privilege for the OE.CUSTOMER table. If HR tries to query the OE.CUSTOMERS table, then the following error appears:

```
SELECT COUNT(*) FROM OE.CUSTOMERS;

ERROR at line 1:
ORA-00942: table or view does not exist
```

4.14.6 Sharing Application Common Objects

Database objects can be configured so that their metadata links, data links, and extended data links can be shared in the application root.

- Metadata-Linked Application Common Objects
  A metadata link enables database objects in an application pluggable database (PDB) to share metadata with objects in the application root.

- Data-Linked Application Common Objects
  Data links manage references and privileges for common objects.
- Extended Data-Linked Application Common Objects
  Extended data links can combine data from an application pluggable database (PDB) with an application root.

Related Topics

- *Oracle Database Administrator's Guide*

4.14.6.1 Metadata-Linked Application Common Objects

A metadata link enables database objects in an application pluggable database (PDB) to share metadata with objects in the application root.

Metadata links are useful for reducing disk and memory requirements because they store only one copy of an object's metadata (such as the source code for a PL/SQL package) for identically defined objects (such as Oracle-suppled PL/SQL packages). This improves the performance of upgrade operations because changes to this metadata will be made in one place, the application root.

You must configure the metadata link from the application root. You can use the `DBMS_PDB.SET_MEDATADATA_LINKED` PL/SQL procedure to change the database object to a metadata link.

The following example shows how to use the `DBMS_PDB.SET_METADATA_LINKED` procedure to change the `update_emp_rating` procedure in the `hr_mgr` schema to a metadata-linked application common object.

Example 4-7 Changing an Object to a Metadata-Linked Application Common Object

```
BEGIN
  DBMS_PDB.SET_METADATA_LINKED (
   SCHEMA_NAME => 'hr_mgr',
   OBJECT_NAME => 'update_emp_rating',
   NAMESPACE   => 1);
END;
/
```

Any common user can own metadata links. Metadata links can only be used to share the metadata of application common objects that their creator in the application root owns.

To find if an object has a metadata link, query the `SHARING` column of the `DBA_OBJECTS` data dictionary view.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

4.14.6.2 Data-Linked Application Common Objects

Data links manage references and privileges for common objects.

A data link (previously called an object link) enables references to, and privilege grants on, objects in an application root from an application pluggable database (PDB) that belong to the same application container.

If an application common user who owns an application common object wants to grant access to that object to a user in a PDB, then the application common user can accomplish this by granting the privilege on a data link that points to the common object. For example, you can

create data links for objects such as tables, views, clusters, sequences, or PL/SQL packages if you want to ensure that an operation on the object (such as a query, a DML, an EXECUTE statement, and so on) that refers to this operation affects the same object regardless of the container in which the operation is performed.

You must configure the data link from an application root. You can use the DBMS_PDB.SET_DATA_LINKED PL/SQL procedure to change the data link. You should use this procedure only when you want to convert an existing object to become data linked.

The following example shows how to use the DBMS_PDB.SET_DATA_LINKED procedure to change the emp_ratings table in the hr_mgr schema to a data-linked application common object.

Example 4-8 Changing an Object to a Data-Linked Application Common Object

```
BEGIN
  DBMS_PDB.SET_DATA_LINKED (
   SCHEMA_NAME => 'hr_mgr',
   OBJECT_NAME => 'emp_ratings',
   NAMESPACE   => 1);
END;
/
```

Any common user can own data links.

To find if an object has an data link, query the SHARING column of the DBA_OBJECTS data dictionary view. The NAMESPACE column of this view provides the namespace number.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

4.14.6.3 Extended Data-Linked Application Common Objects

Extended data links can combine data from an application pluggable database (PDB) with an application root.

An extended data link enables a data link to combine data found in a table in the PDB with data from a corresponding table in the application root.

You can think of an extended data link as a hybrid of a metadata link and a data link. An extended data-link object in an application PDB inherits metadata from the extended data link object in the application root. The data for the object is stored in the application root and, optionally, in each application PDB. You can create extended data links for tables and views only. When you query the DBA_OBJECTS data dictionary view for an extended data link object, this view returns extended data link-related rows from both the application PDB and the application root.

You must configure the extended data link from an application root. You can use the DBMS_PDB.SET_EXT_DATA_LINKED PL/SQL procedure to change the database object to an extended data link.

The following example shows how to use the DBMS_PDB.SET_EXT_DATA_LINKED procedure to change the emp_salaries data dictionary view in the hr_mgr schema to an extended data-linked application common object.

Example 4-9 Changing an Object to an Extended Data-Linked Application Common Object

```
BEGIN
  DBMS_PDB.SET_EXT_DATA_LINKED (
   SCHEMA_NAME => 'hr_mgr',
```

```
   OBJECT_NAME => 'emp_salaries',
   NAMESPACE   => 1);
END;
/
```

Any common user can own extended data links.

To find if an object has an extended data link, query the `SHARING` column of the `DBA_OBJECTS` data dictionary view.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

4.15 Managing Dictionary Protection for Oracle-Maintained Schemas

Oracle-maintained schemas such as `AUDSYS` have dictionary protection to prevent users from using system privileges on these schemas.

- About Managing Dictionary Protection for Oracle-Maintained Schemas
  By default, Oracle-maintained schemas have dictionary protection, but this protection can be temporarily removed if necessary.
- Enabling Dictionary Protection in an Oracle-Maintained Schema
  To enable dictionary protection for an Oracle-maintained schema, use the `ALTER USER` statement with the `ENABLE DICTIONARY PROTECTION` clause.
- Disabling Dictionary Protection in an Oracle-Maintained Schema
  To disable dictionary protection from an Oracle-maintained schema, use the `ALTER USER` statement with the `DISABLE DICTIONARY PROTECTION` clause.

4.15.1 About Managing Dictionary Protection for Oracle-Maintained Schemas

By default, Oracle-maintained schemas have dictionary protection, but this protection can be temporarily removed if necessary.

When a schema is dictionary protected, other users cannot use system privileges (including `ANY` privileges) on the schema, even if they have been granted the system privilege on the schema. Only the `SELECT ANY DICTIONARY` and `ANALYZE ANY DICTIONARY` system privileges can be used on a dictionary-protected schema. Users can still use object privileges on the schema, assuming that the user has been granted the object privilege on the schema. Users who are marked as dictionary protected cannot log in to the database.

For example, suppose an administrator grants the `CREATE USER` and `ALTER USER` system privilege to a user or a tool such as Oracle Identity Manager that is responsible for adding users to the database and managing their passwords. In previous releases, that account would have the privileges that are necessary for setting passwords for accounts that have higher levels of privilege, such as `SYSDB` or `SYSKM`. A malicious user of that account could change the password for `SYSKM`, log in as `SYSKM` with the new password, and then have access to information that they normally would not be allowed to have. This feature prevents that kind of attack.

To find schemas that are dictionary protected, run the following query:

```
SELECT USERNAME, DICTIONARY_PROTECTED FROM DBA_USERS WHERE
DICTIONARY_PROTECTED='YES';
```

The `ALL_USERS` data dictionary view also has the `DICTIONARY_PROTECTED` column.

In most cases, you should allow these schemas to continue to have dictionary protection, but if you need to, you can temporarily disable dictionary protection by using the `ALTER USER` statement with the `DISABLE DICTIONARY PROTECTION` clause. You can manage dictionary protection for Oracle-maintained schemas only if you are logged in as user `SYS` with the `SYSDBA` administrative privilege.

The underlying schemas of the following administrative privileges have dictionary protection enabled. When a user is granted one of these privileges and logs in, the user is using the underlying schema.

- `SYSBACKUP`
- `SYSKM`
- `SYSDG`

4.15.2 Enabling Dictionary Protection in an Oracle-Maintained Schema

To enable dictionary protection for an Oracle-maintained schema, use the `ALTER USER` statement with the `ENABLE DICTIONARY PROTECTION` clause.

1. Log in to the CDB root or to a PDB as user `SYS` with the `SYSDBA` administrative privilege.

   Only user `SYS` with `SYSDBA` can enable a user schema to have dictionary privileges.

2. To find schemas that are not dictionary protected, run a query similar to the following:

   ```
   SELECT USERNAME, DICTIONARY_PROTECTED FROM DBA_USERS
   WHERE DICTIONARY_PROTECTED = 'NO' ORDER BY USERNAME;
   ```

3. Run the `ALTER USER` statement with the `ENABLE DICTIONARY PROTECTION` clause.

   For example:

   ```
   ALTER USER AUDSYS ENABLE DICTIONARY PROTECTION;
   ```

4. Ensure that the schema now has dictionary protection.

   For example:

   ```
   SELECT DICTIONARY_PROTECTED FROM DBA_USERS WHERE USERNAME ='AUDSYS';
   ```

4.15.3 Disabling Dictionary Protection in an Oracle-Maintained Schema

To disable dictionary protection from an Oracle-maintained schema, use the `ALTER USER` statement with the `DISABLE DICTIONARY PROTECTION` clause.

1. Log in to the CDB root or to a PDB as user `SYS` with the `SYSDBA` administrative privilege.

   Only user `SYS` with `SYSDBA` can remove dictionary privileges from a user schema.

2. Query the DBA_USERS data dictionary view to find if the schema has dictionary protection.

   For example:

   ```
   SELECT DICTIONARY_PROTECTED FROM DBA_USERS
   WHERE USERNAME = 'AUDSYS';
   ```

   If the output for DICTIONARY_PROTECTED is YES, then you can remove dictionary protection from the schema.

3. Run the ALTER USER statement with the DISABLE DICTIONARY PROTECTION clause.

   For example:

   ```
   ALTER USER AUDSYS DISABLE DICTIONARY PROTECTION;
   ```

4.16 Table Privileges

Object privileges for tables enable table security at the DML or DDL level of operation.

- How Table Privileges Affect Data Manipulation Language Operations
  You can grant privileges to use the DELETE, INSERT, SELECT, and UPDATE DML operations on tables and views.
- How Table Privileges Affect Data Definition Language Operations
  The ALTER, INDEX, and REFERENCES privileges allow DDL operations to be performed on a table.

4.16.1 How Table Privileges Affect Data Manipulation Language Operations

You can grant privileges to use the DELETE, INSERT, SELECT, and UPDATE DML operations on tables and views.

Grant these privileges only to users and roles that need to query or manipulate data in a table.

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

For example, if you do not want data entry users to alter the salary column of the employees table, then 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.

4.16.2 How Table Privileges Affect 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 these 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, you can grant the REFERENCES privilege 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 their 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).

4.17 View Privileges

You can apply DML object privileges to views, similar to tables.

- Privileges Required to Create Views
  To create a view, you must have specific privileges.
- Privileges to Query Views in Other Schemas
  A view owner must be granted SELECT WITH GRANT OPTION on the base table of their view before users can query the view from a schema that is different from the schema in which the view is located.
- The Use of Views to Increase Table Security
  Database views can increase table security by restricting the data that users can see.

4.17.1 Privileges Required to Create Views

To create a view, you must have specific privileges.

Object privileges for a view allow various DML operations, which affect the base tables from which the view is derived.

These privileges to create a view are as follows:

- You must be 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 the schema of another user)
- You must be 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
- In addition, before you can grant other users access to you view, you must have object privileges to the base objects with the GRANT OPTION clause or appropriate system privileges with the ADMIN OPTION clause. If you do not have these privileges, then you cannot to grant other users access to your view. If you try, an ORA-01720: grant option does not exist for *object_name* error is raised, with *object_name* referring to the view's underlying object for which you do not have the sufficient privilege.

Related Topics

- *Oracle Database SQL Language Reference*

4.17.2 Privileges to Query Views in Other Schemas

A view owner must be granted `SELECT WITH GRANT OPTION` on the base table of their view before users can query the view from a schema that is different from the schema in which the view is located.

4.17.3 The Use of Views to Increase Table Security

Database views can increase table security by restricting the data that users can see.

To use a view, the user must have the appropriate privileges but only for the view itself, not its underlying objects. However, if access privileges for the underlying objects of the view are removed, then the user no longer has access.

This behavior occurs because the security domain that is used when a user queries the view is that of the definer of the view. If the privileges on the underlying objects are revoked from the view's definer, then the view becomes invalid, and no one can use the view. Therefore, even if a user has been granted access to the view, the user may not be able to use the view if the definer's rights have been revoked from the view's underlying objects.

For example, suppose User A creates a view. User A has definer's rights on the underlying objects of the view. User A then grants the `SELECT` privilege on that view to User B so that User B can query the view. But if User A no longer has access to the underlying objects of that view, then User B no longer has access either.

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

- **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` pseudo column, whose values always refer to the current user. This view combines both column-level security and value-based security.

4.18 Procedure Privileges

The EXECUTE privilege enables users to run procedures and functions, either standalone or in packages.

- The Use of the EXECUTE Privilege for Procedure Privileges
  The EXECUTE privilege is a very powerful privilege that should be handled with caution.
- Procedure Execution and Security Domains
  The EXECUTE object privilege for a procedure can be used to run a procedure or compile a program unit that references the procedure.
- System Privileges Required to Create or Replace a Procedure
  You must have specific privileges to create or replace a procedure in your own schema or in another user's schema.
- System Privileges Required to Compile a Procedure
  You must have specific privileges to compile both standalone procedures and procedures that are part of a package.
- How Procedure Privileges Affect Packages and Package Objects
  The powerful EXECUTE privilege enables users to run any public procedures or functions within a package.

4.18.1 The Use of the EXECUTE Privilege for Procedure Privileges

The EXECUTE privilege is a very powerful privilege that should be handled with caution.

The EXECUTE privilege is the only **object privilege** for procedures, including standalone procedures and functions, and for those within packages.

You should grant this privilege only to users who must run a procedure or compile another procedure that calls a desired procedure. You can find the privileges that a user has been granted by querying the DBA_SYS_PRIVS data dictionary view.

4.18.2 Procedure Execution and Security Domains

The EXECUTE object privilege for a procedure can be used to run a procedure or compile a program unit that references the procedure.

Oracle Database performs a run-time privilege check when any PL/SQL unit is called. A user with the EXECUTE ANY PROCEDURE system privilege can run any procedure in the database. Privileges to run procedures can be granted to a user through roles.

Related Topics

- About Definer's Rights and Invoker's Rights
  Definer's rights and invoker's rights are used to control access to privileges during user-defined procedure executions necessary to run a user-created procedure, or program unit.
- *Oracle Database PL/SQL Packages and Types Reference*

4.18.3 System Privileges Required to Create or Replace a Procedure

You must have specific privileges to create or replace a procedure in your own schema or in another user's schema.

To create or replace a procedure in your own schema, you must have the CREATE PROCEDURE system privilege. To create or replace a procedure in another user's schema, you must have the CREATE 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 need to have been explicitly granted the necessary privileges (system or object) on all objects referenced by the procedure. You cannot obtain the required privileges through roles. This includes the EXECUTE privilege for any procedures that are called inside the procedure being created.

> **Note:**
>
> Triggers require that privileges on referenced objects be granted directly to the owner of the trigger. Anonymous PL/SQL blocks can use any privilege, whether the privilege is granted explicitly or through a role.

4.18.4 System Privileges Required to Compile a Procedure

You must have specific privileges to compile both standalone procedures and procedures that are part of a package.

To compile a standalone procedure, you should run the ALTER PROCEDURE statement with the COMPILE clause. To compile a procedure that is part of a package, you should run the ALTER PACKAGE statement.

The following example shows how to compile a standalone procedure.

```
ALTER PROCEDURE psmith.remove_emp COMPILE;
```

If the standalone or packaged procedure is in another user's schema, you must have the ALTER ANY PROCEDURE privilege to recompile it. You can recompile procedures in your own schema without any privileges.

4.18.5 How Procedure Privileges Affect Packages and Package Objects

The powerful EXECUTE privilege enables users to run any public procedures or functions within a package.

- About the Effect of Procedure Privileges on Packages and Package Objects
  The EXECUTE object privilege for a package applies to any procedure or function within this package.
- Example: Procedure Privileges Used in One Package
  The CREATE PACKAGE BODY statement can create a package body that contains procedures to manage procedure privileges used in one package.
- Example: Procedure Privileges and Package Objects
  The CREATE PACKAGE BODY statement can create a package body containing procedure definitions to manage procedure privileges and package objects.

4.18.5.1 About the Effect of Procedure Privileges on Packages and Package Objects

The EXECUTE object privilege for a package applies to any procedure or function within this package.

A user with the`EXECUTE` object privilege for a package can run any public procedure or function in the package, and can access or modify the value of any public package variable.

You cannot grant specific `EXECUTE` privileges for individual constructs in a package. Therefore, you may find it useful to consider two alternatives for establishing security when developing procedures, functions, and packages for a database application. The following examples describe these alternatives.

4.18.5.2 Example: Procedure Privileges Used in One Package

The `CREATE PACKAGE BODY` statement can create a package body that contains procedures to manage procedure privileges used in one package.

Example 4-10 shows four procedures created in the bodies of two packages.

Example 4-10 Procedure Privileges Used in One Packagee

```
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;
```

The following `GRANT EXECUTE` statements enable the `big_bosses` and `little_bosses` roles to run the appropriate procedures:

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

4.18.5.3 Example: Procedure Privileges and Package Objects

The `CREATE PACKAGE BODY` statement can create a package body containing procedure definitions to manage procedure privileges and package objects.

Example 4-11 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.

Example 4-11 Procedure Privileges and Package Objects

```
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;
```

Be aware that granting `EXECUTE` privilege for a package provides uniform access to all package objects.

4.19 Type Privileges

You can control system and object privileges for types, methods, and objects.

- System Privileges for Named Types
  System privileges for named types can enable users to perform actions such as creating named types in their own schemas.
- Object Privileges for Named Types
  The only object privilege that applies to named types is `EXECUTE`.
- Method Execution Model for Named Types
  The method execution for named types is the same as any other stored PL/SQL procedure.
- Privileges Required to Create Types and Tables Using Types
  To create a type, you must have the appropriate privileges.
- Example: Privileges for Creating Types and Tables Using Types
  The `EXECUTE` privilege with the `GRANT OPTION` is required for users to grant the `EXECUTE` privilege on a type to other users.
- 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.

- Type Dependencies
  As with stored objects, such as procedures and tables, types that are referenced by other objects are called dependencies.

4.19.1 System Privileges for Named Types

System privileges for named types can enable users to perform actions such as creating named types in their own schemas.

Table 4-8 lists system privileges for named types (object types, VARRAYs, and nested tables).

Table 4-8 System Privileges for Named Types

Privilege	Enables 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 RESOURCE role includes the CREATE TYPE system privilege. The DBA role includes all of these privileges.

4.19.2 Object Privileges for Named Types

The only object privilege that applies to named types is EXECUTE.

If the EXECUTE privilege exists on a named type, then 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 methods in the type, including the type constructor. This is similar to the EXECUTE privilege on a stored PL/SQL procedure.

4.19.3 Method Execution Model for Named Types

The method execution for named types is the same as any other stored PL/SQL procedure.

Users must be granted the appropriate privileges for using the named types, such as the EXECUTE privilege. As with all privilege grants, only grant these privileges to trusted users. You can find the privileges that a user has been granted by querying the DBA_SYS_PRIVS data dictionary view.

Related Topics

- Procedure Privileges
  The EXECUTE privilege enables users to run procedures and functions, either standalone or in packages.

4.19.4 Privileges Required to Create Types and Tables Using Types

To create a type, you must have the appropriate privileges.

These privileges are as follows:

- 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 the schema of another user. These privileges can be acquired explicitly or through a role.
- The owner of the type must be 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 obtain the required privileges through roles.
- If the type owner intends to grant access to the type to other users, then the owner must receive the `EXECUTE` privileges to the referenced types with the `GRANT OPTION` or the `EXECUTE ANY TYPE` system privilege with the `ADMIN OPTION`. If not, then 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 the following additional requirements:

- The owner of the table must have been directly granted the `EXECUTE` object privilege to access all types referenced by the table, or has been granted the `EXECUTE ANY TYPE` system privilege. The owner cannot exercise the required privileges if these privileges were granted through roles.
- If the table owner intends to grant access to the table to other users, then the owner must have the `EXECUTE` privilege to the referenced types with the `GRANT OPTION` or the `EXECUTE ANY TYPE` system privilege with the `ADMIN OPTION`. If not, then the table owner has insufficient privileges to grant access on the table.

Related Topics

- Table Privileges
  Object privileges for tables enable table security at the DML or DDL level of operation.

4.19.5 Example: Privileges for Creating Types and Tables Using Types

The `EXECUTE` privilege with the `GRANT OPTION` is required for users to grant the `EXECUTE` privilege on a type to other users.

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

- `user1`
- `user2`
- `user3`

The following DDL is run in the schema of `user1`:

```
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;
```

The following DDL is performed in the schema of `user2`:

```
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.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.type1` with the `GRANT OPTION`:

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

The following statements can be successfully run by `user3`:

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

> **Note:**
>
> The `CONNECT` role presently retains only the `CREATE SESSION` and `SET CONTAINER` privileges.

4.19.6 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.

Table 4-9 lists the privileges for object tables.

Table 4-9 Privileges for Object Tables

Privilege	Enables you to...
`SELECT`	Access an object and its attributes from the table
`UPDATE`	Modify the attributes of the objects that make up the rows in the table
`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 to interpret the type instance images. When a client requests type information, Oracle Database checks for the `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;
```

In addition, consider the following two queries:

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

For either query, Oracle Database checks the `SELECT` privilege of the user for the `emp` table. For the first query, the user must obtain the `emp_type` type information to interpret the data. When the query accesses the `emp_type` type, Oracle Database checks the `EXECUTE` privilege of the user.

The second query, however, does not involve named types, so Oracle Database does not check type privileges.

In addition, by 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 Database 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 Database to check for the `SELECT` privilege on the containing object table.
- Modifying an existing object or flushing an object from the object cache causes Oracle Database to check for the `UPDATE` privilege on the destination object table.
- Flushing a new object causes Oracle Database to check for the `INSERT` privilege on the destination object table.
- Deleting an object causes Oracle Database to check for the `DELETE` privilege on the destination table.
- Pinning an object of a named type causes Oracle Database to check `EXECUTE` privilege on the object.

Modifying the attributes of an object in a client third-generation language application causes Oracle Database to update the entire object. Therefore, the user needs the `UPDATE` privilege on the object table. Having the `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 Database does not support column-level privileges for object tables.

4.19.7 Type Dependencies

As with stored objects, such as procedures and tables, types that are referenced by other objects are called dependencies.

There are some special issues for types on which tables depend. 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 are when necessary privileges required to use the type are revoked, or the type or dependent types are dropped. If 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 was dropped can never be accessed again, and the only permissible action is to drop the table.

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

4.20 Grants of User Privileges and Roles

The `GRANT` statement provides privileges for a user to perform specific actions, such as executing a procedure.

- Granting System Privileges and Roles to Users and Roles
  Before you grant system privileges and roles to users and roles, be aware of how privileges for these types of grants work.
- Granting Object Privileges to Users and Roles
  You can grant object privileges to users and roles, and enable the grantee to grant the privilege to other users.

4.20.1 Granting System Privileges and Roles to Users and Roles

Before you grant system privileges and roles to users and roles, be aware of how privileges for these types of grants work.

- Privileges for Grants of System Privileges and Roles to Users and Roles
  You can use the `GRANT` SQL statement to grant system privileges and roles to users and roles.
- Example: Granting a System Privilege and a Role to a User
  You can use the `GRANT` statement to grant system privileges and roles to users.
- Example: Granting the EXECUTE Privilege on a Directory Object
  You can use the `GRANT` statement to grant the `EXECUTE` privilege on a directory object.
- Use of the ADMIN Option to Enable Grantee Users to Grant the Privilege
  The `WITH ADMIN OPTION` clause can be used to expand the capabilities of a privilege grant.
- Creating a New User with the GRANT Statement
  You can create a new user and grant this user a privilege in one `GRANT` SQL statement.

4.20.1.1 Privileges for Grants of System Privileges and Roles to Users and Roles

You can use the `GRANT` SQL statement to grant system privileges and roles to users and roles.

The following privileges are required:

- To grant a system privilege, a user must be granted the system privilege with the `ADMIN` option or must be granted the `GRANT ANY PRIVILEGE` system privilege.
- To grant a role, a user must be granted the role with the `ADMIN` option or was granted the `GRANT ANY ROLE` system privilege.

> **Note:**
>
> Object privileges cannot be granted along with system privileges and roles in the same `GRANT` statement.

4.20.1.2 Example: Granting a System Privilege and a Role to a User

You can use the `GRANT` statement to grant system privileges and roles to users.

Example 4-12 grants the system privilege `CREATE SESSION` and the `accts_pay` role to the user `jward`.

Example 4-12 Granting a System Privilege and a Role to a User

```
GRANT CREATE SESSION, accts_pay TO jward;
```

4.20.1.3 Example: Granting the EXECUTE Privilege on a Directory Object

You can use the `GRANT` statement to grant the `EXECUTE` privilege on a directory object.

Example 4-12 grants the `EXECUTE` privilege on the `exec_dir` directory object to the user `jward`.

Example 4-13 Granting the EXECUTE Privilege on a Directory Object

```
GRANT EXECUTE ON DIRECTORY exec_dir TO jward;
```

4.20.1.4 Use of the ADMIN Option to Enable Grantee Users to Grant the Privilege

The `WITH ADMIN OPTION` clause can be used to expand the capabilities of a privilege grant.

These capabilities are as follows:

- The grantee can grant or revoke the system privilege or role to or from any other user or role in the database. Users cannot revoke a role from themselves.
- The grantee can grant the system privilege or role with the `ADMIN` option.
- The grantee of a role can alter or drop the role.

Example 4-14 grants the `new_dba` role with the `WITH ADMIN OPTION` clause to user `michael`.

Example 4-14 Granting the ADMIN Option

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

User `michael` is able to not only use all of the privileges implicit in the `new_dba` role, but this user can also grant, revoke, and drop the `new_dba` role as deemed necessary. Because of these powerful capabilities, use caution when granting system privileges or roles with the `ADMIN` option. These privileges are usually reserved for a security administrator, and are rarely granted to other administrators or users of the system. Be aware that when a user creates a role, the role is automatically granted to the creator with the `ADMIN` option.

4.20.1.5 Creating a New User with the GRANT Statement

You can create a new user and grant this user a privilege in one `GRANT` SQL statement.

In most cases, you will want to grant the user the `CREATE SESSION` privilege.

- To create a new user with the `GRANT` statement, include the privilege and the `IDENTIFIED BY` clause.

For example, to create user `psmith` as a new user while granting `psmith` the `CREATE SESSION` system privilege:

```
GRANT CREATE SESSION TO psmith IDENTIFIED BY password;
```

If you specify a password using the `IDENTIFIED BY` clause, and the user name does not exist in the database, then a new user with that user name and password is created.

Related Topics

- Creating User Accounts
  A user account can have restrictions such as profiles, a default role, and tablespace restrictions.
- Minimum Requirements for Passwords
  Oracle provides a set of minimum requirements for passwords.

4.20.2 Granting Object Privileges to Users and Roles

You can grant object privileges to users and roles, and enable the grantee to grant the privilege to other users.

- About Granting Object Privileges to Users and Roles
  You can use the `GRANT` statement to grant object privileges to roles and users.
- How the WITH GRANT OPTION Clause Works
  The `WITH GRANT OPTION` clause with the `GRANT` statement can enable a grantee to grant object privileges to other users.
- Grants of Object Privileges on Behalf of the Object Owner
  The `GRANT ANY OBJECT PRIVILEGE` system privilege enables users to grant and revoke any object privilege on behalf of the object owner.
- Grants of Privileges on Columns
  You can grant `INSERT`, `UPDATE`, or `REFERENCES` privileges on individual columns in a table.
- Row-Level Access Control
  You can provide access control at the row level, that is, within objects, but not with the `GRANT` statement.

4.20.2.1 About Granting Object Privileges to Users and Roles

You can use the `GRANT` statement to grant object privileges to roles and users.

To grant an object privilege, you must fulfill one of the following conditions:

- You own the object specified.
- You have been granted the `GRANT ANY OBJECT PRIVILEGE` system privilege. This privilege enables you to grant and revoke privileges on behalf of the object owner.
- The `WITH GRANT OPTION` clause was specified when you were granted the object privilege.

> **Note:**
>
> System privileges and roles cannot be granted along with object privileges in the same `GRANT` statement.

The following example grants the `READ`, `INSERT`, and `DELETE` object privileges for all columns of the `emp` table to the users `jfee` and `tsmith`.

```
GRANT READ, INSERT, DELETE ON emp TO jfee, tsmith;
```

To grant all object privileges on the `salary` view to user `jfee`, use the `ALL` keyword as shown in the following example:

```
GRANT ALL ON salary TO jfee;
```

> **Note:**
>
> A grantee cannot regrant access to objects unless the original grant included the `GRANT OPTION`. Thus in the example just given, `jfee` cannot use the `GRANT` statement to grant object privileges to anyone else.

4.20.2.2 How the WITH GRANT OPTION Clause Works

The `WITH GRANT OPTION` clause with the `GRANT` statement can enable a grantee to grant object privileges to other users.

The user whose schema contains an object is automatically granted all associated object privileges with the `WITH GRANT OPTION` clause. This special privilege allows the grantee several expanded privileges:

- The grantee can grant the object privilege to any user in the database, with or without the `GRANT OPTION`, and to any role in the database.
- If both of the following conditions are true, then the grantee can create views on the table, and grant the corresponding privileges on the views to any user or role in the database:
  - The grantee receives object privileges for the table with the `GRANT OPTION`.
  - The grantee has the `CREATE VIEW` or `CREATE ANY VIEW` system privilege.

> **Note:**
>
> The `WITH GRANT OPTION` clause is not valid if you try to grant an object privilege to a role. Oracle Database prevents the propagation of object privileges through roles so that grantees of a role cannot propagate object privileges received by means of roles.

4.20.2.3 Grants of Object Privileges on Behalf of the Object Owner

The `GRANT ANY OBJECT PRIVILEGE` system privilege enables users to grant and revoke any object privilege on behalf of the object owner.

This privilege provides a convenient means for database and application administrators to grant access to objects in any schema without requiring that they connect to the schema. Login credentials do not need to be maintained for schema owners who have this privilege, which reduces the number of connections required during configuration.

This system privilege is part of the Oracle Database supplied `DBA` role and is thus granted (with the `ADMIN option`) to any user connecting `AS SYSDBA` (user `SYS`). As with other system privileges, the `GRANT ANY OBJECT PRIVILEGE` system privilege can only be granted by a user who possesses the `ADMIN option`.

The *recorded* grantor of access rights to an object is either the object owner or the person exercising the `GRANT ANY OBJECT PRIVILEGE` system privilege. If the grantor with `GRANT ANY OBJECT PRIVILEGE` does *not* have the object privilege with the `GRANT OPTION`, then the object owner is shown as the grantor. Otherwise, when that grantor has the object privilege with the `GRANT OPTION`, then that grantor is recorded as the grantor of the grant.

> **Note:**
>
> The audit record generated by the `GRANT` statement always shows the actual user who performed the grant.

For example, consider the following scenario. User `adams` possesses the `GRANT ANY OBJECT PRIVILEGE` system privilege. This user does not possess any other grant privileges. User `adams` issues the following statement:

```
GRANT SELECT ON HR.EMPLOYEES TO blake WITH GRANT OPTION;
```

If you examine the `DBA_TAB_PRIVS` view, then you will see that `HR` is shown as the grantor of the privilege:

```
SELECT GRANTEE, GRANTOR, PRIVILEGE, GRANTABLE
  FROM DBA_TAB_PRIVS
  WHERE TABLE_NAME = 'EMPLOYEES' and OWNER = 'HR';

GRANTEE  GRANTOR PRIVILEGE    GRANTABLE
-------- ------- -----------  ----------
BLAKE    HR       SELECT      YES
```

Now assume that user `blake` also has the `GRANT ANY OBJECT PRIVILEGE` system. He issues the following statement:

```
GRANT SELECT ON HR.EMPLOYEES TO clark;
```

In this case, when you query the `DBA_TAB_PRIVS` view again, you see that `blake` is shown as being the grantor of the privilege:

```
GRANTEE  GRANTOR  PRIVILEGE  GRANTABLE
-------- -------- ---------  ----------
BLAKE    HR       SELECT     YES
CLARK    BLAKE    SELECT     NO
```

This occurs because `blake` already possesses the `SELECT` privilege on `HR.EMPLOYEES` with the `GRANT OPTION`.

Related Topics

- Revokes of Object Privileges on Behalf of the Object Owner
  The `GRANT ANY OBJECT PRIVILEGE` system privilege can be used to revoke any object privilege where the object owner is the grantor.

4.20.2.4 Grants of Privileges on Columns

You can grant `INSERT`, `UPDATE`, or `REFERENCES` privileges on individual columns in a table.

> **Note:**
>
> Before granting a column-specific `INSERT` privilege, determine if the table contains any columns on which `NOT NULL` constraints are defined. Granting selective insert capability without including the `NOT NULL` columns prevents the user from inserting any rows into the table. To avoid this situation, ensure that each `NOT NULL` column can either be inserted into or has a non-`NULL` default value. Otherwise, the grantee will not be able to insert rows into the table and will receive an error.

The following statement grants the `INSERT` privilege on the `acct_no` column of the `accounts` table to user `psmith`:

```
GRANT INSERT (acct_no) ON accounts TO psmith;
```

In the following example, object privilege for the `ename` and `job` columns of the `emp` table are granted to the users `jfee` and `tsmith`:

```
GRANT INSERT(ename, job) ON emp TO jfee, tsmith;
```

You can grant the `INSERT` and `UPDATE` privileges on individual columns of a view.

4.20.2.5 Row-Level Access Control

You can provide access control at the row level, that is, within objects, but not with the `GRANT` statement.

To perform this kind of access control, you must use either Oracle Virtual Private Database (VPD) or Oracle Label Security (OLS).

Related Topics

- Using Oracle Virtual Private Database to Control Data Access
  Oracle Virtual Private Database (VPD) enables you to filter users who access data.
- *Oracle Label Security Administrator's Guide*

4.21 Revokes of Privileges and Roles from a User

When you revoke system or object privileges, be aware of the cascading effects of revoking a privilege.

- Revokes of System Privileges and Roles
  The `REVOKE` SQL statement revokes system privileges and roles.

- Revokes of Object Privileges
  You can revoke multiple object privileges, object privileges on behalf of an object owner, column-selective object privileges, and the REFERENCES object privilege.
- Cascading Effects of Revoking Privileges
  There are no cascading effects for revoked object privileges related to DDL operations, but there are cascading effects for object privilege revocations.

4.21.1 Revokes of System Privileges and Roles

The REVOKE SQL statement revokes system privileges and roles.

Any user with the ADMIN option for a system privilege or role can revoke the privilege or role from any other database user or role. The revoker does not have to be the user that originally granted the privilege or role. Users with GRANT ANY ROLE can revoke *any* role.

Example 4-15 revokes the CREATE TABLE system privilege and the accts_rec role from user psmith:

Example 4-15 Revoking a System Privilege and a Role from a User

```
REVOKE CREATE TABLE, accts_rec FROM psmith;
```

Be aware that the ADMIN option for a system privilege or role cannot be selectively revoked. Instead, revoke the privilege or role, and then grant the privilege or role again but without the ADMIN option.

4.21.2 Revokes of Object Privileges

You can revoke multiple object privileges, object privileges on behalf of an object owner, column-selective object privileges, and the REFERENCES object privilege.

- About Revokes of Object Privileges
  To revoke an object privilege, you must meet the appropriate requirements.
- Revokes of Multiple Object Privileges
  The REVOKE statement can revoke multiple privileges on one object.
- Revokes of Object Privileges on Behalf of the Object Owner
  The GRANT ANY OBJECT PRIVILEGE system privilege can be used to revoke any object privilege where the object owner is the grantor.
- Revokes of Column-Selective Object Privileges
  GRANT and REVOKE operations for column-specific operations have different privileges and restrictions.
- Revokes of the REFERENCES Object Privilege
  When you revoke the REFERENCES object privilege, it affects foreign key constraints.

4.21.2.1 About Revokes of Object Privileges

To revoke an object privilege, you must meet the appropriate requirements.

The requirements are either of the following conditions:

- You previously granted the object privilege to the user or role.
- You possess the GRANT ANY OBJECT PRIVILEGE system privilege that enables you to grant and revoke privileges on behalf of the object owner.

You can only revoke the privileges that you, the person who granted the privilege, directly authorized. You cannot revoke grants that were made by other users to whom you granted the GRANT OPTION. However, there is a cascading effect. If the object privileges of the user who granted the privilege are revoked, then the object privilege grants that were propagated using the GRANT OPTION are revoked as well.

4.21.2.2 Revokes of Multiple Object Privileges

The REVOKE statement can revoke multiple privileges on one object.

Assuming you are the original grantor of the privilege, the following statement revokes the SELECT and INSERT privileges on the emp table from users jfee and psmith:

```
REVOKE SELECT, INSERT ON emp FROM jfee, psmith;
```

The following statement revokes all object privileges for the dept table that you originally granted to the human_resource role:

```
REVOKE ALL ON dept FROM human_resources;
```

> **Note:**
>
> The GRANT OPTION for an object privilege cannot be selectively revoked. Instead, revoke the object privilege and then grant it again but without the GRANT OPTION. Users cannot revoke object privileges from themselves.

4.21.2.3 Revokes of Object Privileges on Behalf of the Object Owner

The GRANT ANY OBJECT PRIVILEGE system privilege can be used to revoke any object privilege where the object owner is the grantor.

This occurs when the object privilege is granted by the object owner, or on behalf of the owner by any user holding the GRANT ANY OBJECT PRIVILEGE system privilege.

In a situation where the object privilege was granted by both the owner of the object and the user executing the REVOKE statement (who has both the specific object privilege and the GRANT ANY OBJECT PRIVILEGE system privilege), Oracle Database only revokes the object privilege granted by the user issuing the REVOKE statement. This can be illustrated by continuing the example that is shown earlier of a grant of object privileges made on behalf of an object owner.

At this point, user blake granted the SELECT privilege on HR.EMPLOYEES to clark. Even though blake possesses the GRANT ANY OBJECT PRIVILEGE system privilege, this user also holds the specific object privilege, thus this grant is attributed to him. Assume that user HR also grants the SELECT privilege on HR.EMPLOYEES to user clark. A query of the DBA_TAB_PRIVS view shows that the following grants are in effect for the HR.EMPLOYEES table:

```
GRANTEE  GRANTOR PRIVILEGE    GRANTABLE
-------- ------- -----------  ----------
BLAKE    HR      SELECT       YES
CLARK    BLAKE   SELECT       NO
CLARK    HR      SELECT       NO
```

User blake now issues the following REVOKE statement:

```
REVOKE  SELECT ON HR.EMPLOYEES FROM clark;
```

Only the object privilege for user clark granted by user blake is removed. The grant by the object owner, HR, remains.

```
GRANTEE  GRANTOR PRIVILEGE    GRANTABLE
-------- ------- -----------  ----------
BLAKE    HR       SELECT      YES
CLARK    HR       SELECT      NO
```

If blake issues the REVOKE statement again, then this time the effect is to remove the object privilege granted by adams (on behalf of HR), using the GRANT ANY OBEJCT PRIVILEGE system privilege.

Related Topics

- Grants of Object Privileges on Behalf of the Object Owner
  The GRANT ANY OBJECT PRIVILEGE system privilege enables users to grant and revoke any object privilege on behalf of the object owner.

4.21.2.4 Revokes of Column-Selective Object Privileges

GRANT and REVOKE operations for column-specific operations have different privileges and restrictions.

Although users can grant column-specific INSERT, UPDATE, and REFERENCES privileges for tables and views, they cannot selectively revoke column-specific privileges with a similar REVOKE statement. Instead, the grantor must first revoke the object privilege for all columns of a table or view, and then selectively repeat the grant of the column-specific privileges that the grantor intends to keep in effect.

For example, assume that role human_resources was granted the UPDATE privilege on the deptno and dname columns of the table dept. To revoke the UPDATE privilege on just the deptno column, issue the following two statements:

```
REVOKE UPDATE ON dept FROM human_resources;
GRANT UPDATE (dname) ON dept TO human_resources;
```

The REVOKE statement revokes the UPDATE privilege on all columns of the dept table from the role human_resources. The GRANT statement then repeats, restores, or reissues the grant of the UPDATE privilege on the dname column to the role human_resources.

4.21.2.5 Revokes of the REFERENCES Object Privilege

When you revoke the REFERENCES object privilege, it affects foreign key constraints.

If the grantee of the REFERENCES object privilege has used the privilege to create a foreign key constraint (that currently exists), then the grantor can revoke the privilege only by specifying the CASCADE CONSTRAINTS option in the REVOKE statement.

For example:

```
REVOKE REFERENCES ON dept FROM jward CASCADE CONSTRAINTS;
```

Any foreign key constraints currently defined that use the revoked REFERENCES privilege are dropped when the CASCADE CONSTRAINTS clause is specified.

4.21.3 Cascading Effects of Revoking Privileges

There are no cascading effects for revoked object privileges related to DDL operations, but there are cascading effects for object privilege revocations.

- Cascading Effects When Revoking System Privileges
  There are no cascading effects when you revoke a system privilege that is related to DDL operations.
- Cascading Effects When Revoking Object Privileges
  Revoking an object privilege can have cascading effects.

4.21.3.1 Cascading Effects When Revoking System Privileges

There are no cascading effects when you revoke a system privilege that is related to DDL operations.

This applies regardless of whether the privilege was granted with or without the `ADMIN` option.

For example, assume the following:

1. The security administrator grants the `CREATE TABLE` system privilege to user `jfee` with the `ADMIN option`.
2. User `jfee` creates a table.
3. User `jfee` grants the `CREATE TABLE` system privilege to user `tsmith`.
4. User `tsmith` creates a table.
5. The security administrator revokes the `CREATE TABLE` system privilege from user `jfee`.
6. The table created by user `jfee` continues to exist. User `tsmith` still has the table and the `CREATE TABLE` system privilege.

You can observe cascading effects when you revoke a system privilege related to a DML operation. If the `SELECT ANY TABLE` privilege is revoked from a user, then all procedures contained in the user's schema relying on this privilege can no longer be run successfully until the privilege is reauthorized.

4.21.3.2 Cascading Effects When Revoking Object Privileges

Revoking an object privilege can have cascading effects.

Note the following:

- **Object definitions that depend on a DML object privilege can be affected if the DML object privilege is revoked.** For example, assume that the body of the `test` procedure includes a SQL statement that queries data from the `emp` table. If the `SELECT` privilege on the `emp` table is revoked from the owner of the `test` procedure, then the procedure can no longer be run successfully.
- **When a REFERENCES privilege for a table is revoked from a user, any foreign key integrity constraints that are defined by the user and require the dropped REFERENCES privilege are automatically dropped.** For example, assume that user `jward` is granted the `REFERENCES` privilege for the `deptno` column of the `dept` table. This user now creates a foreign key on the `deptno` column in the `emp` table that references the `deptno` column of the `dept` table. If the `REFERENCES` privilege on the `deptno` column of the

dept table is revoked, then the foreign key constraint on the deptno column of the emp table is dropped in the same operation.

- **The object privilege grants propagated using the GRANT OPTION are revoked if the object privilege of a grantor is revoked.** For example, assume that user1 is granted the SELECT object privilege on the emp table with the GRANT OPTION, and grants the SELECT privilege on emp to user2. Subsequently, the SELECT privilege is revoked from user1. This REVOKE statement is also cascaded to user2. Any objects that depend on the revoked SELECT privilege of user1 and user2 can also be affected, as described earlier.

Object definitions that require the ALTER and INDEX DDL object privileges are not affected if the ALTER or INDEX object privilege is revoked. For example, if the INDEX privilege is revoked from a user that created an index on a table that belongs to another user, then the index continues to exist after the privilege is revoked.

4.22 Grants and Revokes of Privileges to and from the PUBLIC Role

You can grant and revoke privileges and roles from the role PUBLIC.

Because PUBLIC is accessible to every database user, all privileges and roles granted to PUBLIC are accessible to every database user. By default, PUBLIC does not have privileges granted to it.

Security administrators and database users should grant a privilege or role to PUBLIC only if every database user requires the privilege or role. This recommendation reinforces the general rule that, at any given time, each database user should have only the privileges required to accomplish the current group tasks successfully.

Revoking a privilege from the PUBLIC role can cause significant cascading effects. If any privilege related to a DML operation is revoked from PUBLIC (for example, SELECT ANY TABLE or UPDATE ON emp), then all procedures in the database, including functions and packages, must be *reauthorized* before they can be used again. Therefore, be careful when you grant and revoke DML-related privileges to or from PUBLIC.

Related Topics

- Guidelines for Securing Data
  Oracle provides guidelines for securing data on your system.
- *Oracle Database Administrator's Guide*

4.23 Grants of Roles Using the Operating System or Network

Using the operating system or network to manage roles can help centralize the role management in a large enterprise.

- About Granting Roles Using the Operating System or Network
  The operating system on which Oracle Database runs can be used to grant roles to users at connect time.
- Operating System Role Identification
  The OS_ROLES initialization parameter can be used to control how the operating system identifies roles.

- Operating System Role Management
  When you use operating system-managed roles, remember that database roles are being granted to an operating system user.
- Role Grants and Revokes When OS_ROLES Is Set to TRUE
  Setting the `OS_ROLES` initialization parameter to `TRUE` enables the operating system to manage role grants and revokes to users.
- Role Enablements and Disablements When OS_ROLES Is Set to TRUE
  Setting the `OS_ROLES` initialization parameter to `TRUE` enables the `SET ROLE` statement to dynamically enable roles granted by the operating system.
- Network Connections with Operating System Role Management
  By default, users cannot connect to the database through a shared server if the operating system manages roles.

4.23.1 About Granting Roles Using the Operating System or Network

The operating system on which Oracle Database runs can be used to grant roles to users at connect time.

This feature is an alternative to a security administrator explicitly having to granting and revoking database roles to and from users using `GRANT` and `REVOKE` statements.

Roles can be administered using the operating system and passed to Oracle Database when a user creates a session. As part of this mechanism, the default roles of a user and the roles granted to a user with the `ADMIN` option can be identified. If the operating system is used to authorize users for roles, then all roles must be created in the database and privileges assigned to the role with `GRANT` statements.

Roles can also be granted through a network service.

The advantage of using the operating system to identify the database roles of a user is that privilege management for an Oracle database can be externalized. The security facilities offered by the operating system control user privileges. This option may offer advantages of centralizing security for several system activities, such as the following situation:

- MVS Oracle administrators want RACF groups to identify database user roles.
- UNIX Oracle administrators want UNIX groups to identify database user roles.
- VMS Oracle administrators want to use rights identifiers to identify database user roles.

The main disadvantage of using the operating system to identify the database roles of a user is that privilege management can only be performed at the role level. Individual privileges cannot be granted using the operating system, but they can still be granted inside the database using `GRANT` statements.

A second disadvantage of using this feature is that, by default, users cannot connect to the database through the shared server or any other network connection if the operating system is managing roles. However, you can change this default.

You can use operating system authentication for a database administrator only for the CDB root. You cannot use it for PDBs, the application root, or application PDBs.

> **Note:**
>
> The features described in this section are available only on some operating systems. See your operating system-specific Oracle Database documentation to determine if you can use these features.

Related Topics

- Network Connections with Operating System Role Management
  By default, users cannot connect to the database through a shared server if the operating system manages roles.

4.23.2 Operating System Role Identification

The `OS_ROLES` initialization parameter can be used to control how the operating system identifies roles.

To have the database use the operating system to identify the database roles of each user when a session is created, you can set the initialization parameter `OS_ROLES` to `TRUE`.

If the instance is current running, you must restart the instance. When a user tries to create a session with the database, Oracle Database initializes the user security domain using the database roles identified by the operating system.

To identify database roles for a user, the operating system account for each Oracle Database user must have operating system identifiers (these may be called groups, rights identifiers, or other similar names) that indicate which database roles are to be available for the user. Role specification can also indicate which roles are the default roles of a user and which roles are available with the `ADMIN` option. No matter which operating system is used, the role specification at the operating system level follows the format:

```
ora_ID_ROLE[[_d][_a][_da]]
```

In this specification:

- `ID` has a definition that varies on different operating systems. For example, on VMS, `ID` is the instance identifier of the database; on VMS, it is the computer type; and on UNIX, it is the system `ID`.

  `ID` is case-sensitive to match your `ORACLE_SID`. `ROLE` is not case-sensitive.
- `ROLE` is the name of the database role.
- `d` is an optional character that indicates this role is to be a default role of the database user.
- `a` is an optional character that indicates this role is to be granted to the user with the `ADMIN` option. This allows the user to grant the role to other roles only. Roles cannot be granted to users if the operating system is used to manage roles.

  If either the `d` or `a` character is specified, then precede that character by an underscore (_).

For example, suppose an operating system account has the following roles identified in its profile:

```
ora_PAYROLL_ROLE1
ora_PAYROLL_ROLE2_a
ora_PAYROLL_ROLE3_d
ora_PAYROLL_ROLE4_da
```

When the corresponding user connects to the `payroll` instance of Oracle Database, `role3` and `role4` are defaults, while `role2` and `role4` are available with the `ADMIN` option.

4.23.3 Operating System Role Management

When you use operating system-managed roles, remember that database roles are being granted to an operating system user.

Any database user to which the operating system user is able to connect will have the authorized database roles enabled. For this reason, you should consider defining all Oracle Database users as `IDENTIFIED EXTERNALLY` if you are using `OS_ROLES = TRUE`, so that the database accounts are tied to the operating system account that was granted privileges.

4.23.4 Role Grants and Revokes When OS_ROLES Is Set to TRUE

Setting the `OS_ROLES` initialization parameter to `TRUE` enables the operating system to manage role grants and revokes to users.

Any previous granting of roles to users using `GRANT` statements do not apply. However, they are still listed in the data dictionary. Only the role grants to users made at the operating system level apply. Users can still grant privileges to roles and users.

> **Note:**
>
> If the operating system grants a role to a user with the `ADMIN` option, then the user can grant the role only to other roles.

4.23.5 Role Enablements and Disablements When OS_ROLES Is Set to TRUE

Setting the `OS_ROLES` initialization parameter to `TRUE` enables the `SET ROLE` statement to dynamically enable roles granted by the operating system.

This still applies, even if the role was defined to require a password or operating system authorization. However, any role not identified in the operating system account of a user cannot be specified in a `SET ROLE` statement, even if a role was granted using a `GRANT` statement when `OS_ROLES = FALSE`. (If you specify such a role, then Oracle Database ignores it.)

When `OS_ROLES` is set to `TRUE`, then the user can enable up to 148 roles. Remember that this number includes other roles that may have been granted to the role.

4.23.6 Network Connections with Operating System Role Management

By default, users cannot connect to the database through a shared server if the operating system manages roles.

This restriction is the default because a remote user could impersonate another operating system user over an unsecure connection.

If you are not concerned with this security risk and want to use operating system role management with the shared server, or any other network connection, then set the initialization

parameter REMOTE_OS_ROLES to TRUE. The change takes effect the next time you start the instance and mount the database. The default setting of this parameter is FALSE.

The REMOTE_OS_ROLES initialization parameter is deprecated in Oracle Database 23ai

4.24 How Grants and Revokes Work with SET ROLE and Default Role Settings

Privilege grants and the SET ROLE statement affect when and how grants and revokes take place.

- When Grants and Revokes Take Effect
  Depending on the privilege that is granted or revoked, a grant or revoke takes effect at different times.
- How the SET ROLE Statement Affects Grants and Revokes
  During a user session, a user or an application can use the SET ROLE statement multiple times to change the roles enabled for the session.
- Specifying the Default Role for a User
  When a user logs on, Oracle Database enables all privileges granted explicitly to the user and all privileges in the user's default roles.
- The Maximum Number of Roles That a User Can Have Enabled
  You can grant a user as many roles as you want, but no more than 148 roles can be enabled for a logged-in user at any given time.

4.24.1 When Grants and Revokes Take Effect

Depending on the privilege that is granted or revoked, a grant or revoke takes effect at different times.

The grants and revokes take effect as follows:

- All grants and revokes of system and object privileges to anything (users, roles, and PUBLIC) take immediate effect.
- All grants and revokes of roles to anything (users, other roles, PUBLIC) take effect only when a current user session issues a SET ROLE statement to reenable the role after the grant and revoke, or when a new user session is created after the grant or revoke.

You can see which roles are currently enabled by examining the SESSION_ROLES data dictionary view.

4.24.2 How the SET ROLE Statement Affects Grants and Revokes

During a user session, a user or an application can use the SET ROLE statement multiple times to change the roles enabled for the session.

The user must already be granted the roles that are named in the SET ROLE statement.

The following example enables the role clerk, which you have already been granted, and specifies the password.

```
SET ROLE clerk IDENTIFIED BY password;
```

Replace *password* with a password that is secure.

The following example shows how to use `SET ROLE` to disable all roles.

```
SET ROLE NONE;
```

Related Topics

- Guidelines for Securing Passwords
  Oracle provides guidelines for securing passwords in a variety of situations.

4.24.3 Specifying the Default Role for a User

When a user logs on, Oracle Database enables all privileges granted explicitly to the user and all privileges in the user's default roles.

1. Ensure that the user who you want to set the default role for has been directly granted the role with a `GRANT` statement, or that the role was created by the user with the `CREATE ROLE` privilege.
2. Use the `ALTER USER` statement with the `DEFAULT ROLE` clause to specify the default roles for the user.

For example, to set the default roles `payclerk` and `pettycash` for user `jane`:

```
ALTER USER jane DEFAULT ROLE payclerk, pettycash;
```

You cannot set default roles for a user in the `CREATE USER` statement. When you first create a user, the default user role setting is `ALL`, which causes all roles subsequently granted to the user to be default roles. Use the `ALTER USER` statement to limit the default user roles.

> **Note:**
>
> When you create a role (other than a global role or an application role), it is granted implicitly to you, and your set of default roles is updated to include the new role. Be aware that only 148 roles can be enabled for a user session. When aggregate roles, such as the `DBA` role, are granted to a user, the roles granted to the role are included in the number of roles the user has. For example, if a role has 20 roles granted to it and you grant that role to the user, then the user now has 21 additional roles. Therefore, when you grant new roles to a user, use the `DEFAULT ROLE` clause of the `ALTER USER` statement to ensure that not too many roles are specified as that user's default roles.

Related Topics

- *Oracle Database SQL Language Reference*

4.24.4 The Maximum Number of Roles That a User Can Have Enabled

You can grant a user as many roles as you want, but no more than 148 roles can be enabled for a logged-in user at any given time.

The 148 role maximum includes roles that are granted to other roles, not just top-level roles. Therefore, not all privileges will be available to this user during the user session. As a best practice, restrict the number of roles granted to a user to the minimum roles the user needs.

Related Topics

- Guidelines for Securing Roles
  Oracle provides guidelines for role management.

4.25 Configuring Read-Only Users

You can override the privileges and roles that have been granted to a user by making the user a read-only user.

This allows `SELECT` operations but will not permit `CREATE`, `INSERT`, `UPDATE`, or `DELETE`.

This feature enables an administrator to block users from using their full set of privileges for as long as the user is set to read-only. For example, a database user who has been granted full privileges to insert, update, and delete data, but then made read-only will be unable to perform `INSERT`, `UPDATE`, or `DELETE` operations until they are altered to be read-write. The read-only restriction overrides privilege grants, including schema or system grants. Read-only restrictions even override the `DBA` role. If the user tries to perform these types of operations, an `ORA-28194: Can perform read operations only` error appears.

Use cases for configuring read-only users are as follows:

- A user or application normally has access to the system as required by the application or granted by the administrator, but for maintenance or investigative reasons the administrators may want to prohibit any changes to the database. In that case, you can set a user to `READ ONLY` without having to modify the user's other privileges.
- An otherwise empowered user must have read-only access to parts of an application. In the application code, you can embed a simple `ALTER SESSION` statement to grant the user `READ ONLY` access.

Read-only users may be appropriate in cases where users normally need only read access to data, but need the ability to elevate to read-write under certain conditions. With a single SQL command, these accounts can change “modes” and gain the ability to perform data updates.

To configure the read-only restriction for a user, you use the `CREATE USER` or `ALTER USER` statement. To find the read-only status of a user, you can query the `READ_ONLY` column of the `DBA_USERS` or `ALL_USERS` data dictionary view.

Table 4-10 Read-Only User Modification and Verification Procedures

Operation	Procedure
Creating a user as read-only	`CREATE USER user_name READ ONLY;`
Modifying a user to be read-only	`ALTER USER user_name READ ONLY;`
Enabling the user to have read-write access again	`ALTER USER user_name READ WRITE;`

Table 4-10 (Cont.) Read-Only User Modification and Verification Procedures

Operation	Procedure

Finding the read-only status of a user	`SELECT USERNAME, READ_ONLY from DBA_USERS`
	`WHERE USERNAME = 'user_name';`
	
	Output similar to the following appears. For example, if user `PFITCH` has read-only access:
	
	`USERNAME READ_ONLY`
	`------------ -----------`
	`PFITCH YES`

Related Topics

- *Oracle Multitenant Administrator's Guide*

4.26 User Privilege and Role Data Dictionary Views

You can use special queries to find information about various types of privilege and role grants.

- Data Dictionary Views to Find Information about Privilege and Role Grants
  Oracle Database provides data dictionary views that describe privilege and role grants.
- Query to List All System Privilege Grants
  The `DBA_SYS_PRIVS` data dictionary view returns all system privilege grants made to roles and users.
- Query to List Schema Privilege Grants
  The `DBA_SCHEMA_PRIVS` data dictionary view, accessed by users who have the `DBA` role, lists all the schema privileges granted to users or roles in the database.
- Query to List All Role Grants
  The `DBA_ROLE_PRIVS` query returns all the roles granted to users and other roles.
- Query to List Object Privileges Granted to a User
  The `DBA_TAB_PRIVS` and `DBA_COL_PRIVS` data dictionary views list object privileges that have bee granted to users.
- Query to List the Current Privilege Domain of Your Session
  The `SESSION_ROLES` and `SESSION_PRIVS` data dictionary views list the current privilege domain of a database session.
- Query to List Roles of the Database
  The `DBA_ROLES` data dictionary view lists all roles of a database and the authentication used for each role.
- Query to List Information About the Privilege Domains of Roles
  The `ROLE_ROLE_PRIVS`, `ROLE_SYS_PRIVS`, and `ROLE_TAB_PRIVS` data dictionary views list information about the privilege domains of roles.

4.26.1 Data Dictionary Views to Find Information about Privilege and Role Grants

Oracle Database provides data dictionary views that describe privilege and role grants.

Table 4-11 lists views that you can query to access information about grants of privileges and roles.

Table 4-11 Data Dictionary Views That Display Privilege and Role Information

View	Description
ALL_COL_PRIVS	Describes all column object grants for which the current user or PUBLIC is the object owner, grantor, or grantee
ALL_COL_PRIVS_MADE	Lists column object grants for which the current user is object owner or grantor
ALL_COL_PRIVS_RECD	Describes column object grants for which the current user or PUBLIC is the grantee
ALL_TAB_PRIVS	Lists the grants on objects where the user or PUBLIC is the grantee
ALL_TAB_PRIVS_MADE	Lists the all object grants made by the current user or made on the objects owned by the current user
ALL_TAB_PRIVS_RECD	Lists object grants for which the user or PUBLIC is the grantee
DBA_COL_PRIVS	Describes all column object grants in the database
DBA_CONTAINER_DATA	Displays default (user-level) and object-specific CONTAINER_DATA attributes. Objects that are created with the CONTAINER_DATA clause include CONTAINER_DATA attributes.
DBA_EPG_DAD_AUTHORIZATION	Describes the database access descriptors (DAD) that are authorized to use a different user's privileges
DBA_LOCKDOWN_PROFILES	Describes information that pertains to PDB lockdown profiles
DBA_OBJECTS	Lists objects that have object links or metadata links. To find these objects, query the OBJECT_NAME and SHARING columns.
DBA_SCHEMA_PRIVS	List all the schema privileges that have been granted to users or roles in the database
DBA_TAB_PRIVS	Lists all grants on all objects in the database
DBA_ROLES	Lists all roles that exist in the database, including secure application roles. Note that it does not list the PUBLIC role
DBA_ROLE_PRIVS	Lists roles directly granted to users and roles
DBA_SYS_PRIVS	Lists system privileges granted to users and roles
ROLE_ROLE_PRIVS	Lists roles granted to other roles. Information is provided only about roles to which the user has access
ROLE_SCHEMA_PRIVS	List all the schema privileges that have been granted to the enabled roles of the current user
ROLE_SYS_PRIVS	Lists system privileges granted to roles. Information is provided only about roles to which the user has access
ROLE_TAB_PRIVS	Lists object privileges granted to roles. Information is provided only about roles to which the user has access
SESSION_PRIVS	Lists the privileges that are currently enabled for the user

Table 4-11 (Cont.) Data Dictionary Views That Display Privilege and Role Information

View	Description
SESSION_SCHEMA_PRIVS	Lists all the schema privileges that have been granted to the current user and the schema privileges that have been granted to the enabled roles of the current user
SESSION_ROLES	Lists all roles that are enabled for the current user. Note that it does not list the PUBLIC role
USER_APPLICATION_ROLES	Enables the current user to see all the application roles that have been granted to the user
USER_COL_PRIVS	Describes column object grants for which the current user is the object owner, grantor, or grantee
USER_COL_PRIVS_MADE	Describes column object grants for which the current user is the object owner
USER_COL_PRIVS_RECD	Describes column object grants for which the current user is the grantee
USER_EPG_DAD_AUTHORIZATION	Describes the database access descriptors (DAD) that are authorized to use a different user's privileges
USER_ROLE_PRIVS	Lists roles directly granted to the current user
USER_SCHEMA_PRIVS	Lists all the schema privileges that have been granted to the current user
USER_TAB_PRIVS	Lists grants on all objects where the current user is the grantee
USER_SYS_PRIVS	Lists system privileges granted to the current user
USER_TAB_PRIVS_MADE	Lists grants on all objects owned by the current user
USER_TAB_PRIVS_RECD	Lists object grants for which the current user is the grantee
V$ENABLEDSCHEMAPRIVS	Lists the schema privileges that have been granted to the current user
V$PWFILE_USERS	Lists all users in the current PDB who have been granted administrative privileges

The following table lists views that you can query to access information about grants of privileges and roles.

This section provides some examples of using these views. For these examples, assume the following statements were issued:

```
CREATE ROLE security_admin IDENTIFIED BY password;

GRANT CREATE PROFILE, ALTER PROFILE, DROP PROFILE,
    CREATE ROLE, DROP ANY ROLE, GRANT ANY ROLE, AUDIT ANY,
    AUDIT SYSTEM, CREATE USER, BECOME USER, ALTER USER, DROP USER
    TO security_admin WITH ADMIN OPTION;

GRANT READ, DELETE ON SYS.AUD$ TO security_admin;

GRANT security_admin, CREATE SESSION TO swilliams;

GRANT security_admin TO system_administrator;

GRANT CREATE SESSION TO jward;

GRANT READ, DELETE ON emp TO jward;
```

```
GRANT INSERT (ename, job) ON emp TO swilliams, jward;
```

Related Topics

- *Oracle Database Reference*

4.26.2 Query to List All System Privilege Grants

The `DBA_SYS_PRIVS` data dictionary view returns all system privilege grants made to roles and users.

For example:

```
SELECT GRANTEE, PRIVILEGE, ADM FROM DBA_SYS_PRIVS;

GRANTEE            PRIVILEGE                        ADM
--------------     -------------------------------- ---
SECURITY_ADMIN     ALTER PROFILE                    YES
SECURITY_ADMIN     ALTER USER                       YES
SECURITY_ADMIN     AUDIT ANY                        YES
SECURITY_ADMIN     AUDIT SYSTEM                     YES
SECURITY_ADMIN     BECOME USER                      YES
SECURITY_ADMIN     CREATE PROFILE                   YES
SECURITY_ADMIN     CREATE ROLE                      YES
SECURITY_ADMIN     CREATE USER                      YES
SECURITY_ADMIN     DROP ANY ROLE                    YES
SECURITY_ADMIN     DROP PROFILE                     YES
SECURITY_ADMIN     DROP USER                        YES
SECURITY_ADMIN     GRANT ANY ROLE                   YES
SWILLIAMS          CREATE SESSION                   NO
JWARD              CREATE SESSION                   NO
```

Related Topics

- *Oracle Database Reference*

4.26.3 Query to List Schema Privilege Grants

The `DBA_SCHEMA_PRIVS` data dictionary view, accessed by users who have the `DBA` role, lists all the schema privileges granted to users or roles in the database.

For example:

```
SELECT GRANTEE, PRIVILEGE, SCHEMA FROM DBA_SCHEMA_PRIVS ORDER BY GRANTEE;

GRANTEE             PRIVILEGE           SCHEMA
------------------- ------------------- ---------
PRESTON             SELECT ANY LIBRARY  HR
RLAYTON             SELECT ANY INDEX    HR
```

Related Topics

- *Oracle Database Reference*

4.26.4 Query to List All Role Grants

The `DBA_ROLE_PRIVS` query returns all the roles granted to users and other roles.

For example:

```
SELECT * FROM DBA_ROLE_PRIVS;

GRANTEE            GRANTED_ROLE                       ADM
------------------ ---------------------------------- ---
SWILLIAMS          SECURITY_ADMIN                     NO
```

Related Topics

- *Oracle Database Reference*

4.26.5 Query to List Object Privileges Granted to a User

The `DBA_TAB_PRIVS` and `DBA_COL_PRIVS` data dictionary views list object privileges that have bee granted to users.

The `DBA_TAB_PRIVS` data dictionary view returns all object privileges (not including column-specific privileges) granted to the specified user.

For example:

```
SELECT TABLE_NAME, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
    WHERE GRANTEE = 'jward';

TABLE_NAME   PRIVILEGE    GRANTABLE
-----------  ------------ ----------
EMP          SELECT       NO
EMP          DELETE       NO
```

To list all the column-specific privileges that have been granted, you can use the following query:

```
SELECT GRANTEE, TABLE_NAME, COLUMN_NAME, PRIVILEGE
    FROM DBA_COL_PRIVS;

GRANTEE      TABLE_NAME     COLUMN_NAME      PRIVILEGE
-----------  ------------   -------------    --------------
SWILLIAMS    EMP            ENAME            INSERT
SWILLIAMS    EMP            JOB              INSERT
JWARD        EMP            NAME             INSERT
JWARD        EMP            JOB              INSERT
```

Related Topics

- *Oracle Database Reference*

4.26.6 Query to List the Current Privilege Domain of Your Session

The `SESSION_ROLES` and `SESSION_PRIVS` data dictionary views list the current privilege domain of a database session.

The `SESSION_ROLES` view lists all roles currently enabled for the issuer.

For example:

```
SELECT * FROM SESSION_ROLES;
```

If user `swilliams` has the `security_admin` role enabled and issues the previous query, then Oracle Database returns the following information:

```
ROLE
------------------------------
SECURITY_ADMIN
```

The following query lists all system privileges currently available in the security domain of the issuer, both from explicit privilege grants and from enabled roles:

```
SELECT * FROM SESSION_PRIVS;
```

If user `swilliams` has the `security_admin` role enabled and issues the previous query, then Oracle Database returns the following results:

```
PRIVILEGE
----------------------------------------
AUDIT SYSTEM
CREATE SESSION
CREATE USER
BECOME USER
ALTER USER
DROP USER
CREATE ROLE
DROP ANY ROLE
GRANT ANY ROLE
AUDIT ANY
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
```

If the `security_admin` role is disabled for user `swilliams`, then the first query would return no rows, while the second query would only return a row for the `CREATE SESSION` privilege grant.

Related Topics

- *Oracle Database Reference*

4.26.7 Query to List Roles of the Database

The `DBA_ROLES` data dictionary view lists all roles of a database and the authentication used for each role.

For example:

```
SELECT * FROM DBA_ROLES;

ROLE                  PASSWORD
----------------      --------
CONNECT               NO
RESOURCE              NO
DBA                   NO
SECURITY_ADMIN        YES
```

Related Topics

- *Oracle Database Reference*

4.26.8 Query to List Information About the Privilege Domains of Roles

The `ROLE_ROLE_PRIVS`, `ROLE_SYS_PRIVS`, and `ROLE_TAB_PRIVS` data dictionary views list information about the privilege domains of roles.

For example:

```
SELECT GRANTED_ROLE, ADMIN_OPTION
   FROM ROLE_ROLE_PRIVS
   WHERE ROLE = 'SYSTEM_ADMIN';

GRANTED_ROLE              ADM
----------------          ----
SECURITY_ADMIN            NO
```

The following query lists all the system privileges granted to the `security_admin` role:

```
SELECT * FROM ROLE_SYS_PRIVS WHERE ROLE = 'SECURITY_ADMIN';

ROLE                    PRIVILEGE                      ADM
----------------------- ------------------------------ ---
SECURITY_ADMIN           ALTER PROFILE                 YES
SECURITY_ADMIN           ALTER USER                    YES
SECURITY_ADMIN           AUDIT ANY                     YES
SECURITY_ADMIN           AUDIT SYSTEM                  YES
SECURITY_ADMIN           BECOME USER                   YES
SECURITY_ADMIN           CREATE PROFILE                YES
SECURITY_ADMIN           CREATE ROLE                   YES
SECURITY_ADMIN           CREATE USER                   YES
SECURITY_ADMIN           DROP ANY ROLE                 YES
SECURITY_ADMIN           DROP PROFILE                  YES
SECURITY_ADMIN           DROP USER                     YES
SECURITY_ADMIN           GRANT ANY ROLE                YES
```

The following query lists all the object privileges granted to the `security_admin` role:

```
SELECT TABLE_NAME, PRIVILEGE FROM ROLE_TAB_PRIVS
    WHERE ROLE = 'SECURITY_ADMIN';

TABLE_NAME                    PRIVILEGE
---------------------------   ----------------
AUD$                          DELETE
AUD$                          SELECT
```

Related Topics

- *Oracle Database Reference*