18

SQL Statements: DROP TABLE to LOCK TABLE

This chapter contains the following SQL statements:

DROP TABLE

Purpose

Use the `DROP TABLE` statement to move a table or object table to the recycle bin or to remove the table and all its data from the database entirely.

> **Note:**
>
> Unless you specify the `PURGE` clause, the `DROP TABLE` statement does not result in space being released back to the tablespace for use by other objects, and the space continues to count toward the user's space quota.

For an external table, this statement removes only the table metadata in the database. It has no affect on the actual data, which resides outside of the database.

When you drop a table that is part of a cluster, the table is moved to the recycle bin. However, if you subsequently drop the cluster, then the table is purged from the recycle bin and can no longer be recovered with a `FLASHBACK TABLE` operation.

Dropping a table invalidates dependent objects and removes object privileges on the table. If you want to re-create the table, then you must regrant object privileges on the table, re-create the indexes, integrity constraints, and triggers for the table, and respecify its storage parameters. Truncating has none of these effects. Therefore, removing rows with the `TRUNCATE` statement can be more efficient than dropping and re-creating a table.

> **See Also:**
>
> - CREATE TABLE and ALTER TABLE for information on creating and modifying tables
> - TRUNCATE TABLE and DELETE for information on removing data from a table
> - FLASHBACK TABLE for information on retrieving a dropped table from the recycle bin

Prerequisites

The table must be in your own schema or you must have the `DROP ANY TABLE` system privilege.

You can perform DDL operations (such as `ALTER TABLE`, `DROP TABLE`, `CREATE INDEX`) on a temporary table only when no session is bound to it. A session becomes bound to a temporary table by performing an `INSERT` operation on the table. A session becomes unbound to the temporary table by issuing a `TRUNCATE` statement or at session termination, or, for a transaction-specific temporary table, by issuing a `COMMIT` or `ROLLBACK` statement.

Dropping Private Temporary Tables

You can drop a private temporary table using the existing `DROP TABLE` command. Dropping a private temporary table will not commit an existing transaction. This applies to both transaction-specific and session-specific private temporary tables. Note that a dropped private temporary table will not go into the `RECYCLEBIN`.

Dropping Blockchain and Immutable Tables

Use the `DROP TABLE` statement to drop a blockchain or immutable table. It is recommended that you include the `PURGE` option while dropping these tables. Dropping a blockchain or immutable table removes its definition from the data dictionary, deletes all its rows, and deletes any indexes and triggers defined on the table.

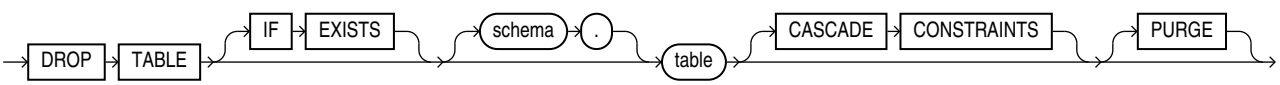
The blockchain or immutable table must be contained in your schema, or you must have the `DROP ANY TABLE` system privilege.

A blockchain or immutable table can be dropped only after it has not been modified for a period of time that is defined by its retention period.

An empty blockchain or immutable table can be dropped regardless of its retention period.

Syntax

***drop_table*::=**



Semantics

IF EXISTS

Specifying `IF EXISTS` drops the table if it exists.

Using `IF NOT EXISTS` with `DROP TABLE` results in `ORA-11544: Incorrect IF EXISTS clause for ALTER/DROP statement.`

schema

Specify the schema containing the table. If you omit *schema*, then Oracle Database assumes the table is in your own schema.

table

Specify the name of the table to be dropped. Oracle Database automatically performs the following operations:

- All rows from the table are dropped.
- All table indexes and domain indexes are dropped, as well as any triggers defined on the table, regardless of who created them or whose schema contains them. If *table* is partitioned, then any corresponding local index partitions are also dropped.
- All the storage tables of nested tables and LOBs of *table* are dropped.
- When you drop a range-, hash-, or list-partitioned table, then the database drops all the table partitions. If you drop a composite-partitioned table, then all the partitions and subpartitions are also dropped.
- When you drop a partitioned table with the `PURGE` keyword, the statement executes as a series of subtransactions, each of which drops a subset of partitions or subpartitions and their metadata. This division of the drop operation into subtransactions optimizes the processing of internal system resource consumption (for example, the library cache), especially for the dropping of very large partitioned tables. As soon as the first subtransaction commits, the table is marked `UNUSABLE`. If any of the subtransactions fails, then the only operation allowed on the table is another `DROP TABLE ... PURGE` statement. Such a statement will resume work from where the previous `DROP TABLE` statement failed, assuming that you have corrected any errors that the previous operation encountered.

  You can list the tables marked `UNUSABLE` by such a drop operation by querying the `status` column of the `*_TABLES`, `*_PART_TABLES`, `*_ALL_TABLES`, or `*_OBJECT_TABLES` data dictionary views, as appropriate.

  > **See Also:**
  >
  > *Oracle Database VLDB and Partitioning Guide* for more information on dropping partitioned tables.

- For an index-organized table, any mapping tables defined on the index-organized table are dropped.
- For a domain index, the appropriate drop routines are invoked. Refer to *Oracle Database Data Cartridge Developer's Guide* for more information on these routines.

- If any statistics types are associated with the table, then the database disassociates the statistics types with the `FORCE` clause and removes any user-defined statistics collected with the statistics type.

  > **See Also:**
  >
  > ASSOCIATE STATISTICS and DISASSOCIATE STATISTICS for more information on statistics type associations

- If the table is not part of a cluster, then the database returns all data blocks allocated to the table and its indexes to the tablespaces containing the table and its indexes.

  To drop a cluster and all its the tables, use the `DROP CLUSTER` statement with the `INCLUDING TABLES` clause to avoid dropping each table individually. See DROP CLUSTER .

- If the table is a base table for a view, a container or master table of a materialized view, or if it is referenced in a stored procedure, function, or package, then the database invalidates these dependent objects but does not drop them. You cannot use these objects unless you re-create the table or drop and re-create the objects so that they no longer depend on the table.

  If you choose to re-create the table, then it must contain all the columns selected by the subqueries originally used to define the materialized views and all the columns referenced in the stored procedures, functions, or packages. Any users previously granted object privileges on the views, stored procedures, functions, or packages need not be regranted these privileges.

  If the table is a master table for a materialized view, then the materialized view can still be queried, but it cannot be refreshed unless the table is re-created so that it contains all the columns selected by the defining query of the materialized view.

  If the table has a materialized view log, then the database drops this log and any other direct-path `INSERT` refresh information associated with the table.

Restrictions on Dropping Tables

- You cannot directly drop the storage table of a nested table. Instead, you must drop the nested table column using the `ALTER TABLE ... DROP COLUMN` clause.
- You cannot drop the parent table of a reference-partitioned table. You must first drop all reference-partitioned child tables.
- You cannot drop a table that uses a flashback data archive for historical tracking. You must first disable the table's use of the flashback archive.

CASCADE CONSTRAINTS

Specify `CASCADE CONSTRAINTS` to drop all referential integrity constraints that refer to primary and unique keys in the dropped table. If you omit this clause, and such referential integrity constraints exist, then the database returns an error and does not drop the table.

PURGE

Specify `PURGE` if you want to drop the table and release the space associated with it in a single step. If you specify `PURGE`, then the database does not place the table and its dependent objects into the recycle bin.

> **Note:**
>
> You cannot roll back a `DROP TABLE` statement with the `PURGE` clause, nor can you recover the table if you have dropped it with the `PURGE` clause.

Using this clause is equivalent to first dropping the table and then purging it from the recycle bin. This clause lets you save one step in the process. It also provides enhanced security if you want to prevent sensitive material from appearing in the recycle bin.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information on the recycle bin and naming conventions for objects in the recycle bin

Examples

Dropping a Table: Example

The following statement drops the `oe.list_customers` table created in "List Partitioning Example".

```
DROP TABLE list_customers PURGE;
```

DROP TABLESPACE

Purpose

Use the `DROP TABLESPACE` statement to remove a tablespace from the database.

When you drop a tablespace, Oracle Database does not place it in the recycle bin. Therefore, you cannot subsequently either purge or undrop the tablespace.

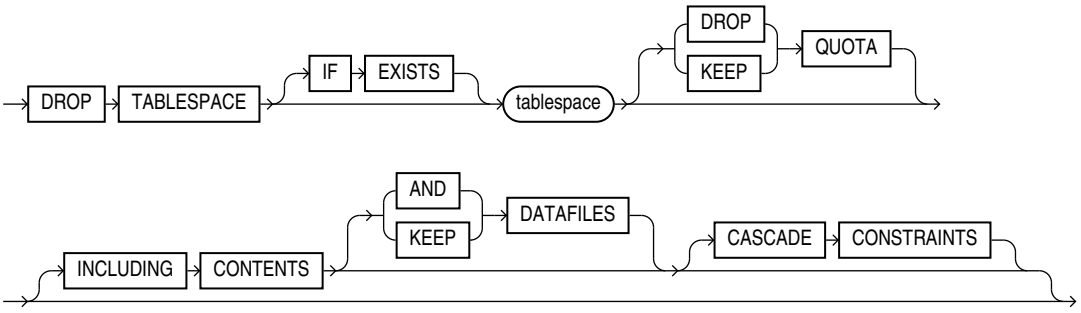
> **See Also:**
>
> CREATE TABLESPACE and ALTER TABLESPACE for information on creating and modifying a tablespace

Prerequisites

You must have the `DROP TABLESPACE` system privilege. You cannot drop a tablespace if it contains any rollback segments holding active transactions.

Syntax

***drop_tablespace*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to drop an existing tablespace.

Specifying `IF NOT EXISTS` with `ALTER` results in error: `Incorrect IF EXISTS clause for ALTER/DROP statement.`

tablespace

Specify the name of the tablespace to be dropped, including those of shadow tablespaces, that store lost write protection updates.

You can drop a tablespace regardless of whether it is online or offline. Oracle recommends that you take the tablespace offline before dropping it to ensure that no SQL statements in currently running transactions access any of the objects in the tablespace.

You cannot drop the `SYSTEM` tablespace. You can drop the `SYSAUX` tablespace only if you have the `SYSDBA` system privilege and you have started the database in `UPGRADE` mode.

You may want to alert any users who have been assigned the tablespace as either a default or temporary tablespace. After the tablespace has been dropped, these users cannot allocate space for objects or sort areas in the tablespace. You can reassign users new default and temporary tablespaces with the `ALTER USER` statement.

Any objects that were previously dropped from the tablespace and moved to the recycle bin are purged from the recycle bin. Oracle Database removes from the data dictionary all metadata about the tablespace and all data files and temp files in the tablespace. The database also automatically drops from the operating system any Oracle-managed data files and temp files in the tablespace. Other data files and temp files are not removed from the operating system unless you specify `INCLUDING CONTENTS AND DATAFILES`.

You cannot use this statement to drop a tablespace group. However, if *`tablespace`* is the only tablespace in a tablespace group, then Oracle Database removes the tablespace group from the data dictionary as well.

Restrictions on Dropping Tablespaces

Dropping tablespaces is subject to the following restrictions:

- You cannot drop a tablespace that contains a domain index or any objects created by a domain index.
- You cannot drop an undo tablespace if it is being used by any instance or if it contains any undo data needed to roll back uncommitted transactions.
- You cannot drop a tablespace that has been designated as the default tablespace for the database. You must first reassign another tablespace as the default tablespace and then drop the old default tablespace.
- You cannot drop a temporary tablespace if it is part of the database default temporary tablespace group. You must first remove the tablespace from the database default temporary tablespace group and then drop it.
- You cannot drop a temporary tablespace if it contains segments that are in use by existing sessions. In this case, no error is raised. The database waits until there are no segments in use by existing sessions and then drops the tablespace.
- You cannot drop a tablespace, even with the `INCLUDING CONTENTS` and `CASCADE CONSTRAINTS` clauses, if doing so would disable a primary key or unique constraint in another tablespace. For example, if the tablespace being dropped contains a primary key index, but the primary key column itself is in a different tablespace, then you cannot drop the tablespace until you have manually disabled the primary key constraint in the other tablespace.

> **See Also:**
>
> *Oracle Database Data Cartridge Developer's Guide* and *Oracle Database Concepts* for more information on domain indexes

{ DROP | KEEP } QUOTA

Specify `DROP QUOTA` to drop all user quotas for the tablespace. Specify `KEEP QUOTA` to retain all user quotas for the tablespace. The default is `KEEP QUOTA`.

You can view all user quotas for a tablespace by querying the `DBA_TS_QUOTAS` data dictionary view.

INCLUDING CONTENTS

Specify `INCLUDING CONTENTS` to drop all the contents of the tablespace, including those of shadow tablespaces that store lost write protection updates. You must specify this clause to drop a tablespace that contains any database objects. If you omit this clause, and the tablespace is not empty, then the database returns an error and does not drop the tablespace.

`DROP TABLESPACE` fails, even if you specify `INCLUDING CONTENTS`, if the tablespace contains some, but not all, of the partitions or subpartitions of a single table. If all the partitions or subpartitions of a partitioned table reside in *`tablespace`*, then `DROP TABLESPACE ... INCLUDING CONTENTS` drops *`tablespace`*, as well as any associated index segments, LOB data and index segments, and nested table data and index segments of *`table`* in other tablespace(s).

For a partitioned index-organized table, if all the primary key index segments are in this tablespace, then this clause will also drop any overflow segments that exist in other tablespaces, as well as any associated mapping table in other tablespaces. If some of the primary key index segments are *not* in this tablespace, then the statement will fail. In that case, before you can drop the tablespace, you must use `ALTER TABLE ... MOVE PARTITION` to move

those primary key index segments into this tablespace, drop the partitions whose overflow data segments are not in this tablespace, and drop the partitioned index-organized table.

If the tablespace contains a master table of a materialized view, then the database invalidates the materialized view.

If the tablespace contains a materialized view log, then the database drops the log and any other direct-path `INSERT` refresh information associated with the table.

AND DATAFILES

When you specify `INCLUDING CONTENTS`, the `AND DATAFILES` clause lets you instruct the database to delete the associated operating system files as well. Oracle Database writes a message to the alert log for each operating system file deleted. This clause is not needed for Oracle Managed Files, because they are removed from the system even if you do not specify `AND DATAFILES`.

KEEP DATAFILES

When you specify `INCLUDING CONTENTS`, the `KEEP DATAFILES` clause lets you instruct the database to leave untouched the associated operating system files, including Oracle Managed Files. You must specify this clause if you are using Oracle Managed Files and you do not want the associated operating system files removed by the `INCLUDING CONTENTS` clause.

CASCADE CONSTRAINTS

Specify `CASCADE CONSTRAINTS` to drop all referential integrity constraints from tables outside *`tablespace`* that refer to primary and unique keys of tables inside *`tablespace`*. If you omit this clause and such referential integrity constraints exist, then Oracle Database returns an error and does not drop the tablespace.

Examples

Dropping a Tablespace: Example

The following statement drops the `tbs_01` tablespace and drops all referential integrity constraints that refer to primary and unique keys inside `tbs_01`:

```
DROP TABLESPACE tbs_01
    INCLUDING CONTENTS
        CASCADE CONSTRAINTS;
```

Dropping a Shadow Tablespace: Example

The following statement tries to move the tracked data in the shadow tablespace to another shadow tablespace. This only works if there are shadow tablespaces in the PDB with enough free space.

```
DROP TABLESPACE <shadow_tablespace_name>
```

The following statement drops the shadow tablespace and all its contents. All the tracking data is lost.

Dropping Shadow Tablespace Including Contents: Example

```
DROP TABLESPACE <shadow_tablespace_name>
    INCLUDING CONTENTS
```

Deleting Operating System Files: Example

The following example drops the `tbs_02` tablespace and deletes all associated operating system data files:

```
DROP TABLESPACE tbs_02
   INCLUDING CONTENTS AND DATAFILES;
```

DROP TABLESPACE SET

> **Note:**
>
> This SQL statement is valid only if you are using Oracle Sharding. For more information on Oracle Sharding, refer to *Oracle Database Administrator's Guide*.

Purpose

Use the `DROP TABLESPACE SET` statement to drop a tablespace set from a shardgroup.

When you drop a tablespace set, Oracle Database does not place it in the recycle bin. Therefore, you cannot subsequently either purge or undrop the tablespace set.

> **See Also:**
>
> CREATE TABLESPACE SET and ALTER TABLESPACE SET

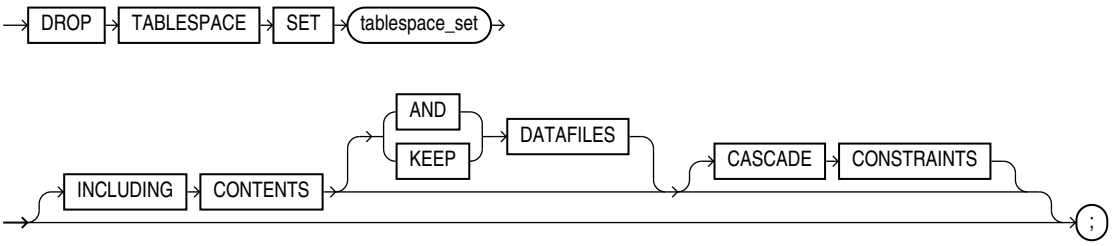
Prerequisites

You must be connected to a shard catalog database as an SDB user.

You must have the `DROP TABLESPACE` system privilege. You cannot drop a tablespace set if its tablespaces contain any rollback segments holding active transactions.

Syntax

***drop_tablespace_set*::=**



Semantics

tablespace_set

Specify the name of the tablespace set to be dropped.

INCLUDING CONTENTS

This clause lets you specify how the database manages objects and datafiles associated with the tablespaces in the tablespace set during the drop operation. The `INCLUDING CONTENTS`

clause has the same semantics here as for the `DROP TABLESPACE` statement. See INCLUDING CONTENTS for the full semantics of this clause.

Examples

Dropping a Tablespace Set: Example

The following statement drops the tablespace set `ts1`:

```
DROP TABLESPACE SET ts1;
```

DROP TRIGGER

Purpose

Triggers are defined using PL/SQL. Refer to *Oracle Database PL/SQL Language Reference* for complete information on creating, altering, and dropping triggers.

Use the `DROP TRIGGER` statement to remove a database trigger from the database.

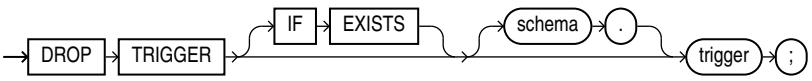
> **See Also:**
>
> CREATE TRIGGER and ALTER TRIGGER

Prerequisites

The trigger must be in your own schema or you must have the `DROP ANY TRIGGER` system privilege. To drop a trigger on `DATABASE` in another user's schema, you must also have the `ADMINISTER DATABASE TRIGGER` system privilege.

Syntax

***drop_trigger*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to drop an existing object.

Specifying `IF NOT EXISTS` with `DROP` results in `ORA-11544: Incorrect IF EXISTS clause for ALTER/DROP statement`.

schema

Specify the schema containing the trigger. If you omit *`schema`*, then Oracle Database assumes the trigger is in your own schema.

trigger

Specify the name of the trigger to be dropped. Oracle Database removes it from the database and does not fire it again.

Examples

Dropping a Trigger: Example

The following statement drops the `salary_check` trigger in the schema `hr`:

```
DROP TRIGGER hr.salary_check;
```

DROP TYPE

Purpose

Object types are defined using PL/SQL. Refer to *Oracle Database PL/SQL Language Reference* for complete information on creating, altering, and dropping object types.

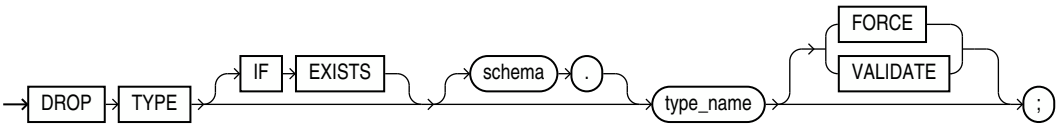
Use the `DROP TYPE` statement to drop the specification and body of an object type, a varray, or a nested table type.

Prerequisites

The object type, varray, or nested table type must be in your own schema or you must have the `DROP ANY TYPE` system privilege.

Syntax

***drop_type*::=**



IF EXISTS

Specify `IF EXISTS` to drop an existing object.

Specifying `IF NOT EXISTS` with `DROP` results in `ORA-11544: Incorrect IF EXISTS clause for ALTER/DROP statement`.

Semantics

schema

Specify the schema containing the type. If you omit *`schema`*, then Oracle Database assumes the type is in your own schema.

type_name

Specify the name of the object, varray, or nested table type to be dropped. You can drop only types with no type or table dependencies. This includes tables in the recycle bin.

When you drop a type, any dependent objects such as subtypes are not placed in the recycle bin and any former dependent objects in the recycle bin are purged.

If *type_name* is a supertype, then this statement will fail unless you also specify FORCE. If you specify FORCE, then the database invalidates all subtypes depending on this supertype.

If *type_name* is a statistics type, then this statement will fail unless you also specify FORCE. If you specify FORCE, then the database first disassociates all objects that are associated with *type_name* and then drops *type_name*.

> **See Also:**
>
> ASSOCIATE STATISTICS and DISASSOCIATE STATISTICS for more information on statistics types

If *type_name* is an object type that has been associated with a statistics type, then the database first attempts to disassociate *type_name* from the statistics type and then drops *type_name*. However, if statistics have been collected using the statistics type, then the database will be unable to disassociate *type_name* from the statistics type, and this statement will fail.

If *type_name* is an implementation type for an indextype, then the indextype will be marked INVALID.

If *type_name* has a public synonym defined on it, then the database will also drop the synonym.

Unless you specify FORCE, you can drop only object types, nested tables, or varray types that are standalone schema objects with no dependencies. This is the default behavior.

> **See Also:**
>
> CREATE INDEXTYPE

FORCE

Specify FORCE to drop the type even if it has dependent database objects. Oracle Database marks UNUSED all columns dependent on the type to be dropped, and those columns become inaccessible.

> **Note:**
>
> Oracle does not recommend that you specify FORCE to drop object types with dependencies. These include dependent tables in the recycle bin. This operation is not recoverable and could cause the data in the dependent tables or columns to become inaccessible.
>
> Refer to *Managing Tables* .

VALIDATE

If you specify `VALIDATE` when dropping a type, then Oracle Database checks for stored instances of this type within substitutable columns of any of its supertypes. If no such instances are found, then the database completes the drop operation.

This clause is meaningful only for subtypes. Oracle recommends the use of this option to safely drop subtypes that do not have any explicit type or table dependencies.

Examples

Dropping an Object Type: Example

The following statement removes object type `person_t`. See *Oracle Database PL/SQL Language Reference* for the example that creates this object type. Any columns that are dependent on person_t are marked `UNUSED` and become inaccessible.

```
DROP TYPE person_t FORCE;
```

DROP TYPE BODY

Purpose

Object types are defined using PL/SQL. Refer to *Oracle Database PL/SQL Language Reference* for complete information on creating, altering, and dropping object types.

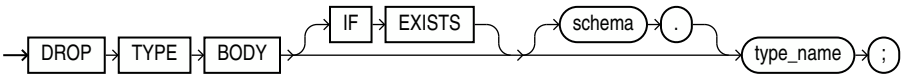
Use the `DROP TYPE BODY` statement to drop the body of an object type, varray, or nested table type. When you drop a type body, the object type specification still exists, and you can re-create the type body. Prior to re-creating the body, you can still use the object type, although you cannot call the member functions.

Prerequisites

The object type body must be in your own schema or you must have the `DROP ANY TYPE` system privilege.

Syntax

***drop_type_body*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to drop an existing object.

Specifying `IF NOT EXISTS` with `DROP` results in `ORA-11544: Incorrect IF EXISTS clause for ALTER/DROP statement`.

schema

Specify the schema containing the object type. If you omit *`schema`*, then Oracle Database assumes the object type is in your own schema.

type_name

Specify the name of the object type body to be dropped.

Restriction on Dropping Type Bodies

You can drop a type body only if it has no type or table dependencies.

Examples

Dropping an Object Type Body: Example

The following statement removes object type body `data_typ1`. See *Oracle Database PL/SQL Language Reference* for the example that creates this object type.

```
DROP TYPE BODY data_typ1;
```

DROP USER

Purpose

Use the `DROP USER` statement to remove a database user and optionally remove the user's objects.

In an Oracle Automatic Storage Management (Oracle ASM) cluster, a user authenticated `AS SYSASM` can use this clause to remove a user from the password file that is local to the Oracle ASM instance of the current node.

When you drop a user, Oracle Database also purges all of that user's schema objects from the recycle bin.

> **Note:**
>
> Do not attempt to drop the users `SYS` or `SYSTEM`. Doing so will corrupt your database.

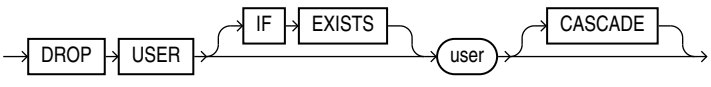
> **See Also:**
>
> CREATE USER and ALTER USER for information on creating and modifying a user

Prerequisites

You must have the `DROP USER` system privilege. In an Oracle ASM cluster, you must be authenticated `AS SYSASM`.

Syntax

***drop_user*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to drop an existing object.

Specifying `IF NOT EXISTS` with `DROP` results in `ORA-11544: Incorrect IF EXISTS clause for ALTER/DROP statement`.

user

Specify the user to be dropped. Oracle Database does not drop users whose schemas contain objects unless you specify `CASCADE` or unless you first explicitly drop the user's objects.

Restriction on Dropping Users

You cannot drop a user whose schema contains a table that uses a flashback data archive for historical tracking. You must first disable the table's use of the flashback data archive.

CASCADE

Specify `CASCADE` to drop all objects in the user's schema before dropping the user. You must specify this clause to drop a user whose schema contains any objects.

- If the user's schema contains tables, then Oracle Database drops the tables and automatically drops any referential integrity constraints on tables in other schemas that refer to primary and unique keys on these tables.
- If this clause results in tables being dropped, then the database also drops all domain indexes created on columns of those tables and invokes appropriate drop routines.

> **See Also:**
>
> *Oracle Database Data Cartridge Developer's Guide* for more information on these routines

- Oracle Database invalidates, but does not drop, the following objects in other schemas:
  - Views or synonyms for objects in the dropped user's schema
  - Stored procedures, functions, or packages that query objects in the dropped user's schema
- Oracle Database does not drop materialized views in other schemas that are based on tables in the dropped user's schema. However, because the base tables no longer exist, the materialized views in the other schemas can no longer be refreshed.
- Oracle Database drops all triggers in the user's schema.
- Oracle Database does not drop roles created by the user.
- Oracle Database drops all domains in the user's schemas. The database will issue `DROP DOMAIN FORCE` for all domains the user owns.

> **Note:**
>
> Oracle Database also drops with `FORCE` all types owned by the user. See the FORCE keyword of DROP TYPE .

Examples

Dropping a Database User: Example

If user Sidney's schema contains no objects, then you can drop `sidney` by issuing the statement:

```
DROP USER sidney;
```

If Sidney's schema contains objects, then you must use the `CASCADE` clause to drop `sidney` and the objects:

```
DROP USER sidney CASCADE;
```

DROP VIEW

Purpose

Use the `DROP VIEW` statement to remove a view or an object view from the database. You can change the definition of a view by dropping and re-creating it.

> **See Also:**
>
> CREATE VIEW and ALTER VIEW for information on creating and modifying a view

Prerequisites

The view must be in your own schema or you must have the `DROP ANY VIEW` system privilege.

Syntax

***drop_view*::=**

Semantics

IF EXISTS

Specify `IF EXISTS` to drop an existing object.

Specifying `IF NOT EXISTS` with `DROP` results in `ORA-11544: Incorrect IF EXISTS clause for ALTER/DROP statement.`

schema

Specify the schema containing the view. If you omit *schema*, then Oracle Database assumes the view is in your own schema.

view

Specify the name of the view to be dropped.

Oracle Database does not drop views, materialized views, and synonyms that are dependent on the view but marks them INVALID. You can drop them or redefine views and synonyms, or you can define other views in such a way that the invalid views and synonyms become valid again.

If any subviews have been defined on *view*, then the database invalidates the subviews as well. To determine whether the view has any subviews, query the SUPERVIEW_NAME column of the USER_, ALL_, or DBA_VIEWS data dictionary views.

> **See Also:**
>
> - CREATE TABLE and CREATE SYNONYM
> - ALTER MATERIALIZED VIEW for information on revalidating invalid materialized views

CASCADE CONSTRAINTS

Specify CASCADE CONSTRAINTS to drop all referential integrity constraints that refer to primary and unique keys in the view to be dropped. If you omit this clause, and such constraints exist, then the DROP statement fails.

Examples

Dropping a View: Example

The following statement drops the emp_view view, which was created in "Creating a View: Example":

```
DROP VIEW emp_view;
```

EXPLAIN PLAN

Purpose

Use the EXPLAIN PLAN statement to determine the execution plan Oracle Database follows to execute a specified SQL statement. This statement inserts a row describing each step of the execution plan into a specified table. You can also issue the EXPLAIN PLAN statement as part of the SQL trace facility.

This statement also determines the cost of executing the statement. If any domain indexes are defined on the table, then user-defined CPU and I/O costs will also be inserted.

The definition of a sample output table PLAN_TABLE is available in a SQL script on your distribution media. Your output table must have the same column names and data types as this

table. The common name of this script is `UTLXPLAN.SQL`. The exact name and location depend on your operating system.

Oracle Database provides information on cached cursors through several dynamic performance views:

- For information on the work areas used by SQL cursors, query `V$SQL_WORKAREA`.
- For information on the execution plan for a cached cursor, query `V$SQL_PLAN`.
- For execution statistics at each step or operation of an execution plan of cached cursors (for example, number of produced rows, number of blocks read), query `V$SQL_PLAN_STATISTICS`.
- For a selective precomputed join of the preceding three views, query `V$SQL_PLAN_STATISTICS_ALL`.
- Execution statistics at each step or operation of an execution plan of cached cursors are displayed in `V$SQL_PLAN_MONITOR` if the statement execution is monitored. You can force monitoring using the `MONITOR` hint.

See Also:

- *Oracle Database SQL Tuning Guide* for information on the output of `EXPLAIN PLAN`, how to use the SQL trace facility, and how to generate and interpret execution plans
- *Oracle Database Reference* for information on dynamic performance views

Prerequisites

To issue an `EXPLAIN PLAN` statement, you must have the privileges necessary to insert rows into an existing output table that you specify to hold the execution plan.

You must also have the privileges necessary to execute the SQL statement for which you are determining the execution plan. If the SQL statement accesses a view, then you must have privileges to access any tables and views on which the view is based. If the view is based on another view that is based on a table, then you must have privileges to access both the other view and its underlying table.

To examine the execution plan produced by an `EXPLAIN PLAN` statement, you must have the privileges necessary to query the output table.

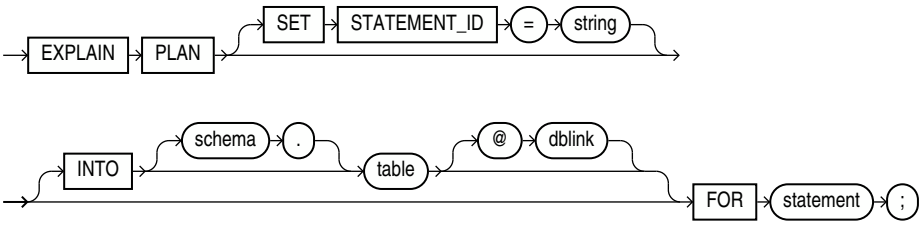
The `EXPLAIN PLAN` statement is a data manipulation language (DML) statement, rather than a data definition language (DDL) statement. Therefore, Oracle Database does not implicitly commit the changes made by an `EXPLAIN PLAN` statement. If you want to keep the rows generated by an `EXPLAIN PLAN` statement in the output table, then you must commit the transaction containing the statement.

See Also:

INSERT and SELECT for information on the privileges you need to populate and query the plan table

Syntax

***explain_plan*::=**



Semantics

SET STATEMENT_ID Clause

Specify a value for the `STATEMENT_ID` column for the rows of the execution plan in the output table. You can then use this value to identify these rows among others in the output table. Be sure to specify a `STATEMENT_ID` value if your output table contains rows from many execution plans. If you omit this clause, then the `STATEMENT_ID` value defaults to null.

INTO *table* Clause

Specify the name of the output table, and optionally its schema and database. This table must exist before you use the `EXPLAIN PLAN` statement.

If you omit *`schema`*, then the database assumes the table is in your own schema.

The *`dblink`* can be a complete or partial name of a database link to a remote Oracle Database where the output table is located. You can specify a remote output table only if you are using Oracle Database distributed functionality. If you omit *`dblink`*, then the database assumes the table is on your local database. See "References to Objects in Remote Databases " for information on referring to database links.

If you omit `INTO` altogether, then the database assumes an output table named `PLAN_TABLE` in your own schema on your local database.

FOR *statement* Clause

Specify a `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `MERGE`, `CREATE TABLE`, `CREATE INDEX`, or `ALTER INDEX ... REBUILD` statement for which the execution plan is generated.

Notes on EXPLAIN PLAN

The following notes apply to `EXPLAIN PLAN`:

- If *`statement`* includes the *`parallel_clause`*, then the resulting execution plan will indicate parallel execution. However, `EXPLAIN PLAN` actually inserts the statement into the plan table, so that the parallel DML statement you submit is no longer the first DML statement in the transaction. This violates the Oracle Database restriction of one parallel DML statement in a single transaction, and the statement will be executed serially. To maintain parallel execution of the statements, you must commit or roll back the `EXPLAIN PLAN` statement, and then submit the parallel DML statement.
- To determine the execution plan for an operation on a temporary table, `EXPLAIN PLAN` must be run from the same session, because the data in temporary tables is session specific.

Examples

EXPLAIN PLAN Examples

The following statement determines the execution plan and cost for an `UPDATE` statement and inserts rows describing the execution plan into the specified `plan_table` table with the `STATEMENT_ID` value of 'Raise in Tokyo':

```
EXPLAIN PLAN
    SET STATEMENT_ID = 'Raise in Tokyo'
    INTO plan_table
    FOR UPDATE employees
        SET salary = salary * 1.10
        WHERE department_id =
           (SELECT department_id FROM departments
               WHERE location_id = 1700);
```

The following `SELECT` statement queries the `plan_table` table and returns the execution plan and the cost:

```
SELECT id, LPAD(' ',2*(LEVEL-1))||operation operation, options,
       object_name, object_alias, position
    FROM plan_table
    START WITH id = 0 AND statement_id = 'Raise in Tokyo'
    CONNECT BY PRIOR id = parent_id AND statement_id = 'Raise in Tokyo'
    ORDER BY id;
```

The query returns this execution plan:

```
 ID OPERATION            OPTIONS              OBJECT_NAME          OBJECT_ALIAS          POSITION
--- -------------------- -------------------- -------------------- -------------------- --------
  0 UPDATE STATEMENT                                                                          4
  1   UPDATE                                 EMPLOYEES                                        1
  2     INDEX            RANGE SCAN           EMP_DEPARTMENT_IX    EMPLOYEES@UPD$1             1
  3       TABLE ACCESS   BY INDEX ROWID       DEPARTMENTS          DEPARTMENTS@SEL$1           1
  4         INDEX        RANGE SCAN           DEPT_LOCATION_IX     DEPARTMENTS@SEL$1           1
```

The value in the `POSITION` column of the first row shows that the statement has a cost of 4.

EXPLAIN PLAN: Partitioned Example

The sample table `sh.sales` is partitioned on the `time_id` column. Partition `sales_q3_2000` contains time values less than Oct. 1, 2000, and there is a local index `sales_time_bix` on the `time_id` column.

Consider the query:

```
EXPLAIN PLAN FOR
  SELECT * FROM sales
     WHERE time_id BETWEEN :h AND '01-OCT-2000';
```

where `:h` represents an already declared bind variable. `EXPLAIN PLAN` executes this query with `PLAN_TABLE` as the output table. The basic execution plan, including partitioning information, is obtained with the following query:

```
SELECT operation, options, partition_start, partition_stop,
       partition_id
  FROM plan_table;
```

FLASHBACK DATABASE

Purpose

Use the `FLASHBACK DATABASE` statement to return the database to a past time or system change number (SCN). This statement provides a fast alternative to performing incomplete database recovery.

Following a `FLASHBACK DATABASE` operation, in order to have write access to the flashed back database, you must reopen it with an `ALTER DATABASE OPEN RESETLOGS` statement.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information on `FLASHBACK DATABASE`

Prerequisites

You must have the `SYSDBA`, `SYSBACKUP`, or `SYSDG` system privilege.

If you are connected to a multitenant container database (CDB):

- To flash back a CDB, you must be connected to the root and you must have the `SYSDBA`, `SYSBACKUP`, or `SYSDG` system privilege granted commonly.
- To flash back a PDB you must be connected to the root and you must have the `SYSDBA`, `SYSBACKUP`, or `SYSDG` system privilege granted commonly, or you must be connected to the PDB you want to flash back and you must have the `SYSDBA`, `SYSBACKUP`, or `SYSDG` system privilege, granted commonly or granted locally in that PDB.

A fast recovery area must have been prepared for the database. The database must have been put in `FLASHBACK` mode with an `ALTER DATABASE FLASHBACK ON` statement unless you are flashing the database back to a guaranteed restore point. The database must be mounted but not open.

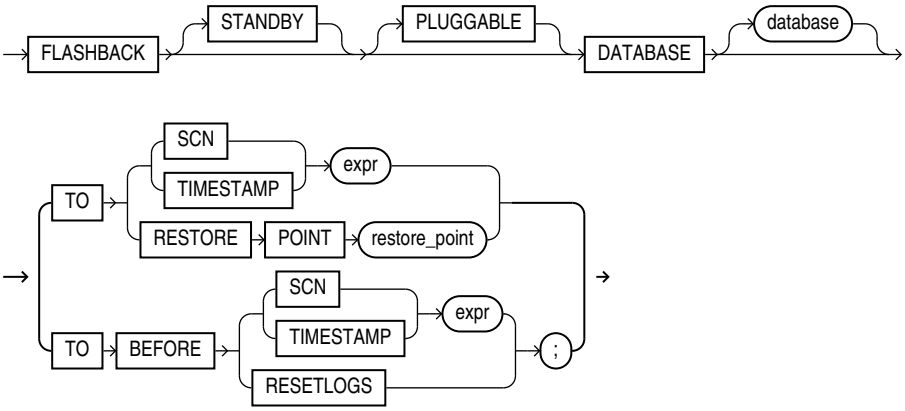
In addition:

- The database must run in `ARCHIVELOG` mode.
- The database must be mounted, but not open, with a current control file. The control file cannot be a backup or re-created. When the database control file is restored from backup or re-created, all existing flashback log information is discarded.
- The database must contain no online tablespaces for which flashback functionality was disabled with the SQL statement `ALTER TABLESPACE ... FLASHBACK OFF`.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* and the `ALTER DATABASE` ... *flashback_mode_clause* for information on putting the database in `FLASHBACK` mode
> - CREATE RESTORE POINT for information on restore points and guaranteed restore points

Syntax

***flashback_database*::=**



Semantics

When you issue a `FLASHBACK DATABASE` statement, Oracle Database first verifies that all required archived and online redo logs are available. If they are available, then it reverts all currently online data files in the database to the SCN or time specified in this statement.

- The amount of Flashback data retained in the database is controlled by the `DB_FLASHBACK_RETENTION_TARGET` initialization parameter and the size of the fast recovery area. You can determine how far back you can flash back the database by querying the `V$FLASHBACK_DATABASE_LOG` view.
- If insufficient data remains in the database to perform the flashback, then you can use standard recovery procedures to recover the database to a past point in time.
- If insufficient data remains for a set of data files, then the database returns an error. In this case, you can take those data files offline and reissue the statement to revert the remainder of the database. You can then attempt to recover the offline data files using standard recovery procedures.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information on recovering data files

STANDBY

Specify `STANDBY` to revert the standby database to an earlier SCN or time. If the database is not a standby database, then the database returns an error. If you omit this clause, then *`database`* can be either a primary or a standby database.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for information on how you can use `FLASHBACK DATABASE` on a standby database to achieve different delays

PLUGGABLE

Specify `PLUGGABLE` to flash back a PDB. You must specify this clause whether the current container is the root or the PDB you want to flash back.

Restrictions on Flashing Back a PDB

- You cannot flash back a proxy PDB.
- If the CDB is in shared undo mode, then you can only flash back a PDB to a clean PDB restore point. Refer to the CLEAN clause of `CREATE RESTORE POINT` for more information.

database

If you are flashing back a CDB, then you can optionally specify the name of the database to be flashed back. If you omit *`database`*, then Oracle Database flashes back the database identified by the value of the initialization parameter `DB_NAME`.

If you are flashing back a PDB and the current container is the root, then use *`database`* to specify the name of the PDB to be flashed back. If you are flashing back a PDB and the current container is that PDB, then you can optionally use *`database`* to specify the PDB name.

TO SCN Clause

Specify a system change number (SCN):

- `TO SCN` reverts the database back to its state at the specified SCN.
- `TO BEFORE SCN` reverts the database back to its state at the system change number just preceding the specified SCN.

You can determine the current SCN by querying the `CURRENT_SCN` column of the `V$DATABASE` view. This in turn lets you save the SCN to a spool file, for example, before running a high-risk batch job.

TO TIMESTAMP Clause

Specify a valid datetime expression.

- `TO TIMESTAMP` reverts the database back to its state at the specified timestamp.
- `TO BEFORE TIMESTAMP` reverts the database back to its state one second before the specified timestamp.

You can represent the timestamp as an offset from a determinate value, such as `SYSDATE`, or as an absolute system timestamp.

TO RESTORE POINT Clause

Specify this clause to flash back the database to the specified restore point. If you have not enabled flashback database, then this is the only clause you can specify in this `FLASHBACK DATABASE` statement. If the database is not in `FLASHBACK` mode, as described in the "Prerequisites" section above, then this is the only clause you can specify for this statement.

RESETLOGS

Specify `TO BEFORE RESETLOGS` to flash the database back to just before the last resetlogs operation (`ALTER DATABASE OPEN RESETLOGS`).

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information about this clause

Examples

Assuming that you have prepared a fast recovery area for the database and enabled media recovery, enable database `FLASHBACK` mode and open the database with the following statements:

```
STARTUP MOUNT
ALTER DATABASE FLASHBACK ON;
ALTER DATABASE OPEN;
```

With your database open for at least a day, you can flash back the database one day with the following statements:

```
SHUTDOWN DATABASE
STARTUP MOUNT
FLASHBACK DATABASE TO TIMESTAMP SYSDATE-1;
```

FLASHBACK TABLE

Purpose

Use the `FLASHBACK TABLE` statement to restore an earlier state of a table in the event of human or application error. The time in the past to which the table can be flashed back is dependent on the amount of undo data in the system. Also, Oracle Database cannot restore a table to an earlier state across any DDL operations that change the structure of the table.

> **Note:**
>
> Oracle strongly recommends that you run your database in automatic undo mode by leaving the `UNDO_MANAGEMENT` initialization parameter set to `AUTO`, which is the default. In addition, set the `UNDO_RETENTION` initialization parameter to an interval large enough to include the oldest data you anticipate needing. For more information refer to the documentation on the `UNDO_MANAGEMENT` and `UNDO_RETENTION` initialization parameters.

You cannot roll back a FLASHBACK TABLE statement. However, you can issue another FLASHBACK TABLE statement and specify a time just prior to the current time. Therefore, it is advisable to record the current SCN before issuing a FLASHBACK TABLE clause.

> **See Also:**
>
> - To set the UNDO_RETENTION initialization parameter, see Setting the Minimum Undo Retention Period
> - FLASHBACK DATABASE for information on reverting the entire database to an earlier version
> - the *flashback_query_clause* of SELECT for information on retrieving past data from a table
> - *Oracle Database Backup and Recovery User's Guide* for additional information on using the FLASHBACK TABLE statement

Prerequisites

To flash back a table to an earlier SCN or timestamp, you must have either the FLASHBACK object privilege on the table or the FLASHBACK ANY TABLE system privilege. In addition, you must have the READ or SELECT object privilege on the table, and you must have the INSERT, DELETE, and ALTER object privileges on the table.

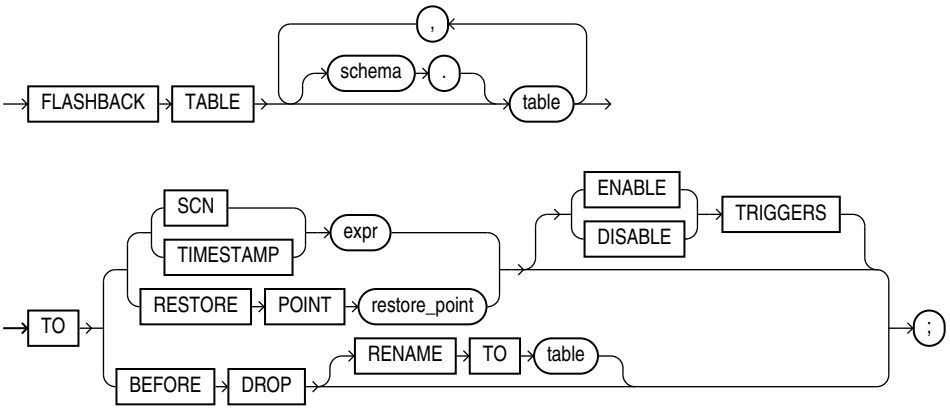
Row movement must be enabled for all tables in the Flashback list unless you are flashing back the table TO BEFORE DROP. That operation is called a **flashback drop** operation, and it uses dropped data in the recycle bin rather than undo data. Refer to *row_movement_clause* for information on enabling row movement.

To flash back a table to a restore point, you must have the SELECT ANY DICTIONARY or FLASHBACK ANY TABLE system privilege or the SELECT_CATALOG_ROLE role.

To flash back a table to before a DROP TABLE operation, you need only the privileges necessary to drop the table.

Syntax

***flashback_table*::=**



Semantics

During an Oracle Flashback Table operation, Oracle Database acquires exclusive DML locks on all the tables specified in the Flashback list. These locks prevent any operations on the tables while they are reverting to their earlier state.

The Flashback Table operation is executed in a single transaction, regardless of the number of tables specified in the Flashback list. Either all of the tables revert to the earlier state or none of them do. If the Flashback Table operation fails on any table, then the entire statement fails.

At the completion of the Flashback Table operation, the data in *table* is consistent with *table* at the earlier time. However, `FLASHBACK TABLE TO SCN` or `TIMESTAMP` does not preserve rowids, and `FLASHBACK TABLE TO BEFORE DROP` does not recover referential constraints.

Oracle Database does not revert statistics associated with *table* to their earlier form. Indexes on *table* that exist currently are reverted and reflect the state of the table at the Flashback point. If the index exists now but did not yet exist at the Flashback point, then the database updates the index to reflect the state of the table at the Flashback point. However, indexes that were dropped during the interval between the Flashback point and the current time are not restored.

schema

Specify the schema containing the table. If you omit *schema*, then the database assumes the table is in your own schema.

table

Specify the name of one or more tables containing data you want to revert to an earlier version.

Restrictions on Flashing Back Tables

This statement is subject to the following restrictions:

- Flashback Table operations are not valid for the following type objects: tables that are part of a cluster, child tables using reference partitioning, materialized views, Advanced Queuing (AQ) tables, static data dictionary tables, system tables, remote tables, object tables, nested tables, or individual table partitions or subpartitions.
- The following DDL operations change the structure of a table, so that you cannot subsequently use the `TO SCN` or `TO TIMESTAMP` clause to flash the table back to a time preceding the operation: upgrading, moving, or truncating a table; adding a constraint to a table, adding a table to a cluster; modifying or dropping a column; changing a column encryption key; adding, dropping, merging, splitting, coalescing, or truncating a partition or subpartition (with the exception of adding a range partition).

TO SCN Clause

Specify the system change number (SCN) corresponding to the point in time to which you want to return the table. The *expr* must evaluate to a number representing a valid SCN.

TO TIMESTAMP Clause

Specify a timestamp value corresponding to the point in time to which you want to return the table. The *expr* must evaluate to a valid timestamp in the past. The table will be flashed back to a time within approximately 3 seconds of the specified timestamp.

TO RESTORE POINT Clause

Specify a restore point to which you want to flash back the table. The restore point must already have been created.

> **See Also:**
>
> CREATE RESTORE POINT for information on creating restore points

ENABLE | DISABLE TRIGGERS

By default, Oracle Database disables all enabled triggers defined on *table* during the Flashback Table operation and then reenables them after the Flashback Table operation is complete. Specify ENABLE TRIGGERS if you want to override this default behavior and keep the triggers enabled during the Flashback process.

This clause affects only those database triggers defined on *table* that are already enabled. To enable currently disabled triggers selectively, use the ALTER TABLE ... *enable_disable_clause* before you issue the FLASHBACK TABLE statement with the ENABLE TRIGGERS clause.

TO BEFORE DROP Clause

Use this clause to retrieve from the recycle bin a table that has been dropped, along with all possible dependent objects. The table must have resided in a locally managed tablespace other than the SYSTEM tablespace.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information on the recycle bin and naming conventions for objects in the recycle bin
> - PURGE for information on removing objects permanently from the recycle bin

You can specify either the original user-specified name of the table or the system-generated name Oracle Database assigned to the object when it was dropped.

- System-generated recycle bin object names are unique. Therefore, if you specify the system-generated name, then the database retrieves that specified object.

  To see the contents of your recycle bin, query the USER_RECYCLEBIN data dictionary view. You can use the RECYCLEBIN synonym instead. The following two statements return the same rows:

  ```
  SELECT * FROM RECYCLEBIN;
  SELECT * FROM USER_RECYCLEBIN;
  ```

- If you specify the user-specified name, and if the recycle bin contains more than one object of that name, then the database retrieves the object that was moved to the recycle bin most recently. If you want to retrieve an older version of the table, then do one of these things:
  - Specify the system-generated recycle bin name of the table you want to retrieve.

  - Issue additional `FLASHBACK TABLE ... TO BEFORE DROP` statements until you retrieve the table you want.

Oracle Database attempts to preserve the original table name. If a new table of the same name has been created in the same schema since the original table was dropped, then the database returns an error unless you also specify the `RENAME TO` clause.

RENAME TO Clause

Use this clause to specify a new name for the table being retrieved from the recycle bin.

Notes on Flashing Back Dropped Tables

The following notes apply to flashing back dropped tables:

- Oracle Database retrieves all indexes defined on the table retrieved from the recycle bin except for bitmap join indexes and domain indexes. (Bitmap join indexes and domain indexes are not put in the recycle bin during a `DROP TABLE` operation, so cannot be retrieved.)
- The database also retrieves all triggers and constraints defined on the table except for referential integrity constraints that reference other tables.

  The retrieved indexes, triggers, and constraints have recycle bin names. Therefore it is advisable to query the `USER_RECYCLEBIN` view before issuing a `FLASHBACK TABLE ... TO BEFORE DROP` statement so that you can rename the retrieved triggers and constraints to more usable names.
- When you drop a table, all materialized view logs defined on the table are also dropped but are not placed in the recycle bin. Therefore, the materialized view logs cannot be flashed back along with the table.
- When you drop a table, any indexes on the table are dropped and put into the recycle bin along with the table. If subsequent space pressures arise, then the database reclaims space from the recycle bin by first purging indexes. In this case, when you flash back the table, you may not get back all of the indexes that were defined on the table.
- You cannot flash back a table if it has been purged, either by a user or by Oracle Database as a result of some space reclamation operation.

Examples

Restoring a Table to an Earlier State: Examples

The examples below create a new table, `employees_test`, with row movement enabled, update values within the new table, and issue the `FLASHBACK TABLE` statement.

Create table `employees_test`, with row movement enabled, from table `employees` of the sample `hr` schema:

```
CREATE TABLE employees_test
  AS SELECT * FROM employees;
```

As a benchmark, list those salaries less than 2500:

```
SELECT salary
  FROM employees_test
  WHERE salary < 2500;

    SALARY
----------
      2400
      2200
```

```
      2100
      2400
      2200
```

> **Note:**
>
> To allow time for the SCN to propagate to the mapping table used by the `FLASHBACK TABLE` statement, wait a minimum of 5 minutes prior to issuing the following statement. This wait would not be necessary if a previously existing table were being used in this example.

Enable row movement for the table:

```
ALTER TABLE employees_test
   ENABLE ROW MOVEMENT;
```

Issue a 10% salary increase to those employees earning less than 2500:

```
UPDATE employees_test
  SET salary = salary * 1.1
  WHERE salary < 2500;

5 rows updated.
COMMIT;
```

As a second benchmark, list those salaries that remain less than 2500 following the 10% increase:

```
SELECT salary
  FROM employees_test
  WHERE salary < 2500;

    SALARY
----------
      2420
      2310
      2420
```

Restore the table `employees_test` to its state prior to the current system time. The unrealistic duration of 1 minute is used so that you can test this series of examples quickly. Under normal circumstances a much greater interval would have elapsed.

```
FLASHBACK TABLE employees_test
  TO TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' minute);
```

List those salaries less than 2500. After the `FLASHBACK TABLE` statement issued above, this list should match the list in the first benchmark.

```
SELECT salary
  FROM employees_test
  WHERE salary < 2500;

    SALARY
----------
      2400
      2200
      2100
```

```
      2400
      2200
```

Retrieving a Dropped Table: Example

If you accidentally drop the `pm.print_media` table and want to retrieve it, then issue the following statement:

```
FLASHBACK TABLE print_media TO BEFORE DROP;
```

If another `print_media` table has been created in the `pm` schema, then use the `RENAME TO` clause to rename the retrieved table:

```
FLASHBACK TABLE print_media TO BEFORE DROP RENAME TO print_media_old;
```

If you know that the employees table has been dropped multiple times, and you want to retrieve the oldest version, then query the `USER_RECYLEBIN` table to determine the system-generated name, and then use that name in the `FLASHBACK TABLE` statement. (System-generated names in your database will differ from those shown here.)

```
SELECT object_name, droptime FROM user_recyclebin
   WHERE original_name = 'PRINT_MEDIA';

OBJECT_NAME                    DROPTIME
------------------------------ -------------------
RB$$45703$TABLE$0              2003-06-03:15:26:39
RB$$45704$TABLE$0              2003-06-12:12:27:27
RB$$45705$TABLE$0              2003-07-08:09:28:01
```

GRANT

Purpose

Use the `GRANT` statement to grant:

- Administrative privileges to users only (not to roles). Table 18-1

  Refer to the *Database Security Guide* for more information about administrative privileges *Managing Administrative Privileges*
- System privileges to users and roles. Table 18-2

  Note that `ANY` system privileges, for example, `SELECT ANY TABLE`, will not work on `SYS` objects or other dictionary objects.
- Schema privileges to users and roles. Table 18-3
- Roles to users, roles, and program units. The granted roles can be either user-defined (local or external) or predefined. For a list of predefined roles, refer to *Oracle Database Security Guide*.
- Object privileges for a particular object to users and roles. Table 18-4

> **Note:**
>
> Global roles (created with `IDENTIFIED GLOBALLY`) are granted through enterprise roles and cannot be granted using the `GRANT` statement.

Notes on Authorizing Database Users

You can authorize database users through means other than the database and the GRANT statement.

- Many Oracle Database privileges are granted through supplied PL/SQL and Java packages. For information on those privileges, refer to the documentation for the appropriate package.
- Some operating systems have facilities that let you grant roles to Oracle Database users with the initialization parameter OS_ROLES. If you choose to grant roles to users through operating system facilities, then you cannot also grant roles to users with the GRANT statement, although you can use the GRANT statement to grant system privileges to users and system privileges and roles to other roles.

Note on Oracle Automatic Storage Management

A user authenticated AS SYSASM can use this statement to grant the administrative privileges SYSASM, SYSOPER, and SYSDBA to a user in the Oracle ASM password file of the current node.

Note on Editionable Objects

A GRANT operation to grant object privileges on an editionable object actualizes the object in the current edition. See *Oracle Database Development Guide* for more information about editions and editionable objects.

> **See Also:**
>
> - CREATE USER and CREATE ROLE for definitions of local, global, and external privileges
> - *Oracle Database Security Guide* for information about other authorization methods and for information about privileges
> - REVOKE for information on revoking grants

Prerequisites

To grant a **system privilege**, one of the following conditions must be met:

- You must have been granted the GRANT ANY PRIVILEGE system privilege. In this case, if you grant the system privilege to a role, then a user to whom the role has been granted does not have the privilege unless the role is enabled in user's session.
- You must have been granted the system privilege with the ADMIN OPTION. In this case, if you grant the system privilege to a role, then a user to whom the role has been granted has the privilege regardless whether the role is enabled in the user's session.

To grant a **role to a user or another role**, you must have been directly granted the role with the ADMIN OPTION, or you must have been granted the GRANT ANY ROLE system privilege, or you must have created the role.

To grant a **role to a program unit in your own schema**, you must have been directly granted the role with either the ADMIN OPTION or the DELEGATE OPTION, or you must have been granted the GRANT ANY ROLE system privilege, or you must have created the role.

To grant a **role to a program unit in another user's schema**, you must be the user SYS and the role must have been created by the schema owner or directly granted to the schema owner.

To grant an **object privilege on a user**, by specifying the ON USER clause of the *on_object_clause*, you must be the user on whom the privilege is granted, or you must have been granted the object privilege on that user with the WITH GRANT OPTION, or you must have been granted the GRANT ANY OBJECT PRIVILEGE system privilege. If you can grant an object privilege on a user only because you have the GRANT ANY OBJECT PRIVILEGE, then the GRANTOR column of the *_TAB_PRIVS views displays the user on whom the privilege is granted rather than the user who issued the GRANT statement.

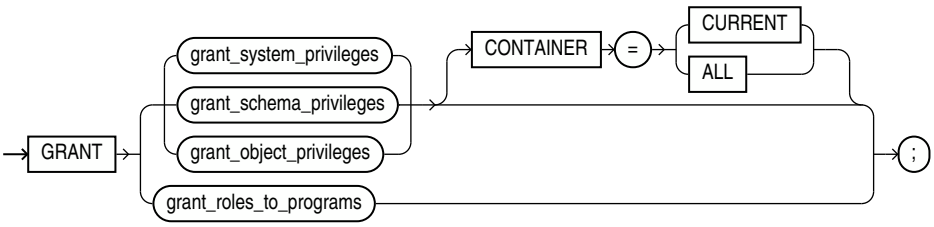
To grant an **object privilege on all other types of objects**, you must own the object, or the owner of the object must have granted you the object privileges with the WITH GRANT OPTION, or you must have been granted the GRANT ANY OBJECT PRIVILEGE system privilege. If you have the GRANT ANY OBJECT PRIVILEGE, then you can grant the object privilege only if the object owner could have granted the same object privilege. In this case, the GRANTOR column of the *_TAB_PRIVS views displays the object owner rather than the user who issued the GRANT statement.

You can revoke privileges on an object if you have the GRANT ANY object privilege. This does not apply to dictionary protected schemas. The ANY keyword in reference to a system privilege means that the user can perform the privilege on any objects owned by any user except for SYS.

To specify the CONTAINER clause, you must be connected to a multitenant container database (CDB). To specify CONTAINER = ALL, the current container must be the root.

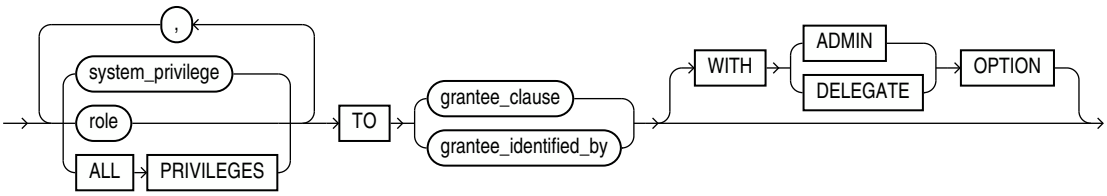
Syntax

***grant*::=**



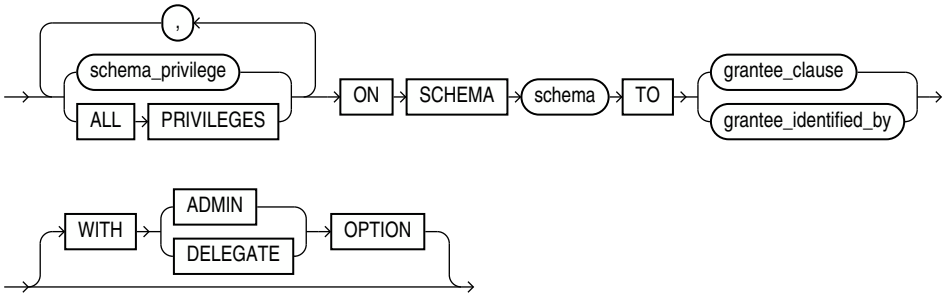
(*grant_system_privileges*::=, *grant_schema_privileges*::=,*grant_object_privileges*::=, *grant_roles_to_programs*::=)

***grant_system_privileges*::=**

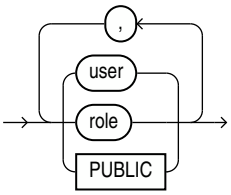


(*grantee_clause*::=, *grantee_identified_by*::=)

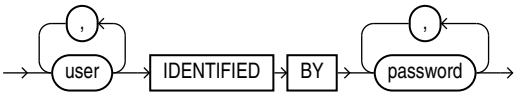
***grant_schema_privileges*::=**



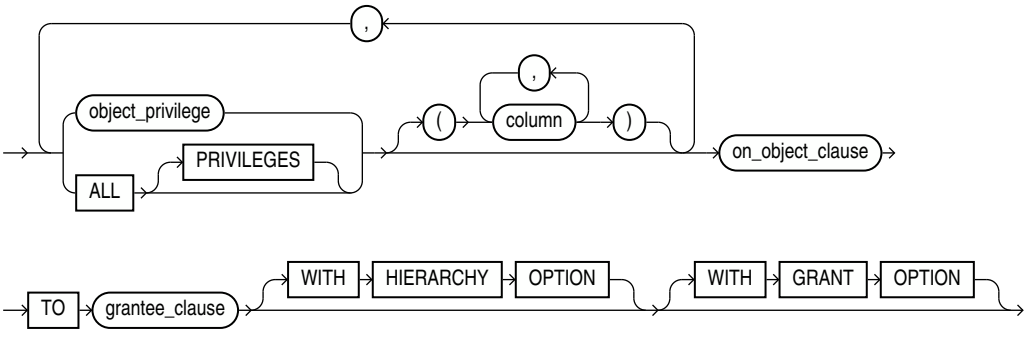
***grantee_clause*::=**



***grantee_identified_by*::=**

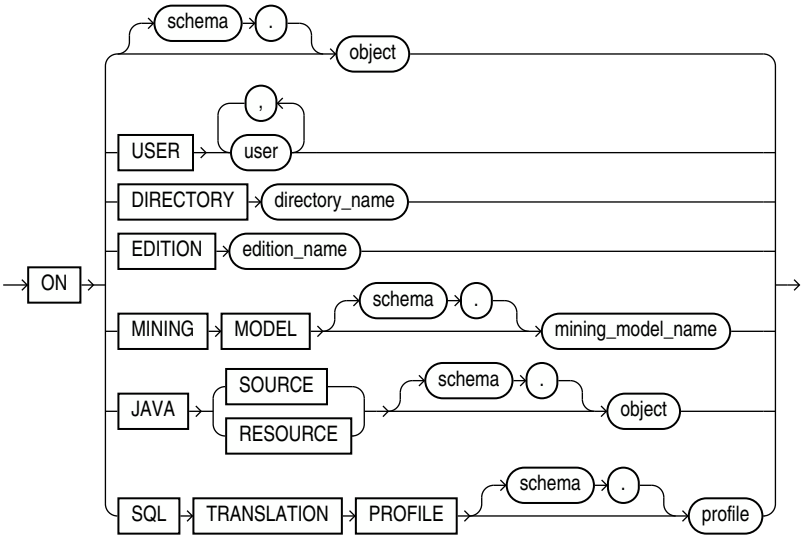


***grant_object_privileges*::=**

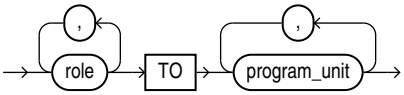


(*on_object_clause*::=, *grantee_clause*::=)

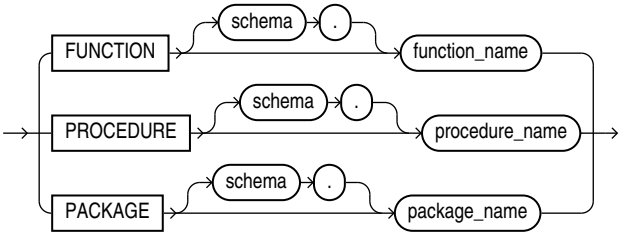
***on_object_clause*::=**



***grant_roles_to_programs*::=**



***program_unit*::=**



Semantics

grant_system_privileges

Use these clauses to grant system privileges.

system_privilege

Specify the system privilege you want to grant. Table 18-2 lists the system privileges, organized by the database object operated upon.

- If you grant a privilege to a **user**, then the database adds the privilege to the user's privilege domain. The user can immediately exercise the privilege. Oracle recommends that you only grant the `ANY` privileges to trusted users.

- If you grant a privilege to a **role**, then the database adds the privilege to the privilege domain of the role. Users who have been granted and have enabled the role can immediately exercise the privilege. Other users who have been granted the role can enable the role and exercise the privilege.

  > **See Also:**
  >
  > Granting a System Privilege to a User: Example and "Granting System Privileges to a Role: Example"

- If you grant a privilege to **PUBLIC**, then the database adds the privilege to the privilege domains of each user. All users can immediately perform operations authorized by the privilege. Oracle recommends against granting system privileges to PUBLIC.

role

Specify the role you want to grant. You can grant an Oracle Database predefined role or a user-defined role.

- If you grant a role to a **user**, then the database makes the role available to the user. The user can immediately enable the role and exercise the privileges in the privilege domain of the role.

  In the case of a secure application role, you need not grant such a role directly to the user. You can let the associated PL/SQL package do this, assuming the user passes appropriate security policies. For more information, see the CREATE ROLE semantics for USING *package* and *Oracle Database Security Guide*

- If you grant a role to another **role**, then the database adds the privilege domain of the granted role to the privilege domain of the grantee role. Users who have been granted the grantee role can enable it and exercise the privileges in the granted role's privilege domain.
- If you grant a role to **PUBLIC**, then the database makes the role available to all users. All users can immediately enable the role and exercise the privileges in the privilege domain of the role.

> **Note:**
>
> Unlimited tablespace is not granted to the DBA role, but when a grant is executed to grant DBA to a user, unlimited tablespace is granted as part of the same grant, provided the user executing the GRANT command has unlimited tablespace granted with the ADMIN or GRANT ANY PRIVILEGE system privilege .

ALL PRIVILEGES

Specify ALL PRIVILEGES to grant all of the system privileges listed in Table 18-2, except the SELECT ANY DICTIONARY, ALTER DATABASE LINK, and ALTER PUBLIC DATABASE LINK privileges.

However, grant and revoke ALL PRIVILEGES do not apply to ADMINISTER KEY MANAGEMENT. Granting ALL PRIVILEGES does not grant ADMINISTER KEY MANAGEMENT. Similarly, revoking ALL PRIVILEGES does not revoke ADMINISTER KEY MANAGEMENT.

> **See Also:**
>
> - *Oracle Database Security Guide* for information on the Oracle predefined roles
> - "Granting a Role to a Role: Example"
> - CREATE ROLE for information on creating a user-defined role

grantee_clause

Use the *grantee_clause* to specify the users or roles to which the system privilege, role, or object privilege is granted.

PUBLIC

Specify PUBLIC to grant the privileges to all users. Oracle recommends against granting system privileges to PUBLIC.

Restriction on Grantees

A user, role, or PUBLIC cannot appear more than once in the *grantee_clause*.

grantee_identified_by

The *grantee_identified_by* clause lets you assign passwords to users when granting them system privileges and roles. You must specify an equal number of users and passwords. The first password is assigned to the first user, the second password is assigned to the second user, and so on. If a specified user exists, then the database resets the user's password. If a specified user does not exist, then the database creates the user with the password.

You can set the password to a maximum length of 1024 bytes.

> **See Also:**
>
> CREATE USER for restrictions on usernames and passwords and "Assigning User Passwords When Granting a System Privilege: Example"

WITH ADMIN OPTION

Specify WITH ADMIN OPTION to enable the grantee to:

- Grant the privilege or role to another user or role, unless the role is a GLOBAL role
- Revoke the privilege or role from another user or role
- Alter the privilege or role to change the authorization needed to access it
- Drop the privilege or role
- Grant the role to a program unit in the grantee's schema.
- Revoke the role from a program unit in the grantee's schema.

If you grant a system privilege or role to a user without specifying WITH ADMIN OPTION, and then subsequently grant the privilege or role to the user WITH ADMIN OPTION, then the user has the ADMIN OPTION on the privilege or role.

To revoke the ADMIN OPTION on a system privilege or role from a user, you must revoke the privilege or role from the user altogether and then grant the privilege or role to the user without the ADMIN OPTION.

> **See Also:**
>
> "Granting a Role with the ADMIN OPTION: Example"

WITH DELEGATE OPTION

You can specify this clause only when granting a role to a user.

Specify WITH DELEGATE OPTION to enable the grantee to:

- Grant the role to a program unit in the grantee's schema
- Revoke the role from a program unit in the grantee's schema

If you grant a role to a user without specifying WITH DELEGATE OPTION, and then subsequently grant the role to the user WITH DELEGATE OPTION, then the user has the DELEGATE OPTION on the role.

To revoke the DELEGATE OPTION on a role from a user, you must revoke the role from the user altogether and then grant the role to the user without the DELEGATE OPTION.

> **See Also:**
>
> - "Granting a Role with the DELEGATE OPTION: Example"
> - The *grant_roles_to_programs* clause for more information on granting roles to program units

Restrictions on Granting System Privileges and Roles

Privileges and roles are subject to the following restrictions:

- A privilege or role cannot appear more than once in the list of privileges and roles to be granted.
- You cannot grant a role to itself.
- You cannot grant a role IDENTIFIED GLOBALLY to anything.
- You cannot grant a role IDENTIFIED EXTERNALLY to a global user or global role.
- You cannot grant roles circularly. For example, if you grant the role banker to the role teller, then you cannot subsequently grant teller to banker.
- You cannot grant an IDENTIFIED BY role, IDENTIFIED USING role, or IDENTIFIED EXTERNALLY role to another role.

grant_schema_privileges

You can grant a schema level privilege to any user or any role if you are a user who :

- Owns the schema.

- Has a schema level privilege `WITH ADMIN OPTION`.
- Has the `GRANT ANY SCHEMA PRIVILEGE` system privilege. If you specify `ALL PRIVILEGES`, all the schema level privileges in Table 18-3 are granted.

You cannot grant schema privileges on the `SYS` schema.

> **See Also:**
>
> *Schema Privileges to Simplify Access Control in the Oracle Database Security Guide*

grant_object_privileges

Use these clauses to grant object privileges.

object_privilege

Specify the object privilege you want to grant. Table 18-4 lists the object privileges, organized by the type of object on which they can be granted. When you grant an object privilege on a editionable object, either to a user or to a role, the object is actualized in the edition in which the grant is made. Refer to CREATE EDITION for information on editionable object types and editions.

> **Note:**
>
> To grant `SELECT` on a view to another user, either you must own all of the objects underlying the view or you must have been granted the `SELECT` object privilege `WITH GRANT OPTION` on all of those underlying objects. This is true even if the grantee already has `SELECT` privileges on those underlying objects.
>
> To grant `READ` on a view to another user, either you must own all of the objects underlying the view or you must have been granted the `READ` or `SELECT` object privilege `WITH GRANT OPTION` on all of those underlying objects. This is true even if the grantee already has the `READ` or `SELECT` privilege on those underlying objects.

Restriction on Object Privileges

A privilege cannot appear more than once in the list of privileges to be granted.

ALL [PRIVILEGES]

Specify `ALL` to grant all the privileges for the object that you have been granted with the `GRANT OPTION`. The user who owns the schema containing an object automatically has all privileges on the object with the `GRANT OPTION`. The keyword `PRIVILEGES` is provided for semantic clarity and is optional.

column

Specify the table or view column on which privileges are to be granted. You can specify columns only when granting the `INSERT`, `REFERENCES`, or `UPDATE` privilege. If you do not list columns, then the grantee has the specified privilege on all columns in the table or view.

For information on existing column object grants, query the `USER_`, `ALL_`, or `DBA_COL_PRIVS` data dictionary view.

> **See Also:**
>
> *Oracle Database Reference* for information on the data dictionary views and "Granting Multiple Object Privileges on Individual Columns: Example"

on_object_clause

The *on_object_clause* identifies the object on which the privileges are granted. Users, directory objects, editions, data mining models, Java source and resource schema objects, and SQL translation profiles are identified separately because they reside in separate namespaces.

> **See Also:**
>
> "Granting Object Privileges to a Role: Example"

object

Specify the schema object on which the privileges are to be granted. If you do not qualify *object* with *schema*, then the database assumes the object is in your own schema. The object can be one of the following types:

- Table, view, or materialized view
- Sequence
- Procedure, function, or package
- User-defined type
- Synonym for any of the preceding items
- Directory, library, operator, or indextype
- Java source, class, or resource

You cannot grant privileges directly to a single partition of a partitioned table.

> **See Also:**
>
> "Granting Object Privileges on a Table to a User: Example", "Granting Object Privileges on a View: Example", and "Granting Object Privileges to a Sequence in Another Schema: Example"

ON USER

Specify the database user you want to grant privileges to.

Restriction on Granting Privileges on Users

You cannot grant privileges on user `PUBLIC`.

> **See Also:**
>
> "Granting an Object Privilege on a User: Example"

ON DIRECTORY

Specify the name of the directory object on which privileges are to be granted. You cannot qualify *directory_name* with a schema name.

> **See Also:**
>
> CREATE DIRECTORY and "Granting an Object Privilege on a Directory: Example"

ON EDITION

Specify the name of the edition on which the `USE` object privilege is to be granted. You cannot qualify *edition_name* with a schema name.

ON MINING MODEL

Specify the name of the mining model on which privileges are to be granted. If you do not qualify *mining_model_name* with *schema*, then the database assumes that the mining model is in your own schema.

ON JAVA SOURCE | RESOURCE

Specify the name of the Java source or resource schema object on which privileges are to be granted. If you do not qualify *object* with *schema*, then the database assumes that the object is in your own schema.

> **See Also:**
>
> CREATE JAVA

ON SQL TRANSLATION PROFILE

Specify the name of the SQL translation profile on which privileges are to be granted. If you do not qualify *profile* with *schema*, then the database assumes that the profile is in your own schema.

WITH HIERARCHY OPTION

Specify `WITH HIERARCHY OPTION` to grant the specified object privilege on all subobjects of *object*, such as subviews created under a view, including subobjects created subsequent to this statement.

This clause is meaningful only in combination with the `READ` or `SELECT` object privilege.

WITH GRANT OPTION

Specify `WITH GRANT OPTION` to enable the grantee to grant the object privileges to other users and roles.

If you grant an object privilege to a user without specifying `WITH GRANT OPTION`, and then subsequently grant the privilege to the user `WITH GRANT OPTION`, then the user has the `GRANT OPTION` on the privilege.

To revoke the `GRANT OPTION` on an object privilege from a user, you must revoke the privilege from the user altogether and then grant the privilege to the user without the `GRANT OPTION`.

Restriction on Granting WITH GRANT OPTION

You can specify `WITH GRANT OPTION` only when granting to a user or to `PUBLIC`, not when granting to a role.

grant_roles_to_programs

Use this clause to grant roles to program units. Such roles are called code based access control (CBAC) roles.

role

Specify the role you want to grant. You can grant an Oracle Database predefined role or a user-defined role. The role must have been created by or directly granted to the schema owner of the program unit.

program_unit

Specify the program unit to which the role is to be granted. You can specify a PL/SQL function, procedure, or package. If you do not specify *`schema`*, then Oracle Database assumes the function, procedure, or package is in your own schema.

> **See Also:**
>
> *Oracle Database Security Guide* for more information on granting code based access control roles to program units

CONTAINER Clause

If the current container is a pluggable database (PDB):

- Specify `CONTAINER = CURRENT` to locally grant a system privilege, object privilege, or role to a user or role. The privilege or role is granted to the user or role only in the current PDB.

If the current container is the root:

- Specify `CONTAINER = CURRENT` to locally grant a system privilege, object privilege, or role to a common user or common role. The privilege or role is granted to the user or role only in the root.
- Specify `CONTAINER = ALL` to commonly grant a system privilege, object privilege on a common object, or role, to a common user or common role.

If you omit this clause, then `CONTAINER = CURRENT` is the default.

> **Note:**
>
> If you specify the CONTAINER clause when granting a privilege or role, then the current container must be the same container and you must specify the same CONTAINER clause when you revoke the privilege or role. Refer to the CONTAINER Clause of the REVOKE statement for more information.

Listings of System Administrative, System, Schema, and Object Privileges

Table 18-1 Administrative Privileges

Administrative Privilege Name	Operations Authorized
SYSBACKUP	Perform the following backup and recovery operations: STARTUP and SHUTDOWN. CREATE CONTROLFILE. CREATE PFILE and CREATE SPFILE. FLASHBACK DATABASE. Create, use, view, and drop restore points (including guaranteed restore points). Execute procedures in the DBMS_DATAPUMP, DBMS_PIPE, DBMS_TDB, and DBMS_TTS packages. SELECT on X$ tables, V$ views, and GV$ views. Includes the ALTER DATABASE, ALTER SESSION, ALTER SYSTEM, ALTER TABLESPACE, CREATE ANY CLUSTER, CREATE ANY DIRECTORY, CREATE ANY TABLE, CREATE SESSION, DROP DATABASE, DROP TABLESPACE, RESUMABLE, SELECT ANY DICTIONARY, SELECT ANY TRANSACTION, UNLIMITED TABLESPACE privileges and the SELECT_CATALOG_ROLE role.
SYSDBA	This is the most powerful administrative privilege. It allows most operations including the ability to view user data. STARTUP and SHUTDOWN. ALTER DATABASE: open, mount, back up, or change character set. CREATE DATABASE. DROP DATABASE. ARCHIVELOG and RECOVERY. CREATE SPFILE. Includes the RESTRICTED SESSION privilege.
SYSDG	Perform the following Oracle Data Guard operations: STARTUP and SHUTDOWN. FLASHBACK DATABASE. Create, use, view, and drop restore points (including guaranteed restore points). SELECT on X$ tables, V$ views, and GV$ views. Includes the ALTER DATABASE, ALTER SESSION, ALTER SYSTEM, CREATE SESSION, and SELECT ANY DICTIONARY privileges.

Table 18-1 (Cont.) Administrative Privileges

Administrative Privilege Name	Operations Authorized
SYSKM	Perform the following encryption key management operations: Connect to the database even if the database is not open. SELECT on the following views when the database is open: V$CLIENT_SECRETS, V$ENCRYPTED_TABLESPACES, V$ENCRYPTION_KEYS, V$ENCRYPTION_WALLET and V$WALLET. Includes the ADMINISTER KEY MANAGEMENT and CREATE SESSION privileges.
SYSOPER	STARTUP and SHUTDOWN operations. ALTER DATABASE: open, mount, or back up. ARCHIVELOG and RECOVERY. CREATE SPFILE. Includes the RESTRICTED SESSION privilege.

Table 18-2 System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
Advisor Framework Privileges:	All of the advisor framework privileges are part of the DBA role.
ADVISOR	Access the advisor framework through PL/SQL packages such as DBMS_ADVISOR and DBMS_SQLTUNE.
ADMINISTER SQL TUNING SET	Create, drop, select (read), load (write), and delete SQL tuning sets owned by the grantee through the DBMS_SQLTUNE package.
ADMINISTER ANY SQL TUNING SET	Create, drop, select (read), load (write), and delete SQL tuning sets owned by any user through the DBMS_SQLTUNE package.
CREATE ANY SQL PROFILE	Accept a SQL Profile recommended by the SQL Tuning Advisor, which is accessed through Enterprise Manager or by the DBMS_SQLTUNE package. **Note**: This privilege has been deprecated in favor of ADMINISTER SQL MANAGEMENT OBJECT.
ALTER ANY SQL PROFILE	Alter the attributes of an existing SQL Profile. **Note**: This privilege has been deprecated in favor of ADMINISTER SQL MANAGEMENT OBJECT.
DROP ANY SQL PROFILE	Drop existing SQL Profiles. **Note**: This privilege has been deprecated in favor of ADMINISTER SQL MANAGEMENT OBJECT.
ADMINISTER SQL MANAGEMENT OBJECT	Create, alter, and drop SQL Profiles owned by any user through the DBMS_SQLTUNE package.
ANALYTIC VIEWS	—
CREATE ANALYTIC VIEW	Create analytic views in the grantee's schema.
CREATE ANY ANALYTIC VIEW	Create analytic views in any schema except SYS, AUDSYS.
ALTER ANY ANALYTIC VIEW	Rename analytic views in any schema except SYS, AUDSYS.
DROP ANY ANALYTIC VIEW	Drop analytic views in any schema except SYS, AUDSYS .
ATTRIBUTE DIMENSIONS	—

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
CREATE ATTRIBUTE DIMENSION	Create attribute dimensions in the grantee's schema.
CREATE ANY ATTRIBUTE DIMENSION	Create attribute dimensions in any schema except SYS,AUDSYS.
ALTER ANY ATTRIBUTE DIMENSION	Rename attribute dimensions in any schema except SYS,AUDSYS.
DROP ANY ATTRIBUTE DIMENSION	Drop attribute dimensions in any schema except SYS,AUDSYS.
AUDIT:	—
AUDIT ANY	Audit an object in any schema, except SYS,AUDSYS, using AUDIT *schema_objects* statements.
AUDIT SYSTEM	Issue AUDIT statements.
ADMINISTER FINE GRAINED AUDIT POLICY	Allow management of fine-grained audit policies
CLUSTERS:	—
CREATE CLUSTER	Create clusters in the grantee's schema.
CREATE ANY CLUSTER	Create clusters in any schema except SYS,AUDSYS. Behaves similarly to CREATE ANY TABLE.
ALTER ANY CLUSTER	Alter clusters in any schema except SYS, AUDSYS.
DROP ANY CLUSTER	Drop clusters in any schema except SYS,AUDSYS.
CONTEXTS:	—
CREATE ANY CONTEXT	Create any context namespace.
DROP ANY CONTEXT	Drop any context namespace.
DATA REDACTION:	—
EXEMPT REDACTION POLICY	Bypass any existing Oracle Data Redaction policies and view actual data from tables or views on which Data Redaction policies are defined.
ADMINISTER REDACTION POLICY	Allow management of Redaction policies .
DATABASE:	—
ALTER DATABASE	Alter the database.
ALTER SYSTEM	Issue ALTER SYSTEM statements.
DATABASE LINKS:	—
CREATE DATABASE LINK	Create private database links in the grantee's schema.
CREATE PUBLIC DATABASE LINK	Create public database links.
ALTER DATABASE LINK	Modify a fixed-user database link when the password of the connection or authentication user changes.
ALTER PUBLIC DATABASE LINK	Modify a public fixed-user database link when the password of the connection or authentication user changes.
DROP PUBLIC DATABASE LINK	Drop public database links.
DEBUGGING:	—
DEBUG CONNECT SESSION	Connect the current session to a debugger.
DEBUG ANY PROCEDURE	Debug all PL/SQL and Java code in any database object. Display information on all SQL statements executed by the application. **Note:** Granting this privilege is equivalent to granting the DEBUG object privilege on all applicable objects in the database.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
DICTIONARIES:	—
ANALYZE ANY DICTIONARY	Analyze any data dictionary object.
DIMENSIONS:	—
CREATE DIMENSION	Create dimensions in the grantee's schema.
CREATE ANY DIMENSION	Create dimensions in any schema except SYS,AUDSYS.
ALTER ANY DIMENSION	Alter dimensions in any schema except SYS,AUDSYS.
DROP ANY DIMENSION	Drop dimensions in any schema except SYS,AUDSYS.
DIRECTORIES:	—
CREATE ANY DIRECTORY	Create directory database objects.
DROP ANY DIRECTORY	Drop directory database objects.
DOMAINS:	—
CREATE DOMAIN	Create a domain in your own schema.
CREATE ANY DOMAIN	Create a domain in any schema.
ALTER ANY DOMAIN	Alter a domain in any schema.
DROP ANY DOMAIN	Drop a domain in any schema.
EXECUTE ANY DOMAIN	Refer to a domain in any schema.
EDITIONS:	—
CREATE ANY EDITION	Create editions.
DROP ANY EDITION	Drop editions.
FLASHBACK DATA ARCHIVES:	—
FLASHBACK ARCHIVE ADMINISTER	Create, alter, or drop any flashback data archive.
HIERARCHIES	—
CREATE HIERARCHY	Create hierarchies in the grantee's schema.
CREATE ANY HIERARCHY	Create hierarchies in any schema except SYS,AUDSYS.
ALTER ANY HIERARCHY	Rename hierarchies in any schema except SYS,AUDSYS.
DROP ANY HIERARCHY	Drop hierarchies in any schema except SYS, AUDSYS.
INDEXES:	—
CREATE ANY INDEX	Create in any schema, except SYS, AUDSYS, a domain index or an index on any table in any schema except SYS,AUDSYS.
ALTER ANY INDEX	Alter indexes in any schema except SYS,AUDSYS.
DROP ANY INDEX	Drop indexes in any schema except SYS,AUDSYS.
INDEXTYPES:	—
CREATE INDEXTYPE	Create indextypes in the grantee's schema.
CREATE ANY INDEXTYPE	Create indextypes in any schema except SYS and create comments on indextypes in any schema except SYS.
ALTER ANY INDEXTYPE	Modify indextypes in any schema except SYS,AUDSYS.
DROP ANY INDEXTYPE	Drop indextypes in any schema except SYS,AUDSYS.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
EXECUTE ANY INDEXTYPE	Reference indextypes in any schema except SYS,AUDSYS.
JOB SCHEDULER OBJECTS:	The following privileges are needed to execute procedures in the DBMS_SCHEDULER package. This privileges do not apply to lightweight jobs, which are not database objects. Refer to *Oracle Database Administrator's Guide* for more information about lightweight jobs.
CREATE JOB	Create, alter, or drop jobs, chains, schedules, programs, credentials, resource objects, or incompatibility resource objects in the grantee's schema.
CREATE ANY JOB	Create, alter, or drop jobs, chains, schedules, programs, credentials, resource objects, or incompatibility resource objects in any schema except SYS,AUDSYS. **Note:** This extremely powerful privilege allows the grantee to execute code as any other user. It should be granted with caution.
CREATE EXTERNAL JOB	Create in the grantee's schema an executable scheduler job that runs on the operating system.
EXECUTE ANY CLASS	Specify any job class in a job in the grantee's schema.
EXECUTE ANY PROGRAM	Use any program in a job in the grantee's schema.
MANAGE SCHEDULER	Create, alter, or drop any job class, window, or window group.
USE ANY JOB RESOURCE	Associate any schedule resource object with any program or job in the grantee's schema.
KEY MANAGEMENT FRAMEWORK:	—
ADMINISTER KEY MANAGEMENT	Manage keys and keystores.
LIBRARIES:	**Caution:** CREATE LIBARARY, CREATE ANY LIBRARY, ALTER ANY LIBRARY, and EXECUTE ANY LIBRARY are extremely powerful privileges that should be granted only to trusted users. Refer to *Oracle Database Security Guide* before granting these privileges.
CREATE LIBRARY	Create external procedure or function libraries in the grantee's schema.
CREATE ANY LIBRARY	Create external procedure or function libraries in any schema except SYS,AUDSYS.
ALTER ANY LIBRARY	Alter external procedure or function libraries in any schema except SYS,AUDSYS.
DROP ANY LIBRARY	Drop external procedure or function libraries in any schema except SYS,AUDSYS.
EXECUTE ANY LIBRARY	Use external procedure or function libraries in any schema except SYS,AUDSYS.
LOGMINER:	—
LOGMINING	Execute procedures in the DBMS_LOGMNR package in a CDB or a PDB. Query the contents of the V$LOGMNR_CONTENTS view.
MATERIALIZED VIEWS:	—
CREATE MATERIALIZED VIEW	Create materialized views in the grantee's schema.
CREATE ANY MATERIALIZED VIEW	Create materialized views in any schema except SYS,AUDSYS.
ALTER ANY MATERIALIZED VIEW	Alter materialized views in any schema except SYS,AUDSYS.
DROP ANY MATERIALIZED VIEW	Drop materialized views in any schema except SYS,AUDSYS.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
QUERY REWRITE	This privilege has been deprecated. No privileges are needed for a user to enable rewrite for a materialized view that references tables or views in the user's own schema.
GLOBAL QUERY REWRITE	Enable rewrite using a materialized view when that materialized view references tables or views in any schema except SYS.
ON COMMIT REFRESH	Create a refresh-on-commit materialized view on any table in the database. Alter a refresh-on-demand materialized view on any table in the database to refresh-on-commit.
FLASHBACK ANY TABLE	Issue a SQL Flashback Query on any table, view, or materialized view in any schema except SYS. This privilege is not needed to execute the DBMS_FLASHBACK procedures.
MINING MODELS:	—
CREATE MINING MODEL	Create mining models in the grantee's schema using the DBMS_DATA_MINING.CREATE_MODEL procedure.
CREATE ANY MINING MODEL	Create mining models in any schema, except SYS, AUDSYS, using the DBMS_DATA_MINING.CREATE_MODEL procedure.
ALTER ANY MINING MODEL	Change the mining model name or the associated cost matrix of a model in any schema, except SYS, AUDSYS, using the applicable DBMS_DATA_MINING procedures.
DROP ANY MINING MODEL	Drop mining models in any schema, except SYS,AUDSYS, using the DBMS_DATA_MINING.DROP_MODEL procedure.
SELECT ANY MINING MODEL	Score or view mining models in any schema except SYS, AUDSYS. Scoring is done either with the PREDICTION family of SQL functions or with the DBMS_DATA_MINING.APPLY procedure. Viewing the model is done with the DBMS_DATA_MINING.GET_MODEL_DETAILS_* procedures.
COMMENT ANY MINING MODEL	Create comments on mining models in any schema, except SYS, AUDSYS, using the SQL COMMENT statement.
OLAP CUBES:	The following privileges are valid when you are using Oracle Database with the OLAP option.
CREATE CUBE	Create OLAP cubes in the grantee's schema.
CREATE ANY CUBE	Create OLAP cubes in any schema except SYS,AUDSYS.
ALTER ANY CUBE	Alter OLAP cubes in any schema except SYS,AUDSYS.
DROP ANY CUBE	Drop OLAP cubes in any schema except SYS,AUDSYS.
SELECT ANY CUBE	Query or view OLAP cubes in any schema except SYS,AUDSYS.
UPDATE ANY CUBE	Update OLAP cubes in any schema except SYS,AUDSYS.
OLAP CUBE MEASURE FOLDERS:	The following privileges are valid when you are using Oracle Database with the OLAP option.
CREATE MEASURE FOLDER	Create OLAP measure folders in the grantee's schema.
CREATE ANY MEASURE FOLDER	Create OLAP measure folders in any schema except SYS,AUDSYS.
DELETE ANY MEASURE FOLDER	Delete a measure from an OLAP measure folder in any schema except SYS,AUDSYS.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
DROP ANY MEASURE FOLDER	Drop OLAP measure folders in any schema except SYS,AUDSYS.
INSERT ANY MEASURE FOLDER	Insert a measure into an OLAP measure folder in any schema except SYS,AUDSYS.
OLAP CUBE DIMENSIONS:	The following privileges are valid when you are using Oracle Database with the OLAP option.
CREATE CUBE DIMENSION	Create OLAP cube dimension in the grantee's schema.
CREATE ANY CUBE DIMENSION	Create OLAP cube dimensions in any schema except SYS,AUDSYS.
ALTER ANY CUBE DIMENSION	Alter OLAP cube dimensions in any schema except SYS,AUDSYS.
DELETE ANY CUBE DIMENSION	Delete from OLAP cube dimensions in any schema except SYS, AUDSYS.
DROP ANY CUBE DIMENSION	Drop OLAP cube dimensions in any schema except SYS,AUDSYS.
INSERT ANY CUBE DIMENSION	Insert into OLAP cube dimensions in any schema except SYS,AUDSYS.
SELECT ANY CUBE DIMENSION	View or query OLAP cube dimensions in any schema except SYS,AUDSYS.
UPDATE ANY CUBE DIMENSION	Update OLAP cube dimensions in any schema except SYS,AUDSYS.
OLAP CUBE BUILD PROCESSES:	—
CREATE CUBE BUILD PROCESS	Create OLAP cube build processes in the grantee's schema.
CREATE ANY CUBE BUILD PROCESS	Create OLAP cube build processes in any schema except SYS,AUDSYS.
DROP ANY CUBE BUILD PROCESS	Drop OLAP cube build processes in any schema except SYS,AUDSYS.
UPDATE ANY CUBE BUILD PROCESS	Update OLAP cube build processes in any schema except SYS,AUDSYS.
OPERATORS:	—
CREATE OPERATOR	Create an operator and its bindings in the grantee's schema.
CREATE ANY OPERATOR	Create an operator and its bindings in any schema and create a comment on an operator in any schema.
ALTER ANY OPERATOR	Modify operators in any schema.
DROP ANY OPERATOR	Drop operators in any schema.
EXECUTE ANY OPERATOR	Reference operators in any schema.
OUTLINES:	—
CREATE ANY OUTLINE	Create public outlines that can be used in any schema that uses outlines.
ALTER ANY OUTLINE	Modify outlines.
DROP ANY OUTLINE	Drop outlines.
PDB LOCKDOWN PROFILES:	—
CREATE LOCKDOWN PROFILE	Create PDB lockdown profiles.
ALTER LOCKDOWN PROFILE	Alter PDB lockdown profiles.
DROP LOCKDOWN PROFILE	Drop PDB lockdown profiles.
PLAN MANAGEMENT:	—

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
ADMINISTER SQL MANAGEMENT OBJECT	Perform controlled manipulation of plan history and SQL plan baselines maintained for various SQL statements.
PLUGGABLE DATABASES:	—
CREATE PLUGGABLE DATABASE	Create a PDB. Plug in a PDB that was previously unplugged from a CDB. Clone a PDB.
SET CONTAINER	Allow a common user to switch into the container for which this privilege was granted. This privilege can be granted only to a common user or common role.
PROCEDURES:	—
CREATE PROCEDURE	Create stored procedures, functions, or packages in the grantee's schema.
CREATE ANY PROCEDURE	Create stored procedures, functions, or packages in any schema except SYS,AUDSYS.
ALTER ANY PROCEDURE	Alter stored procedures, functions, or packages in any schema except SYS,AUDSYS.
DROP ANY PROCEDURE	Drop stored procedures, functions, or packages in any schema except SYS,AUDSYS.
EXECUTE ANY PROCEDURE	Execute procedures or functions, either standalone or packaged. Reference public package variables in any schema except SYS,AUDSYS.
INHERIT ANY REMOTE PRIVILEGES	Execute definer's rights procedures or functions that contain current user database links.
PROFILES:	—
CREATE PROFILE	Create profiles.
ALTER PROFILE	Alter profiles.
DROP PROFILE	Drop profiles.
PROPERTY GRAPHS:	—
CREATE PROPERTY GRAPH	Create property graph in the grantee’s schema.
CREATE ANY PROPERTY GRAPH	Create property graph in any schema except SYS, AUDSYS.
ALTER ANY PROPERTY GRAPH	Alter property graph in any schema except SYS, AUDSYS.
DROP ANY PROPERTY GRAPH	Drop property graph in any schema except SYS, AUDSYS.
READ ANY PROPERTY GRAPH	Query property graph in any schema except SYS, AUDSYS.
ROLES:	—
CREATE ROLE	Create roles.
ALTER ANY ROLE	Alter any role in the database.
DROP ANY ROLE	Drop roles.
GRANT ANY ROLE	Grant any role in the database.
ROLLBACK SEGMENTS:	—
CREATE ROLLBACK SEGMENT	Create rollback segments.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
ALTER ROLLBACK SEGMENT	Alter rollback segments.
DROP ROLLBACK SEGMENT	Drop rollback segments.
SEQUENCES:	—
CREATE SEQUENCE	Create sequences in the grantee's schema.
CREATE ANY SEQUENCE	Create sequences in any schema except SYS,AUDSYS.
ALTER ANY SEQUENCE	Alter sequences in any schema except SYS,AUDSYS.
DROP ANY SEQUENCE	Drop sequences in any schema except SYS,AUDSYS.
SELECT ANY SEQUENCE	Reference sequences in any schema except SYS,AUDSYS.
SESSIONS:	—
CREATE SESSION	Connect to the database.
ALTER RESOURCE COST	Set costs for session resources.
ALTER SESSION	Enable and disable the SQL trace facility.
RESTRICTED SESSION	Logon after the instance is started using the SQL*Plus STARTUP RESTRICT statement.
SNAPSHOTS:	See MATERIALIZED VIEWS
SQL Firewall Administration	–
ADMINISTER SQL FIREWALL	This system privilege is required to execute the PL/SQL procedures in SYS.DBMS_SQL_FIREWALL package. Just like any other system privileges, SYS is assumed to have this privilege. However this system privilege will not be granted to the DBA role by default.
SQL TRANSLATION PROFILES:	—
CREATE SQL TRANSLATION PROFILE	Create SQL translation profiles in the grantee's schema.
CREATE ANY SQL TRANSLATION PROFILE	Create SQL translation profiles in any schema except SYS,AUDSYS.
ALTER ANY SQL TRANSLATION PROFILE	Alter the translator, custom SQL statement translations, or custom error translations of a SQL translation profile in any schema except SYS,AUDSYS.
USE ANY SQL TRANSLATION PROFILE	Use SQL translation profiles in any schema except SYS,AUDSYS.
DROP ANY SQL TRANSLATION PROFILE	Drop SQL translation profiles in any schema except SYS,AUDSYS.
TRANSLATE ANY SQL	Translate SQL through the grantee's SQL translation profile for any user.
SYNONYMS:	**Caution:** CREATE PUBLIC SYNONYM and DROP PUBLIC SYNONYM are extremely powerful privileges that should be granted only to trusted users. Refer to *Oracle Database Security Guide* before granting these privileges.
CREATE SYNONYM	Create synonyms in the grantee's schema.
CREATE ANY SYNONYM	Create private synonyms in any schema except SYS,AUDSYS.
CREATE PUBLIC SYNONYM	Create public synonyms.
DROP ANY SYNONYM	Drop private synonyms in any schema except SYS,AUDSYS.
DROP PUBLIC SYNONYM	Drop public synonyms.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
TABLES:	**Note:** For external tables, the only valid privileges are CREATE ANY TABLE, ALTER ANY TABLE, DROP ANY TABLE, READ ANY TABLE, and SELECT ANY TABLE.
CREATE TABLE	Create tables in the grantee's schema.
CREATE ANY TABLE	Create a table in any schema except SYS,AUDSYS. The owner of the schema containing the table must have space quota on the tablespace to contain the table.
ALTER ANY TABLE	Alter a table or view in any schema except SYS, AUDSYS.
BACKUP ANY TABLE	Use the Export utility to incrementally export objects from the schema of other users except SYS,AUDSYS.
DELETE ANY TABLE	Delete rows from tables, table partitions, or views in any schema except SYS,AUDSYS.
DROP ANY TABLE	Drop or truncate tables or table partitions in any schema except SYS,AUDSYS.
INSERT ANY TABLE	Insert rows into tables and views in any schema except SYS,AUDSYS.
LOCK ANY TABLE	Lock tables and views in any schema except SYS,AUDSYS.
READ ANY TABLE	Query tables, views, or materialized views in any schema except SYS,AUDSYS.
SELECT ANY TABLE	Query tables, views, or materialized views in any schema except SYS,AUDSYS. Obtain row locks using a SELECT ... FOR UPDATE.
FLASHBACK ANY TABLE	Issue a SQL Flashback Query on any table, view, or materialized view in any schema except SYS,AUDSYS. This privilege is not needed to execute the DBMS_FLASHBACK procedures.
UPDATE ANY TABLE	Update rows in tables and views in any schema except SYS,AUDSYS.
REDEFINE ANY TABLE	Perform online redefinition without granting any of the privileges in USER or FULL mode.
TABLE RETENTION	Create a blockchain table or an immutable table whose table retention exceeds the threshold specified by the parameter BLOCKCHAIN_TABLE_RETENTION_THRESHOLD. Increase the table retention for an existing blockchain table or immutable table to a value above the threshold specified by the parameter BLOCKCHAIN_TABLE_RETENTION_THRESHOLD.
TABLESPACES:	—
CREATE TABLESPACE	Create tablespaces.
ALTER TABLESPACE	Alter tablespaces.
DROP TABLESPACE	Drop tablespaces.
MANAGE TABLESPACE	Take tablespaces offline and online and begin and end tablespace backups.
UNLIMITED TABLESPACE	Use an unlimited amount of any tablespace. This privilege overrides any specific quotas assigned. If you revoke this privilege from a user, then the user's schema objects remain but further tablespace allocation is denied unless authorized by specific tablespace quotas. You cannot grant this system privilege to roles.
TRIGGERS:	—

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
CREATE TRIGGER	Create database triggers in the grantee's schema.
CREATE ANY TRIGGER	Create database triggers in any schema except SYS, AUDSYS.
ALTER ANY TRIGGER	Enable, disable, or compile database triggers in any schema except SYS,AUDSYS.
DROP ANY TRIGGER	Drop database triggers in any schema except SYS,AUDSYS.
ADMINISTER DATABASE TRIGGER	Create a trigger on DATABASE. You must also have the CREATE TRIGGER or CREATE ANY TRIGGER system privilege.
TYPES:	—
CREATE TYPE	Create object types and object type bodies in the grantee's schema.
CREATE ANY TYPE	Create object types and object type bodies in any schema except SYS,AUDSYS.
ALTER ANY TYPE	Alter object types in any schema except SYS,AUDSYS.
DROP ANY TYPE	Drop object types and object type bodies in any schema except SYS,AUDSYS.
EXECUTE ANY TYPE	Use and reference object types and collection types in any schema except SYS,AUDSYS, and invoke methods of an object type in any schema, except SYS,AUDSYS, if you make the grant to a specific user. If you grant EXECUTE ANY TYPE to a role, then users holding the enabled role will not be able to invoke methods of an object type in any schema.
UNDER ANY TYPE	Create subtypes under any nonfinal object types.
USERS:	—
CREATE USER	Create users. This privilege also allows the creator to: • Assign quotas on any tablespace. • Set default and temporary tablespaces. • Assign a profile as part of a CREATE USER statement.
ALTER USER	Alter any user except SYS. This privilege authorizes the grantee to: • Change another user's password or authentication method. • Assign quotas on any tablespace. • Set default and temporary tablespaces. • Assign a profile and default roles.
DROP USER	Drop users
VIEWS:	—
CREATE VIEW	Create views in the grantee's schema.
CREATE ANY VIEW	Create views in any schema except SYS,AUDSYS.
DROP ANY VIEW	Drop views in any schema except SYS,AUDSYS.
UNDER ANY VIEW	Create subviews under any object views.
FLASHBACK ANY TABLE	Issue a SQL Flashback Query on any table, view, or materialized view in any schema except SYS,AUDSYS. This privilege is not needed to execute the DBMS_FLASHBACK procedures.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
MERGE ANY VIEW	If a user has been granted the MERGE ANY VIEW privilege, then for any query issued by that user, the optimizer can use view merging to improve query performance without performing the checks that would otherwise be performed to ensure that view merging does not violate any security intentions of the view creator. See *Oracle Database SQL Tuning Guide* for information on view merging.
VIRTUAL PRIVATE DATABASE	
EXEMPT ACCESS POLICY	Bypass fine-grained access control. **Caution:** This is a very powerful system privilege, as it lets the grantee bypass application-driven security policies. Database administrators should use caution when granting this privilege.
ADMINISTER ROW LEVEL SECURITY POLICY	Allow management of Virtual Private Database (VPD) policies (fine-grained access control, row-level security)
MISCELLANEOUS:	—
ANALYZE ANY	Analyze a table, cluster, or index in any schema except SYS.
BECOME USER	Allow users of the Data Pump Import utility (impdp) and the original Import utility (imp) to assume the identity of another user in order to perform operations that cannot be directly performed by a third party (for example, loading objects such as object privilege grants). Allow Streams administrators to create or alter capture users and apply users in a Streams environment. By default this privilege is part of the DBA role. Database Vault removes this privileges from the DBA role. Therefore, this privilege is needed by Streams only in an environment where Database Vault is installed.
CHANGE NOTIFICATION	Create a registration on queries and receive database change notifications in response to DML or DDL changes to the objects associated with the registered queries. Refer to *Oracle Database Development Guide* for more information on database change notification.
COMMENT ANY TABLE	Comment on a table, view, or column in any schema except SYS,AUDSYS.
ENABLE DIAGNOSTICS	• Set debug-events via ALTER SESSION or ALTER SYSTEM • Set debug-actions to execute during an event via ALTER SESSION or ALTER SYSTEM • Execute debug-actions immediately via ALTER SESSION or ALTER SYSTEM by specifying the IMMEDIATE keyword instead of an event name • Set the event parameter in the spfile via ALTER SYSTEM
FORCE ANY TRANSACTION	Force the commit or rollback of any in-doubt distributed transaction in the local database. Induce the failure of a distributed transaction.
FORCE TRANSACTION	Force the commit or rollback of the grantee's in-doubt distributed transactions in the local database.
GRANT ANY OBJECT PRIVILEGE	Grant any object privilege that the object owner is permitted to grant. Revoke any object privilege that was granted by the object owner or by some other user with the GRANT ANY OBJECT PRIVILEGE privilege.
GRANT ANY PRIVILEGE	Grant any system privilege.

Table 18-2 (Cont.) System Privileges (Organized by the Database Object Operated Upon)

System Privilege Name	Operations Authorized
INHERIT ANY PRIVILEGES	Execute invoker's rights procedures owned by the grantee with the privileges of the invoker.
KEEP DATE TIME	The SYSDATE and SYSTIMESTAMP functions return their original values during replay for Application Continuity when the grantee is running the application. This privilege is useful for providing bind variable consistency after recoverable errors. **Note:** If this privilege is granted or revoked between runtime and failover of a request, then the original values are not returned during replay for Application Continuity for that request.
KEEP SYSGUID	The SYS_GUID function returns its original value during replay for Application Continuity when the grantee is running the application. This privilege is useful for providing bind variable consistency after recoverable errors. **Note:** If this privilege is granted or revoked between runtime and failover of a request, then the original value is not returned during replay for Application Continuity for that request.
PURGE DBA_RECYCLEBIN	Remove all objects from the system-wide recycle bin.
RESUMABLE	Enable resumable space allocation.
SELECT ANY DICTIONARY	Query any data dictionary object in the dictioanry protected schema, with the exception of the following objects: SYS.DEFAULT_PWD$, SYS.ENC$, SYS.LINK$, SYS.USER$, SYS.USER_HISTORY$, and SYS.XS$VERIFIERS. **Note:** The privilege will NOT allow the grantee to execute SELECT .. FOR UPDATE on a dictionary table. It will allow “READ” on dictionary objects
SELECT ANY TRANSACTION	Query the contents of the FLASHBACK_TRANSACTION_QUERY view. **Caution:** This is a very powerful system privilege, as it lets the grantee view all data in the database, including past data. This privilege should be granted only to users who need to use the Oracle Flashback Transaction Query feature.

Table 18-3 Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
ANALYTIC VIEWS	—
CREATE ANY ANALYTIC VIEW	Create analytic views in any schema except SYS, AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE ANALYTIC VIEW privilege.
ALTER ANY ANALYTIC VIEW	Rename analytic views in any schema except SYS, AUDSYS.
DROP ANY ANALYTIC VIEW	Drop analytic views in any schema except SYS, AUDSYS .
ATTRIBUTE DIMENSIONS	—
CREATE ANY ATTRIBUTE DIMENSION	Create attribute dimensions in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE ATTRIBUTE DIMENSION privilege.
ALTER ANY ATTRIBUTE DIMENSION	Rename attribute dimensions in any schema except SYS,AUDSYS.

Table 18-3 (Cont.) Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
DROP ANY ATTRIBUTE DIMENSION	Drop attribute dimensions in any schema except SYS,AUDSYS.
AUDIT GROUP:	—
AUDIT ANY	Audit an object in any schema, except SYS,AUDSYS, using AUDIT *schema_objects* statements.
ADMINISTER FINE GRAINED AUDIT POLICY	Allow management of fine-grained audit policies in a schema.
CLUSTERS:	—
CREATE ANY CLUSTER	Create clusters in any schema except SYS,AUDSYS. Behaves similarly to CREATE ANY TABLE. If the grantee is the schema owner, then grantee should have been granted CREATE CLUSTER privilege.
ALTER ANY CLUSTER	Alter clusters in any schema except SYS, AUDSYS.
DROP ANY CLUSTER	Drop clusters in any schema except SYS,AUDSYS.
DATA REDACTION:	—
EXEMPT REDACTION POLICY	Bypass any existing Oracle Data Redaction policies and view actual data from tables or views on which Data Redaction policies are defined in the schema.
ADMINISTER REDACTION POLICY	Allow management of redaction policies in a schema.
DIMENSIONS:	—
CREATE ANY DIMENSION	Create dimensions in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE DIMENSION privilege.
ALTER ANY DIMENSION	Alter dimensions in any schema except SYS,AUDSYS.
DROP ANY DIMENSION	Drop dimensions in any schema except SYS,AUDSYS.
DOMAINS:	—
CREATE ANY DOMAIN	Create a DOMAIN in any non-dictionary protected schema. If the grantee is the schema owner, then grantee should have been granted CREATE DOMAIN privilege.
ALTER ANY DOMAIN	Rename a domain in any non-dictionary protected schema.
DROP ANY DOMAIN	Drop a domain in any non-dictionary protected schema.
EXECUTE ANY DOMAIN	Execute a domain in a designated non-dictionary protected schema.
HIERARCHIES	—
CREATE ANY HIERARCHY	Create hierarchies in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE HIERARCHY privilege.
ALTER ANY HIERARCHY	Rename hierarchies in any schema except SYS,AUDSYS.
DROP ANY HIERARCHY	Drop hierarchies in any schema except SYS, AUDSYS.
INDEXES:	—

Table 18-3 (Cont.) Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
CREATE ANY INDEX	Create in any schema, except SYS, AUDSYS, a domain index or an index on any table in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE INDEX privilege.
ALTER ANY INDEX	Alter indexes in any schema except SYS,AUDSYS.
DROP ANY INDEX	Drop indexes in any schema except SYS,AUDSYS.
INDEXTYPES:	—
CREATE ANY INDEXTYPE	Create indextypes in any schema except SYS and create comments on indextypes in any schema except SYS. If the grantee is the schema owner, then grantee should have been granted CREATE INDEXTYPE privilege.
ALTER ANY INDEXTYPE	Modify indextypes in any schema except SYS,AUDSYS.
DROP ANY INDEXTYPE	Drop indextypes in any schema except SYS,AUDSYS.
EXECUTE ANY INDEXTYPE	Reference indextypes in any schema except SYS,AUDSYS.
JOB SCHEDULER OBJECTS:	The following privileges are needed to execute procedures in the DBMS_SCHEDULER package. This privileges do not apply to lightweight jobs, which are not database objects. Refer to *Oracle Database Administrator's Guide* for more information about lightweight jobs.
CREATE ANY JOB	Create, alter, or drop jobs, chains, schedules, programs, credentials, resource objects, or incompatibility resource objects in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE JOB privilege. **Note:** This extremely powerful privilege allows the grantee to execute code as any other user. It should be granted with caution.
LIBRARIES:	**Caution:** CREATE LIBARARY, CREATE ANY LIBRARY, ALTER ANY LIBRARY, and EXECUTE ANY LIBRARY are extremely powerful privileges that should be granted only to trusted users. Refer to *Oracle Database Security Guide* before granting these privileges.
CREATE ANY LIBRARY	Create external procedure or function libraries in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE LIBRARY privilege.
ALTER ANY LIBRARY	Alter external procedure or function libraries in any schema except SYS,AUDSYS.
DROP ANY LIBRARY	Drop external procedure or function libraries in any schema except SYS,AUDSYS.
EXECUTE ANY LIBRARY	Use external procedure or function libraries in any schema except SYS,AUDSYS.
MATERIALIZED VIEWS:	—
CREATE ANY MATERIALIZED VIEW	Create materialized views in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE MATERIALIZED VIEW privilege.
ALTER ANY MATERIALIZED VIEW	Alter materialized views in any schema except SYS,AUDSYS.

Table 18-3 (Cont.) Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
DROP ANY MATERIALIZED VIEW	Drop materialized views in any schema except SYS,AUDSYS.
GLOBAL QUERY REWRITE	Enable rewrite using a materialized view when that materialized view references tables or views in any schema except SYS.
FLASHBACK ANY TABLE	Issue a SQL Flashback Query on any table, view, or materialized view in any schema except SYS. This privilege is not needed to execute the DBMS_FLASHBACK procedures.
MINING MODELS:	—
CREATE ANY MINING MODEL	Create mining models in any schema, except SYS, AUDSYS, using the DBMS_DATA_MINING.CREATE_MODEL procedure. If the grantee is the schema owner, then grantee should have been granted CREATE MINING MODEL privilege.
ALTER ANY MINING MODEL	Change the mining model name or the associated cost matrix of a model in any schema, except SYS, AUDSYS, using the applicable DBMS_DATA_MINING procedures.
DROP ANY MINING MODEL	Drop mining models in any schema, except SYS,AUDSYS, using the DBMS_DATA_MINING.DROP_MODEL procedure.
SELECT ANY MINING MODEL	Score or view mining models in any schema except SYS, AUDSYS. Scoring is done either with the PREDICTION family of SQL functions or with the DBMS_DATA_MINING.APPLY procedure. Viewing the model is done with the DBMS_DATA_MINING.GET_MODEL_DETAILS_* procedures.
COMMENT ANY MINING MODEL	Create comments on mining models in any schema, except SYS, AUDSYS, using the SQL COMMENT statement.
OLAP CUBES:	The following privileges are valid when you are using Oracle Database with the OLAP option.
CREATE ANY CUBE	Create OLAP cubes in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE CUBE privilege.
ALTER ANY CUBE	Alter OLAP cubes in any schema except SYS,AUDSYS.
DROP ANY CUBE	Drop OLAP cubes in any schema except SYS,AUDSYS.
SELECT ANY CUBE	Query or view OLAP cubes in any schema except SYS,AUDSYS.
UPDATE ANY CUBE	Update OLAP cubes in any schema except SYS,AUDSYS.
OLAP CUBE MEASURE FOLDERS:	The following privileges are valid when you are using Oracle Database with the OLAP option.
CREATE ANY MEASURE FOLDER	Create OLAP measure folders in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE MEASURE FOLDER privilege.
DELETE ANY MEASURE FOLDER	Delete a measure from an OLAP measure folder in any schema except SYS,AUDSYS.
DROP ANY MEASURE FOLDER	Drop OLAP measure folders in any schema except SYS,AUDSYS.
INSERT ANY MEASURE FOLDER	Insert a measure into an OLAP measure folder in any schema except SYS,AUDSYS.

Table 18-3 (Cont.) Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
OLAP CUBE DIMENSIONS:	The following privileges are valid when you are using Oracle Database with the OLAP option.
CREATE ANY CUBE DIMENSION	Create OLAP cube dimensions in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE CUBE DIMENSION privilege.
ALTER ANY CUBE DIMENSION	Alter OLAP cube dimensions in any schema except SYS,AUDSYS.
DELETE ANY CUBE DIMENSION	Delete from OLAP cube dimensions in any schema except SYS, AUDSYS.
DROP ANY CUBE DIMENSION	Drop OLAP cube dimensions in any schema except SYS,AUDSYS.
INSERT ANY CUBE DIMENSION	Insert into OLAP cube dimensions in any schema except SYS,AUDSYS.
SELECT ANY CUBE DIMENSION	View or query OLAP cube dimensions in any schema except SYS,AUDSYS.
UPDATE ANY CUBE DIMENSION	Update OLAP cube dimensions in any schema except SYS,AUDSYS.
OLAP CUBE BUILD PROCESSES:	—
CREATE ANY CUBE BUILD PROCESS	Create OLAP cube build processes in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE CUBE BUILD PROCESS privilege.
DROP ANY CUBE BUILD PROCESS	Drop OLAP cube build processes in any schema except SYS,AUDSYS.
UPDATE ANY CUBE BUILD PROCESS	Update OLAP cube build processes in any schema except SYS,AUDSYS.
OPERATORS:	—
CREATE ANY OPERATOR	Create an operator and its bindings in any schema and create a comment on an operator in any schema. If the grantee is the schema owner, then grantee should have been granted CREATE OPERATOR privilege.
ALTER ANY OPERATOR	Modify operators in any schema.
DROP ANY OPERATOR	Drop operators in any schema.
EXECUTE ANY OPERATOR	Reference operators in any schema.
PROCEDURES:	—
CREATE ANY PROCEDURE	Create stored procedures, functions, or packages in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE PROCEDURE privilege.
ALTER ANY PROCEDURE	Alter stored procedures, functions, or packages in any schema except SYS,AUDSYS.
DROP ANY PROCEDURE	Drop stored procedures, functions, or packages in any schema except SYS,AUDSYS.
EXECUTE ANY PROCEDURE	Execute procedures or functions, either standalone or packaged. Reference public package variables in any schema except SYS,AUDSYS.
SEQUENCES:	—

Table 18-3 (Cont.) Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
CREATE ANY SEQUENCE	Create sequences in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE SEQUENCE privilege.
ALTER ANY SEQUENCE	Alter sequences in any schema except SYS,AUDSYS.
DROP ANY SEQUENCE	Drop sequences in any schema except SYS,AUDSYS.
SELECT ANY SEQUENCE	Reference sequences in any schema except SYS,AUDSYS.
SQL TRANSLATION PROFILES:	—
CREATE ANY SQL TRANSLATION PROFILE	Create SQL translation profiles in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE SQL TRANSLATION PROFILE privilege.
ALTER ANY SQL TRANSLATION PROFILE	Alter the translator, custom SQL statement translations, or custom error translations of a SQL translation profile in any schema except SYS,AUDSYS.
USE ANY SQL TRANSLATION PROFILE	Use SQL translation profiles in any schema except SYS,AUDSYS.
DROP ANY SQL TRANSLATION PROFILE	Drop SQL translation profiles in any schema except SYS,AUDSYS.
TRANSLATE ANY SQL	Translate SQL through the grantee's SQL translation profile for any user.
SYNONYMS:	-
CREATE ANY SYNONYM	Create private synonyms in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE SYNONYM privilege.
DROP ANY SYNONYM	Drop private synonyms in any schema except SYS,AUDSYS.
TABLES:	-
CREATE ANY TABLE	Create a table in any schema except SYS,AUDSYS. The owner of the schema containing the table must have space quota on the tablespace to contain the table. If the grantee is the schema owner, then grantee should have been granted CREATE TABLE privilege.
ALTER ANY TABLE	Alter a table or view in any schema except SYS, AUDSYS.
BACKUP ANY TABLE	Use the Export utility to incrementally export objects from the schema of other users except SYS,AUDSYS.
DELETE ANY TABLE	Delete rows from tables, table partitions, or views in any schema except SYS,AUDSYS.
DROP ANY TABLE	Drop or truncate tables or table partitions in any schema except SYS,AUDSYS.
INSERT ANY TABLE	Insert rows into tables and views in any schema except SYS,AUDSYS.
LOCK ANY TABLE	Lock tables and views in any schema except SYS,AUDSYS.
READ ANY TABLE	Query tables, views, or materialized views in any schema except SYS,AUDSYS.
SELECT ANY TABLE	Query tables, views, or materialized views in any schema except SYS,AUDSYS. Obtain row locks using a SELECT ... FOR UPDATE.

Table 18-3 (Cont.) Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
FLASHBACK ANY TABLE	Issue a SQL Flashback Query on any table, view, or materialized view in any schema except SYS,AUDSYS. This privilege is not needed to execute the DBMS_FLASHBACK procedures.
UPDATE ANY TABLE	Update rows in tables and views in any schema except SYS,AUDSYS.
TRIGGERS:	—
CREATE ANY TRIGGER	Create database triggers in any schema except SYS, AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE TRIGGER privilege.
ALTER ANY TRIGGER	Enable, disable, or compile database triggers in any schema except SYS,AUDSYS.
DROP ANY TRIGGER	Drop database triggers in any schema except SYS,AUDSYS.
TYPES:	—
CREATE ANY TYPE	Create object types and object type bodies in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE TYPE privilege.
ALTER ANY TYPE	Alter object types in any schema except SYS,AUDSYS.
DROP ANY TYPE	Drop object types and object type bodies in any schema except SYS,AUDSYS.
EXECUTE ANY TYPE	Use and reference object types and collection types in any schema except SYS,AUDSYS, and invoke methods of an object type in any schema, except SYS,AUDSYS, if you make the grant to a specific user. If you grant EXECUTE ANY TYPE to a role, then users holding the enabled role will not be able to invoke methods of an object type in any schema.
UNDER ANY TYPE	Create subtypes under any nonfinal object types.
VIEWS:	—
CREATE ANY VIEW	Create views in any schema except SYS,AUDSYS. If the grantee is the schema owner, then grantee should have been granted CREATE VIEW privilege.
DROP ANY VIEW	Drop views in any schema except SYS,AUDSYS.
UNDER ANY VIEW	Create subviews under any object views.
FLASHBACK ANY TABLE	Issue a SQL Flashback Query on any table, view, or materialized view in any schema except SYS,AUDSYS. This privilege is not needed to execute the DBMS_FLASHBACK procedures.
VIRTUAL PRIVATE DATABASE:	—
EXEMPT ACCESS POLICY	Bypass fine-grained access control. **Caution:** This is a very powerful system privilege, as it lets the grantee bypass application-driven security policies. Database administrators should use caution when granting this privilege.
ADMINISTER ROW LEVEL SECURITY POLICY	Allow management of Virtual Private Database (VPD) policies in a schema (fine-grained access control, row-level security).
MISCELLANEOUS:	—
ANALYZE ANY	Analyze a table, cluster, or index in any schema except SYS.

Table 18-3 (Cont.) Schema Privileges (Organized by the Operations Authorized)

Schema Privilege Name	Operations Authorized
COMMENT ANY TABLE	Comment on a table, view, or column in any schema except SYS,AUDSYS.

Table 18-4 Object Privileges (Organized by the Database Object Operated Upon)

Object Privilege Name	Operations Authorized
ANALYTIC VIEW PRIVILEGES	The following **analytic view privileges** authorize operations on analytic views.
ALTER	Rename the analytic view.
READ	Query the object with the SELECT statement.
SELECT	Query the object with the SELECT statement.
ATTRIBUTE DIMENSION PRIVILEGES	The following **attribute dimension privileges** authorize operations on attribute dimensions..
ALTER	Rename the attribute dimension.
DIRECTORY PRIVILEGES	The following **directory privileges** provide secured access to the files stored in the operating system directory to which the directory object serves as a pointer. The directory object contains the full path name of the operating system directory where the files reside. Because the files are actually stored outside the database, Oracle Database server processes also need to have appropriate file permissions on the file system server. Granting object privileges on the directory database object to individual database users, rather than on the operating system, allows the database to enforce security during file operations.
READ	Read files in the directory.
WRITE	Write files in the directory. This privilege is useful only in connection with external tables. It allows the grantee to determine whether the external table agent can write a log file or a bad file to the directory. **Restriction:** This privilege does not allow the grantee to write to a BFILE.
EXECUTE	Execute a preprocessor program that resides in the directory. A preprocessor program converts data to a supported format when loading data records from an external table with the ORACLE_LOADER access driver. Refer to *Oracle Database Utilities* for more information. This privilege does not implicitly allow READ access on the external table data.
EDITION PRIVILEGE	The following **edition privilege** authorizes the use of an edition.
USE	Use an edition.
FLASHBACK DATA ARCHIVE PRIVILEGE	The following **flashback data archive privilege** authorizes operations on flashback data archives.
FLASHBACK ARCHIVE	Enable or disable historical tracking for a table.
HIERARCHY PRIVILEGES	The following **hierarchy privileges** authorize operations on hierarchies.
ALTER	Rename the hierarchy.
READ	Query the object with the SELECT statement.
SELECT	Query the object with the SELECT statement.
INDEXTYPE PRIVILEGE	The following **indextype privilege** authorizes operations on indextypes.
EXECUTE	Reference an indextype.
LIBRARY PRIVILEGE	The following **library privilege** authorizes operations on a library.

Table 18-4 (Cont.) Object Privileges (Organized by the Database Object Operated Upon)

Object Privilege Name	Operations Authorized
EXECUTE	Use and reference the specified object and invoke its methods. **Caution:** This extremely powerful privilege should be granted only to trusted users. Refer to *Oracle Database Security Guide* before granting this privilege.
MATERIALIZED VIEW PRIVILEGES	The following **materialized view privileges** authorize operations on a materialized view. The DELETE, INSERT, and UPDATE privileges can be granted only to updatable materialized views.
ON COMMIT REFRESH	Create a refresh-on-commit materialized view on the specified table.
QUERY REWRITE	Create a materialized view for query rewrite using the specified table.
READ	Query the materialized view.
SELECT	Query the materialized view. Obtain row locks with the SELECT ... FOR UPDATE or LOCK TABLE statement.
MINING MODEL PRIVILEGES	The following mining model privileges authorize operations on a mining model. These privileges are not required for models within the users own schema.
ALTER	Change the mining model name or the associated cost matrix using the applicable DBMS_DATA_MINING procedures.
SELECT	Score or view the mining model. Scoring is done with the PREDICTION family of SQL functions or with the DBMS_DATA_MINING.APPLY procedure. Viewing the model is done with the DBMS_DATA_MINING.GET_MODEL_DETAILS_* procedures.
OBJECT TYPE PRIVILEGES	The following **object type privileges** authorize operations on a database object type.
DEBUG	Access, through a debugger, all public and nonpublic variables, methods, and types defined on the object type. Place a breakpoint or stop at a line or instruction boundary within the type body.
EXECUTE	Use and reference the specified object and invoke its methods. Access, through a debugger, public variables, types, and methods defined on the object type.
UNDER	Create a subtype under this type. You can grant this object privilege only if you have the UNDER ANY TYPE privilege WITH GRANT OPTION on the immediate supertype of this type.
OLAP PRIVILEGES	The following object privileges are valid if you are using Oracle Database with the OLAP option.
INSERT	Insert members into the OLAP cube dimension or measures into the measures folder.
ALTER	Change the definition of the OLAP cube dimension or cube.
DELETE	Delete members from the OLAP cube dimension or measures from the measures folder.
SELECT	View or query the OLAP cube or cube dimension.
UPDATE	Update measure values of the OLAP cube or attribute values of the cube dimension.
OPERATOR PRIVILEGE	The following **operator privilege** authorizes operations on user-defined operators.
EXECUTE	Reference an operator.
PROCEDURE, FUNCTION, PACKAGE PRIVILEGES	The following **procedure, function, and package privileges** authorize operations on procedures, functions, and packages. These privileges also apply to **Java sources, classes, and resources**, which Oracle Database treats as though they were procedures for purposes of granting object privileges.

Table 18-4 (Cont.) Object Privileges (Organized by the Database Object Operated Upon)

Object Privilege Name	Operations Authorized
DEBUG	Access, through a debugger, all public and nonpublic variables, methods, and types defined on the object. Place a breakpoint or stop at a line or instruction boundary within the procedure, function, or package. This privilege grants access to the declarations in the method or package specification and body.
EXECUTE	Execute the procedure or function directly, or access any program object declared in the specification of a package, or compile the object implicitly during a call to a currently invalid or uncompiled function or procedure. This privilege does not allow the grantee to explicitly compile using ALTER PROCEDURE or ALTER FUNCTION. For explicit compilation you need the appropriate ALTER system privilege. Access, through a debugger, public variables, types, and methods defined on the procedure, function, or package. This privilege grants access to the declarations in the method or package specification only. Job scheduler objects are created using the DBMS_SCHEDULER package. After these objects are created, you can grant the EXECUTE object privilege on job scheduler classes and programs. You can also grant ALTER privilege on job scheduler jobs, programs, and schedules. **Note:** Users do not need this privilege to execute a procedure, function, or package indirectly.
PROPERTY GRAPH PRIVILEGES	The following object privileges authorize operations on property graphs.
ALTER	Change the graph definition using ALTER PROPERTY GRAPH statement.
READ	Query the PROPERTY GRAPH with the SELECT statement. Does not allow SELECT … FOR UPDATE.
SELECT	Query the PROPERTY GRAPH with the SELECT statement.
SCHEDULER PRIVILEGES	Job scheduler objects are created using the DBMS_SCHEDULER package. After these objects are created, you can grant the following privileges.
EXECUTE	Operations on job classes, programs, chains, and credentials.
ALTER	Modifications to jobs, programs, chains, credentials, and schedules.
USE	Associate the specified scheduler resource object with programs and jobs.
SEQUENCE PRIVILEGES	The following **sequence privileges** authorize operations on a sequence.
ALTER	Change the sequence definition with the ALTER SEQUENCE statement.
KEEP SEQUENCE	The sequence pseudocolumn NEXTVAL retains its original value during replay for Application Continuity when the grantee is running the application. This privilege is useful for providing bind variable consistency when replaying after recoverable errors. If this privilege is granted or revoked between runtime and failover of a request, then the original value of NEXTVAL is not retained during replay for Application Continuity for that request. **Note:** This privilege is not granted by the GRANT ALL PRIVILEGES ON *sequence* statement. You must explicitly grant this privilege. **Note:** This privilege is part of the DBA role.
SELECT	Examine and increment values of the sequence with the CURRVAL and NEXTVAL pseudocolumns.
SQL TRANSLATION PROFILE PRIVILEGES	The following **SQL translation profile privileges** authorize operations on a SQL translation profile.

Table 18-4 (Cont.) Object Privileges (Organized by the Database Object Operated Upon)

Object Privilege Name	Operations Authorized
ALTER	Alter the translator, custom SQL statement translations, or custom error translations of a SQL translation profile.
USE	Use a SQL translation profile.
SYNONYM PRIVILEGES	**Synonym privileges** are the same as the privileges for the target object. Granting a privilege on a synonym is equivalent to granting the privilege on the base object. Similarly, granting a privilege on a base object is equivalent to granting the privilege on all synonyms for the object. If you grant to a user a privilege on a synonym, then the user can use either the synonym name or the base object name in the SQL statement that exercises the privilege.
TABLE PRIVILEGES	The following **table privileges** authorize operations on a table. Any one of following object privileges, except the READ privilege, allows the grantee to lock the table in any lock mode with the LOCK TABLE statement. **Note:** For external tables, the only valid object privileges are ALTER, READ, and SELECT.
ALTER	Change the table definition with the ALTER TABLE statement.
DEBUG	Access, through a debugger: • PL/SQL code in the body of any triggers defined on the table • Information on SQL statements that reference the table directly
DELETE	Remove rows from the table with the DELETE statement. **Note:** You must grant the SELECT privilege on the table along with the DELETE privilege if the table is on a remote database.
INDEX	Create an index on the table with the CREATE INDEX statement.
INSERT	Add new rows to the table with the INSERT statement. **Note:** You must grant the SELECT privilege on the table along with the INSERT privilege if the table is on a remote database.
READ	Query the table with the SELECT statement. Does not allow SELECT ... FOR UPDATE.
REFERENCES	Create a constraint that refers to the table. You cannot grant this privilege to a role.
SELECT	To allow access to specific tables during queries, grant the SELECT privilege on the table. Query the table with the SELECT statement, including SELECT ... FOR UPDATE.
UPDATE	Change data in the table with the UPDATE statement. **Note:** You must grant the SELECT privilege on the table along with the UPDATE privilege if the table is on a remote database.
FLASHBACK	To allow access to a specific table during queries, grant the FLASHBACK privilege on the table. Issue a SQL Flashback Query on the table.
SIGN	A user SCOTT with the SIGN privilege on a blockchain table can sign a row in the blockchain table as a delegate if the following condtions are met: The row does not already have a delegate signature The row does not contain a non-NULL delegate user ID that is different than the ID of user SCOTT.
USER PRIVILEGES	The following privileges authorize operations on a user.

Table 18-4 (Cont.) Object Privileges (Organized by the Database Object Operated Upon)

Object Privilege Name	Operations Authorized
INHERIT PRIVILEGES	Execute invoker's rights procedures or functions owned by the grantee with the privileges of the invoker when the invoker is the user on whom this privilege is granted.
INHERIT REMOTE PRIVILEGES	Allow the user on whom this privilege is granted to execute definer's rights procedures or functions that contain current user database links and are owned by the grantee.
TRANSLATE SQL	Translate SQL through the grantee's SQL translation profile for the user on whom this privilege is granted.
VIEW PRIVILEGES	The following **view privileges** authorize operations on a view. Any one of the following object privileges, except the READ privilege, allows the grantee to lock the view in any lock mode with the LOCK TABLE statement. To grant a privilege on a view, you must have that privilege with the GRANT OPTION on all of the base tables of the view.
DEBUG	Access, through a debugger: • PL/SQL code in the body of any triggers defined on the view • Information on SQL statements that reference the view directly
DELETE	Remove rows from the view with the DELETE statement.
INSERT	Add new rows to the view with the INSERT statement.
MERGE VIEW	This object privilege has the same behavior as the system privilege MERGE ANY VIEW, except that the privilege is limited to the views specified in the ON clause. For any query issued by the grantee on the specified views, the optimizer can use view merging to improve query performance without performing the checks that would otherwise be performed to ensure that view merging does not violate any security intentions of the view creator.
READ	Query the view with the SELECT statement. Does not allow SELECT ... FOR UPDATE.
REFERENCES	Define foreign key constraints on the view.
SELECT	Query the view with the SELECT statement, including SELECT ... FOR UPDATE. **See Also:** object_privilege for additional information on granting this object privilege on a view
UNDER	Create a subview under this view. You can grant this object privilege only if you have the UNDER ANY VIEW privilege WITH GRANT OPTION on the immediate superview of this view.
UPDATE	Change data in the view with the UPDATE statement.
FLASHBACK	To allow access to a specific view during queries, grant the FLASHBACK privilege on the view. Issue a SQL Flashback Query on the view.

Examples

Granting a System Privilege to a User: Example

To grant the CREATE SESSION system privilege to the sample user hr, allowing hr to log on to Oracle Database, issue the following statement:

```
GRANT CREATE SESSION
   TO hr;
```

Assigning User Passwords When Granting a System Privilege: Example

Assume that user `hr` exists and user `newuser` does not exist. The following statement resets the user `hr` password to *password1*, creates user `newuser` with *password2*, and grants both users the `CREATE SESSION` system privilege:

```
GRANT CREATE SESSION
  TO hr, newuser IDENTIFIED BY password1, password2;
```

Granting System Privileges to a Role: Example

The following statement grants appropriate system privileges to a data warehouse manager role, which was created in the "Creating a Role: Example":

```
GRANT
     CREATE ANY MATERIALIZED VIEW
   , ALTER ANY MATERIALIZED VIEW
   , DROP ANY MATERIALIZED VIEW
   , QUERY REWRITE
   , GLOBAL QUERY REWRITE
   TO dw_manager
   WITH ADMIN OPTION;
```

The `dw_manager` privilege domain now contains the system privileges related to materialized views.

Granting a Role with the ADMIN OPTION: Example

To grant the `dw_manager` role with the `ADMIN OPTION` to the sample user `sh`, issue the following statement:

```
GRANT dw_manager
   TO sh
   WITH ADMIN OPTION;
```

User `sh` can now perform the following operations with the `dw_manager` role:

- Enable the role and exercise any privileges in the privilege domain of the role, including the `CREATE MATERIALIZED VIEW` system privilege
- Grant and revoke the role to and from other users
- Drop the role
- Grant and revoke the `dw_manager` role to and from program units in the `sh` schema

Granting a Role with the DELEGATE OPTION: Example

To grant the `dw_manager` role with the `DELEGATE OPTION` to the sample user `sh`, issue the following statement:

```
GRANT dw_manager
   TO sh
   WITH DELEGATE OPTION;
```

User `sh` can now grant and revoke the `dw_manager` role to and from program units in the `sh` schema.

Granting Object Privileges to a Role: Example

The following example grants the `SELECT` object privileges to a data warehouse user role, which was created in the "Creating a Role: Example":

```
GRANT SELECT ON sh.sales TO warehouse_user;
```

Granting a Role to a Role: Example

The following statement grants the `warehouse_user` role to the `dw_manager` role. Both roles were created in the "Creating a Role: Example":

```
GRANT warehouse_user TO dw_manager;
```

The `dw_manager` role now contains all of the privileges in the domain of the `warehouse_user` role.

Granting an Object Privilege on a User: Example

To grant the `INHERIT PRIVILEGES` object privilege on user `sh` to user `hr`, issue the following statement:

```
GRANT INHERIT PRIVILEGES ON USER sh TO hr;
```

Granting an Object Privilege on a Directory: Example

To grant `READ` on directory `bfile_dir` to user `hr`, with the `GRANT OPTION`, issue the following statement:

```
GRANT READ ON DIRECTORY bfile_dir TO hr
   WITH GRANT OPTION;
```

Granting Object Privileges on a Table to a User: Example

To grant all privileges on the table `oe.bonuses`, which was created in "Merging into a Table: Example", to the user `hr` with the `GRANT OPTION`, issue the following statement:

```
GRANT ALL ON bonuses TO hr
   WITH GRANT OPTION;
```

The user `hr` can subsequently perform the following operations:

- Exercise any privilege on the `bonuses` table
- Grant any privilege on the `bonuses` table to another user or role

Granting Object Privileges on a View: Example

To grant `SELECT` and `UPDATE` privileges on the view `emp_view`, which was created in "Creating a View: Example", to all users, issue the following statement:

```
GRANT SELECT, UPDATE
   ON emp_view TO PUBLIC;
```

All users can subsequently query and update the view of employee details.

Granting Object Privileges to a Sequence in Another Schema: Example

To grant `SELECT` privilege on the `customers_seq` sequence in the schema `oe` to the user `hr`, issue the following statement:

```
GRANT SELECT
   ON oe.customers_seq TO hr;
```

The user `hr` can subsequently generate the next value of the sequence with the following statement:

```
SELECT oe.customers_seq.NEXTVAL
   FROM DUAL;
```

Granting Multiple Object Privileges on Individual Columns: Example

To grant to user oe the REFERENCES privilege on the employee_id column and the UPDATE privilege on the employee_id, salary, and commission_pct columns of the employees table in the schema hr, issue the following statement:

```
GRANT REFERENCES (employee_id),
      UPDATE (employee_id, salary, commission_pct)
   ON hr.employees
   TO oe;
```

The user oe can subsequently update values of the employee_id, salary, and commission_pct columns. User oe can also define referential integrity constraints that refer to the employee_id column. However, because the GRANT statement lists only these columns, oe cannot perform operations on any of the other columns of the employees table.

For example, oe can create a table with a constraint:

```
CREATE TABLE dependent
   (dependno   NUMBER,
    dependname VARCHAR2(10),
    employee   NUMBER
   CONSTRAINT in_emp REFERENCES hr.employees(employee_id) );
```

The constraint in_emp ensures that all dependents in the dependent table correspond to an employee in the employees table in the schema hr.

INSERT

Purpose

Use the INSERT statement to add rows to a table, the base table of a view, a partition of a partitioned table or a subpartition of a composite-partitioned table, or an object table or the base table of an object view.

Prerequisites

For you to insert rows into a table, the table must be in your own schema or you must have the INSERT object privilege on the table.

For you to insert rows into the base table of a view, the owner of the schema containing the view must have the INSERT object privilege on the base table. Also, if the view is in a schema other than your own, then you must have the INSERT object privilege on the view.

If you have the INSERT ANY TABLE system privilege, then you can also insert rows into any table or the base table of any view.

You must also have the READ or SELECT object privilege on the table into which you want to insert rows if the table is on a remote database.

To specify the *returning_clause*, you must have the READ or SELECT object privilege on the object.

If the SQL92_SECURITY initialization parameter is set to TRUE and the INSERT operation references table columns, such as the columns in a *returning_clause*, then you must have the SELECT object privilege on the object into which you want to insert rows.

Conventional and Direct-Path INSERT

You can use the INSERT statement to insert data into a table, partition, or view in two ways: conventional INSERT and direct-path INSERT. When you issue a conventional INSERT statement,

Oracle Database reuses free space in the table into which you are inserting and maintains referential integrity constraints. With direct-path INSERT, the database appends the inserted data after existing data in the table. Data is written directly into data files, bypassing the buffer cache. Free space in the existing data is not reused. This alternative enhances performance during insert operations and is similar to the functionality of the Oracle direct-path loader utility, SQL*Loader. When you insert into a table that has been created in parallel mode, direct-path INSERT is the default.

The manner in which the database generates redo and undo data depends in part on whether you are using conventional or direct-path INSERT:

- Conventional INSERT always generates maximal redo and undo for changes to both data and metadata, regardless of the logging setting of the table and the archivelog and force logging settings of the database.
- Direct-path INSERT generates both redo and undo for *metadata* changes, because these are needed for operation recovery. For *data* changes, undo and redo are generated as follows:
  - Direct-path INSERT always bypasses undo generation for data changes.
  - If the database is not in ARCHIVELOG or FORCE LOGGING mode, then no redo is generated for data changes, regardless of the logging setting of the table.
  - If the database is in ARCHIVELOG mode (but not in FORCE LOGGING mode), then direct-path INSERT generates data redo for LOGGING tables but not for NOLOGGING tables.
  - If the database is in ARCHIVELOG **and** FORCE LOGGING mode, then direct-path SQL generate data redo for both LOGGING and NOLOGGING tables.

Direct-path INSERT is subject to a number of restrictions. If any of these restrictions is violated, then Oracle Database executes conventional INSERT serially without returning any message, unless otherwise noted:

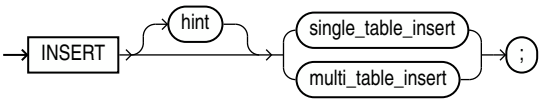
- You can have multiple direct-path INSERT statements in a single transaction, with or without other DML statements. However, after one DML statement alters a particular table, partition, or index, no other DML statement in the transaction can access that table, partition, or index.
- Queries that access the same table, partition, or index are allowed before the direct-path INSERT statement, but not after it.
- If any serial or parallel statement attempts to access a table that has already been modified by a direct-path INSERT in the same transaction, then the database returns an error and rejects the statement.
- The target table cannot be of a cluster.
- The target table cannot contain object type columns.
- Direct-path INSERT is not supported for an index-organized table (IOT) if it has a mapping table, or if it is reference by a materialized view.
- Direct-path INSERT into a single partition of an index-organized table (IOT), into a partitioned IOT with only one partition, or into an IOT that is not partitioned, will be done serially, even if the IOT was created in parallel mode or you specify the APPEND or APPEND_VALUES hint. However, direct-path INSERT operations into a partitioned IOT will honor parallel mode as long as the partition-extended name is not used and the IOT has more than one partition.
- The target table cannot have any triggers or referential integrity constraints defined on it.
- The target table cannot be replicated.

- A transaction containing a direct-path `INSERT` statement cannot be or become distributed.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for a more complete description of direct-path `INSERT`
> - *Oracle Database Utilities* for information on SQL*Loader
> - *Oracle Database SQL Tuning Guide* for information on statistics gathering when inserting into an empty table using direct-path `INSERT`

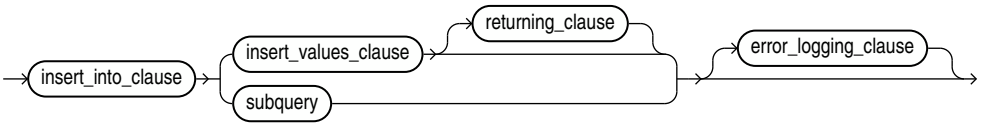
Syntax

***insert*::=**



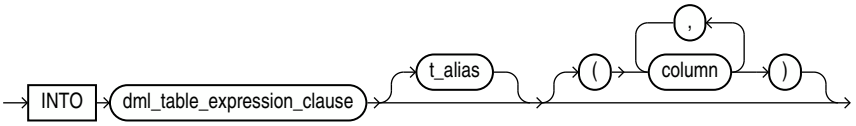
(*single_table_insert*::=, *multi_table_insert*::=)

***single_table_insert*::=**



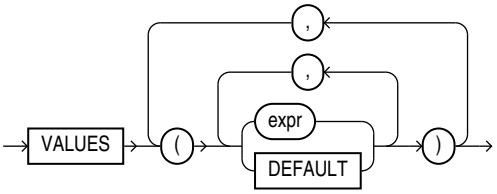
(*insert_into_clause*::=, *insert_values_clause*::=, *returning_clause*::=, *subquery*::=, *error_logging_clause*::=)

***insert_into_clause*::=**

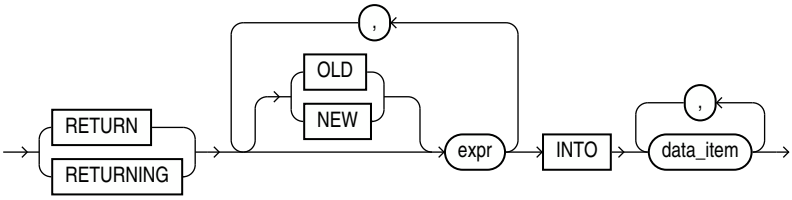


(*DML_table_expression_clause*::=)

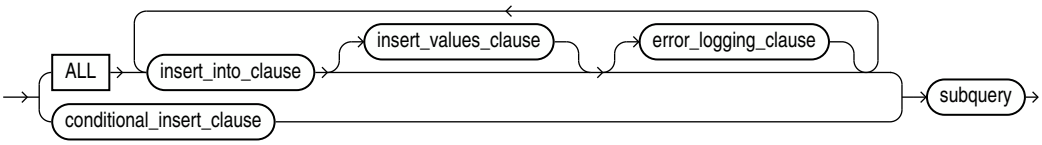
***insert_values_clause*::=**



***returning_clause*::=**

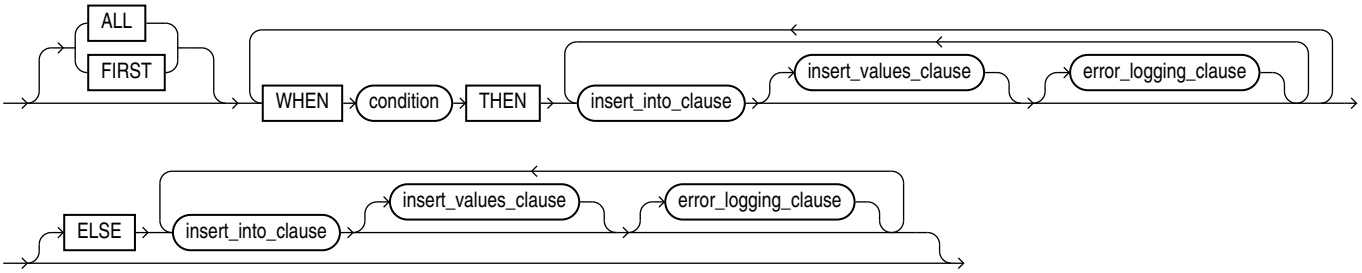


***multi_table_insert*::=**



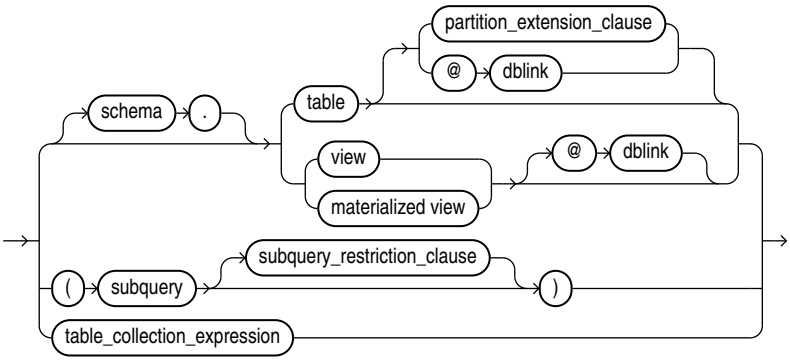
(*insert_into_clause*::=, *insert_values_clause*::=, *conditional_insert_clause*::=, *subquery*::=, *error_logging_clause*::=)

***conditional_insert_clause*::=**



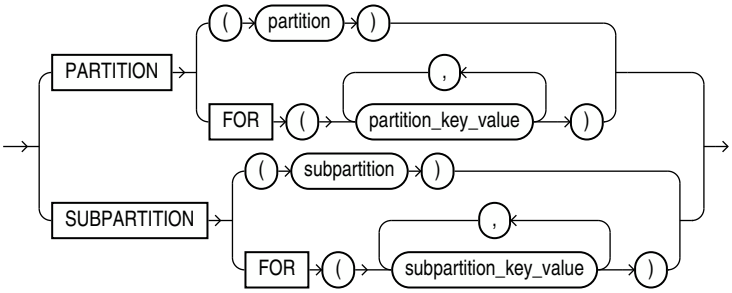
(*insert_into_clause*::=, *insert_values_clause*::=)

***DML_table_expression_clause*::=**

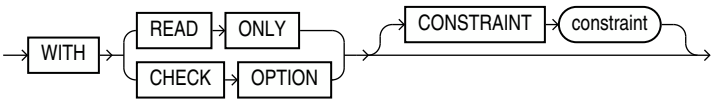


(*partition_extension_clause*::=, *subquery*::=—part of `SELECT`, *subquery_restriction_clause*::=, *table_collection_expression*::=)

***partition_extension_clause*::=**

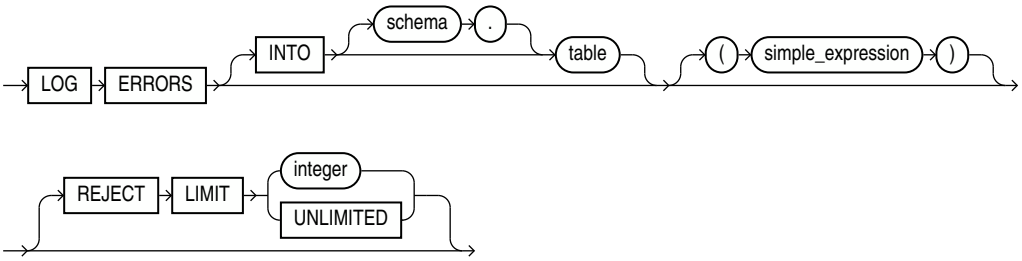


***subquery_restriction_clause*::=**



***table_collection_expression*::=**

***error_logging_clause*::=**



Semantics

hint

Specify a comment that passes instructions to the optimizer on choosing an execution plan for the statement.

For a multi-table insert, if you specify the `PARALLEL` hint for any target table, then the entire multi-table insert statement is parallelized even if the target tables have not been created or altered with `PARALLEL` specified. If you do not specify the `PARALLEL` hint, then the insert

operation will not be parallelized unless all target tables were created or altered with `PARALLEL` specified.

> **See Also:**
>
> - "Hints " for the syntax and description of hints
> - "Restrictions on Multi-Table Inserts"

single_table_insert

In a **single-table insert**, you insert values into one row of a table, view, or materialized view by specifying values explicitly or by retrieving the values through a subquery.

You can use the *`flashback_query_clause`* in *`subquery`* to insert past data into *`table`*. Refer to the *flashback_query_clause* of `SELECT` for more information on this clause.

Restriction on Single-Table Inserts

If you retrieve values through a subquery, then the select list of the subquery must have the same number of columns as the column list of the `INSERT` statement. If you omit the column list, then the subquery must provide values for every column in the table.

> **See Also:**
>
> "Inserting Values into Tables: Examples"

insert_into_clause

Use the `INSERT INTO` clause to specify the target object or objects into which the database is to insert data.

Directory-based sharding uses the same partition extension syntax as system-based sharding.

DML_table_expression_clause

Use the `INTO` *`DML_table_expression_clause`* to specify the objects into which data is being inserted.

schema

Specify the schema containing the table, view, or materialized view. If you omit *`schema`*, then the database assumes the object is in your own schema.

table | view | materialized_view | subquery

Specify the name of the table or object table, view or object view, materialized view, or the column or columns returned by a subquery, into which rows are to be inserted. If you specify a view or object view, then the database inserts rows into the base table of the view.

You cannot insert rows into a read-only materialized view. If you insert rows into a writable materialized view, then the database inserts the rows into the underlying container table. However, the insertions are overwritten at the next refresh operation. If you insert rows into an updatable materialized view that is part of a materialized view group, then the database also inserts the corresponding rows into the master table.

If any value to be inserted is a REF to an object table, and if the object table has a primary key object identifier, then the column into which you insert the REF must be a REF column with a referential integrity or SCOPE constraint to the object table.

If *table*, or the base table of *view*, contains one or more domain index columns, then this statement executes the appropriate indextype insert routine.

Issuing an INSERT statement against a table fires any INSERT triggers defined on the table.

> **See Also:**
>
> *Oracle Database Data Cartridge Developer's Guide* for more information on these routines

Restrictions on the *DML_table_expression_clause*

This clause is subject to the following restrictions:

- You cannot execute this statement if *table* or the base table of *view* contains any domain indexes marked IN_PROGRESS or FAILED.
- You cannot insert into a partition if any affected index partitions are marked UNUSABLE.
- With regard to the ORDER BY clause of the *subquery* in the *DML_table_expression_clause*, ordering is guaranteed only for the rows being inserted, and only within each extent of the table. Ordering of new rows with respect to existing rows is not guaranteed.
- If a view was created using the WITH CHECK OPTION, then you can insert into the view only rows that satisfy the defining query of the view.
- If a view was created using a single base table, then you can insert rows into the view and then retrieve those values using the *returning_clause*.
- You cannot insert rows into a view except with INSTEAD OF triggers if the defining query of the view contains one of the following constructs:

  A set operator
  A DISTINCT operator
  An aggregate or analytic function
  A GROUP BY, ORDER BY, MODEL, CONNECT BY, or START WITH clause
  A collection expression in a SELECT list
  A subquery in a SELECT list
  A subquery designated WITH READ ONLY
  Joins, with some exceptions, as documented in *Oracle Database Administrator's Guide*

- If you specify an index, index partition, or index subpartition that has been marked UNUSABLE, then the INSERT statement will fail unless the SKIP_UNUSABLE_INDEXES session parameter has been set to TRUE. Refer to ALTER SESSION for information on the SKIP_UNUSABLE_INDEXES session parameter.

partition_extension_clause

Specify the name or partition key value of the partition or subpartition within *table*, or the base table of *view*, targeted for inserts.

If a row to be inserted does not map into a specified partition or subpartition, then the database returns an error.

Restriction on Target Partitions and Subpartitions

This clause is not valid for object tables or object views.

> **See Also:**
>
> "References to Partitioned Tables and Indexes "

dblink

Specify a complete or partial name of a database link to a remote database where the table or view is located.

You can insert rows into a remote table or view only if you are using Oracle Database distributed functionality.

To insert rows into a remote table you must have both `INSERT` and `SELECT` privileges on the table.

If you omit *`dblink`*, then Oracle Database assumes that the table or view is on the local database. You can insert rows into a local table with just the `INSERT` privilege.

> **Note:**
>
> Starting with Oracle Database 12*c* Release 2 (12.2), the `INSERT` statement accepts remote LOB locators as bind variables. Refer to the “Distributed LOBs” chapter in *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information.

> **See Also:**
>
> - "Syntax for Schema Objects and Parts in SQL Statements" and "References to Objects in Remote Databases " for information on referring to database links
> - "Inserting into a Remote Database: Example"

subquery_restriction_clause

Use the *`subquery_restriction_clause`* to restrict the subquery in one of the following ways:

WITH READ ONLY

Specify `WITH READ ONLY` to indicate that the table or view cannot be updated.

WITH CHECK OPTION

Specify `WITH CHECK OPTION` to indicate that Oracle Database prohibits any changes to the table or view that would produce rows that are not included in the subquery. When used in the

subquery of a DML statement, you can specify this clause in a subquery in the `FROM` clause but not in subquery in the `WHERE` clause.

CONSTRAINT *constraint*

Specify the name of the `CHECK OPTION` constraint. If you omit this identifier, then Oracle automatically assigns the constraint a name of the form `SYS_Cn`, where n is an integer that makes the constraint name unique within the database.

> **See Also:**
>
> "Using the WITH CHECK OPTION Clause: Example"

table_collection_expression

The *`table_collection_expression`* lets you inform Oracle that the value of *`collection_expression`* should be treated as a table for purposes of query and DML operations. The *`collection_expression`* can be a subquery, a column, a function, or a collection constructor. Regardless of its form, it must return a collection value—that is, a value whose type is nested table or varray. This process of extracting the elements of a collection is called **collection unnesting**.

The optional plus (+) is relevant if you are joining the `TABLE` collection expression with the parent table. The + creates an outer join of the two, so that the query returns rows from the outer table even if the collection expression is null.

> **Note:**
>
> In earlier releases of Oracle, when *`collection_expression`* was a subquery, *`table_collection_expression`* was expressed as `THE` *`subquery`*. That usage is now deprecated.

> **See Also:**
>
> "Table Collections: Examples"

t_alias

Specify a **correlation name**, which is an alias for the table, view, materialized view, or subquery to be referenced elsewhere in the statement.

Restriction on Table Aliases

You cannot specify *`t_alias`* during a multi-table insert.

column

Specify a column of the table, view, or materialized view. In the inserted row, each column in this list is assigned a value from the *`insert_values_clause`* or the subquery. If you want to assign a value to an `INVISIBLE` column, then you must include the column in this list.

If you omit one or more of the table's columns from this list, then the column value of that column for the inserted row is the column default value as specified when the table was created or last altered. If any omitted column has a `NOT NULL` constraint and no default value, then the database returns an error indicating that the constraint has been violated and rolls back the `INSERT` statement. Refer to CREATE TABLE for more information on default column values.

If you omit the column list altogether, then the *`insert_values_clause`* or query must specify values for all columns in the table.

insert_values_clause

For a **single-table insert** operation, specify a row of values to be inserted into the table or view. You must specify a value in the *`insert_values_clause`* for each column in the column list. If you omit the column list, then the *`insert_values_clause`* must provide values for every column in the table.

You can only supply one set of values for each *`insert_into_clause`* in a **multi-table insert** operation.

In a **multi-table insert** operation, each expression in the *`insert_values_clause`* must refer to columns returned by the select list of the subquery. If you omit the *`insert_values_clause`*, then the select list of the subquery determines the values to be inserted, so it must have the same number of columns as the column list of the corresponding *`insert_into_clause`*. If you do not specify a column list in the *`insert_into_clause`*, then the computed row must provide values for all columns in the target table.

For both types of insert operations, if you specify a column list in the *`insert_into_clause`*, then the database assigns to each column in the list a corresponding value from the values clause or the subquery. You can specify `DEFAULT` for any value in the *`insert_values_clause`*. If you have specified a default value for the corresponding column of the table or view, then that value is inserted. If no default value for the corresponding column has been specified, then the database inserts null. Refer to "About SQL Expressions " and SELECT for syntax of valid expressions.

Restrictions on Inserted Values

Values are subject to the following restrictions:

- You cannot insert a `BFILE` value until you have initialized the `BFILE` locator to null or to a directory name and filename.
- When inserting into a list-partitioned table, you cannot insert a value into the partitioning key column that does not already exist in the *`partition_key_value`* list of one of the partitions.
- You cannot specify `DEFAULT` when inserting into a view.
- If you insert string literals into a `RAW` column, then during subsequent queries Oracle Database will perform a full table scan rather than using any index that might exist on the `RAW` column.
- You cannot use default values for columns in row value expressions.

  Inserting multiple default values into an identity column using the table value constructor results in the following error : `Sequence as default is not supported with INSERT using table value constructor .`

> **See Also:**
>
> - BFILENAME for information on initializing `BFILE` values and for an example of inserting into a `BFILE`
> - *Oracle Database SecureFiles and Large Objects Developer's Guide* for information on initializing `BFILE` locators
> - "Using XML in SQL Statements " for information on inserting values into an XMLType table
> - "Inserting into a Substitutable Tables and Columns: Examples", "Inserting Using the TO_LOB Function: Example", "Inserting Sequence Values: Example", and "Inserting Using Bind Variables: Example"

returning_clause

The returning clause retrieves the rows affected by a DML statement. You can specify this clause for tables and materialized views and for views with a single base table.

When operating on a single row, a DML statement with a *`returning_clause`* can retrieve column expressions using the affected row, rowid, and `REF`s to the affected row and store them in host variables or PL/SQL variables.

When operating on multiple rows, a DML statement with the *`returning_clause`* stores values from expressions, rowids, and `REF`s involving the affected rows in bind arrays.

expr

Each item in the *`expr`* list must be a valid expression syntax.

INTO

The `INTO` clause indicates that the values of the changed rows are to be stored in the variable(s) specified in *`data_item`* list.

data_item

Each *`data_item`* is a host variable or PL/SQL variable that stores the retrieved *`expr`* value.

For each expression in the `RETURNING` list, you must specify a corresponding type-compatible PL/SQL variable or host variable in the `INTO` list.

Restrictions

The following restrictions apply to the `RETURNING` clause:

- The *`expr`* is restricted as follows:
  - For `UPDATE` and `DELETE` statements each *`expr`* must be a simple expression or a single-set aggregate function expression. You cannot combine simple expressions and single-set aggregate function expressions in the same *`returning_clause`*. For `INSERT` statements, each *`expr`* must be a simple expression. Aggregate functions are not supported in an `INSERT` statement `RETURNING` clause.
  - Single-set aggregate function expressions cannot include the `DISTINCT` keyword.
- If the *`expr`* list contains a primary key column or other `NOT NULL` column, then the update statement fails if the table has a `BEFORE UPDATE` trigger defined on it.

- You cannot specify the *returning_clause* for a multi-table insert.
- You cannot use this clause with parallel DML or with remote objects.
- You cannot retrieve LONG types with this clause.
- You cannot specify this clause for a view on which an INSTEAD OF trigger has been defined.

> **See Also:**
>
> *Oracle Database PL/SQL Language Reference* for information on using the BULK COLLECT clause to return multiple values to collection variables

multi_table_insert

In a **multi-table insert**, you insert rows returned from the evaluation of a subquery into one or more tables.

Table aliases are not defined by the select list of the subquery. Therefore, they are not visible in the clauses dependent on the select list. For example, this can happen when trying to refer to an object column in an expression. To use an expression with a table alias, you must put the expression into the select list with a column alias, and then refer to the column alias in the VALUES clause or WHEN condition of the multi-table insert.

ALL *into_clause*

Specify ALL followed by multiple *insert_into_clauses* to perform an **unconditional multi-table insert**. Oracle Database executes each *insert_into_clause* once for each row returned by the subquery.

Restrictions Using INSERT ALL

- The maximum number of rows you can insert into a table with 4096 columns is 15.
- The maximum number of rows you can insert into a table with 1000 columns is 65.
- The maximum row limit for tables with fewer columns is not constant and may vary.

conditional_insert_clause

Specify the *conditional_insert_clause* to perform a **conditional multi-table insert**.

You can only supply one set of values for each WHEN or ELSE clause of *conditional_insert_clause*.

Oracle Database filters each *insert_into_clause* through the corresponding WHEN condition, which determines whether that *insert_into_clause* is executed. Each expression in the WHEN condition must refer to columns returned by the select list of the subquery.A single multi-table insert statement can contain up to 127 WHEN clauses.

ALL

If you specify ALL, the default value, then the database evaluates each WHEN clause regardless of the results of the evaluation of any other WHEN clause. For each WHEN clause whose condition evaluates to true, the database executes the corresponding INTO clause list.

FIRST

If you specify FIRST, then the database evaluates each WHEN clause in the order in which it appears in the statement. For the first WHEN clause that evaluates to true, the database executes the corresponding INTO clause and skips subsequent WHEN clauses for the given row.

ELSE clause

For a given row, if no WHEN clause evaluates to true, then:

- If you have specified an ELSE clause, then the database executes the INTO clause list associated with the ELSE clause.
- If you did not specify an else clause, then the database takes no action for that row.

> **See Also:**
>
> "Multi-Table Inserts: Examples"

Restrictions on Multi-Table Inserts

multi-table inserts are subject to the following restrictions:

- You can perform multi-table inserts only on tables, not on views or materialized views.
- You cannot perform a multi-table insert into a remote table.
- You cannot specify a TABLE collection expression when performing a multi-table insert.
- multi-table inserts are not parallelized if any target table is index organized or if any target table has a bitmap index defined on it.
- Plan stability is not supported for multi-table insert statements.
- The subquery of the multitable insert statement cannot use a sequence. For rules pertaining to sequences, see How to Use Sequence Values

subquery

Specify a subquery that returns rows that are inserted into the table. The subquery can refer to any table, view, or materialized view, including the target tables of the INSERT statement. If the subquery selects no rows, then the database inserts no rows into the table.

You can use *subquery* in combination with the TO_LOB function to convert the values in a LONG column to LOB values in another column in the same or another table.

- To migrate LONG values to LOB values in another column in a view, you must perform the migration on the base table and then add the LOB column to the view.
- To migrate LONG values on a remote table to LOB values in a local table, you must perform the migration on the remote table using the TO_LOB function, and then perform an INSERT ... *subquery* operation to copy the LOB values from the remote table into the local table.

Notes on Inserting with a Subquery

The following notes apply when inserting with a subquery:

- If *subquery* returns the partial or total equivalent of a materialized view, then the database may use the materialized view for query rewrite in place of one or more tables specified in *subquery*.

> **See Also:**
>
> *Oracle Database Data Warehousing Guide* for more information on materialized views and query rewrite

- If *subquery* refers to remote objects, then the INSERT operation can run in parallel as long as the reference does not loop back to an object on the local database. However, if the *subquery* in the *DML_table_expression_clause* refers to any remote objects, then the INSERT operation will run serially without notification. See *parallel_clause* for more information.
- If *subquery* includes an ORDER BY clause, then it will override row ordering specified using attribute clustering table properties.

> **See Also:**
>
> - "Inserting Values with a Subquery: Example"
> - BFILENAME for an example of inserting into a BFILE
> - *Oracle Database SecureFiles and Large Objects Developer's Guide* for information on initializing BFILEs
> - "About SQL Expressions " and SELECT for syntax of valid expressions

error_logging_clause

The *error_logging_clause* lets you capture DML errors and the log column values of the affected rows and save them in an error logging table.

INTO *table*

Specify the name of the error logging table. If you omit this clause, then the database assigns the default name generated by the DBMS_ERRLOG package. The default error log table name is ERR$_ followed by the first 25 characters of the name of the table upon which the DML operation is being executed.

simple_expression

Specify the value to be used as a statement tag, so that you can identify the errors from this statement in the error logging table. The expression can be either a text literal, a number literal, or a general SQL expression such as a bind variable. You can also use a function expression if you convert it to a text literal — for example, TO_CHAR(SYSDATE).

REJECT LIMIT

This clause lets you specify an integer as an upper limit for the number of errors to be logged before the statement terminates and rolls back any changes made by the statement. The default rejection limit is zero. For parallel DML operations, the reject limit is applied to each parallel server.

Restrictions on DML Error Logging

- The following conditions cause the statement to fail and roll back without invoking the error logging capability:

  - Violated deferred constraints.
  - Any direct-path `INSERT` or `MERGE` operation that raises a unique constraint or index violation.
  - Any update operation `UPDATE` or `MERGE` that raises a unique constraint or index violation.
- You cannot track errors in the error logging table for `LONG`, LOB, or object type columns. However, the table that is the target of the DML operation can contain these types of columns.
  - If you create or modify the corresponding error logging table so that it contains a column of an unsupported type, and if the name of that column corresponds to an unsupported column in the target DML table, then the DML statement fails at parse time.
  - If the error logging table does not contain any unsupported column types, then all DML errors are logged until the reject limit of errors is reached. For rows on which errors occur, column values with corresponding columns in the error logging table are logged along with the control information.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information on using the `create_error_log` procedure of the `DBMS_ERRLOG` package and *Oracle Database Administrator's Guide* for general information on DML error logging.
> - "Inserting Into a Table with Error Logging: Example"

Examples

Inserting Values into Tables: Examples

The following statement inserts a row into the sample table `departments`:

```
INSERT INTO departments
   VALUES (280, 'Recreation', 121, 1700);
```

If the `departments` table had been created with a default value of 121 for the `manager_id` column, then you could issue the same statement as follows:

```
INSERT INTO departments
   VALUES (280, 'Recreation', DEFAULT, 1700);
```

The following statement inserts a row with six columns into the `employees` table. One of these columns is assigned `NULL` and another is assigned a number in scientific notation:

```
INSERT INTO employees (employee_id, last_name, email,
      hire_date, job_id, salary, commission_pct)
   VALUES (207, 'Gregory', 'pgregory@example.com',
      sysdate, 'PU_CLERK', 1.2E3, NULL);
```

The following statement has the same effect as the preceding example, but uses a subquery in the *`DML_table_expression_clause`*:

```
INSERT INTO
   (SELECT employee_id, last_name, email, hire_date, job_id,
      salary, commission_pct FROM employees)
```

```
   VALUES (207, 'Gregory', 'pgregory@example.com',
      sysdate, 'PU_CLERK', 1.2E3, NULL);
```

Inserting Values with a Subquery: Example

The following statement copies employees whose commission exceeds 25% of their salary into the `bonuses` table, which was created in "Merging into a Table: Example":

```
INSERT INTO bonuses
   SELECT employee_id, salary*1.1
   FROM employees
   WHERE commission_pct > 0.25;
```

Inserting Into a Table with Error Logging: Example

The following statements create a `raises` table in the sample schema `hr`, create an error logging table using the `DBMS_ERRLOG` package, and populate the `raises` table with data from the `employees` table. One of the inserts violates the check constraint on `raises`, and that row can be seen in `errlog`. If more than ten errors had occurred, then the statement would have aborted, rolling back any insertions made:

```
CREATE TABLE raises (emp_id NUMBER, sal NUMBER
   CONSTRAINT check_sal CHECK(sal > 8000));

EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('raises', 'errlog');

INSERT INTO raises
   SELECT employee_id, salary*1.1 FROM employees
   WHERE commission_pct > .2
   LOG ERRORS INTO errlog ('my_bad') REJECT LIMIT 10;

SELECT ORA_ERR_MESG$, ORA_ERR_TAG$, emp_id, sal FROM errlog;

ORA_ERR_MESG$               ORA_ERR_TAG$         EMP_ID SAL
--------------------------- -------------------- ------ -------
ORA-02290: check constraint my_bad               161    7700
 (HR.SYS_C004266) violated
```

Inserting into a Remote Database: Example

The following statement inserts a row into the `employees` table owned by the user `hr` on the database accessible by the database link `remote`:

```
INSERT INTO employees@remote
   VALUES (8002, 'Juan', 'Fernandez', 'juanf@example.com', NULL,
   TO_DATE('04-OCT-1992', 'DD-MON-YYYY'), 'SH_CLERK', 3000,
   NULL, 121, 20);
```

Inserting Sequence Values: Example

The following statement inserts a new row containing the next value of the `departments_seq` sequence into the `departments` table:

```
INSERT INTO departments
   VALUES  (departments_seq.nextval, 'Entertainment', 162, 1400);
```

Inserting Using Bind Variables: Example

The following example returns the values of the inserted rows into output bind variables `:bnd1` and `:bnd2`. The bind variables must first be declared.

```
INSERT INTO employees
      (employee_id, last_name, email, hire_date, job_id, salary)
   VALUES
   (employees_seq.nextval, 'Doe', 'john.doe@example.com',
       SYSDATE, 'SH_CLERK', 2400)
   RETURNING salary*12, job_id INTO :bnd1, :bnd2;
```

Inserting into a Substitutable Tables and Columns: Examples

The following example inserts into the `persons` table, which is created in "Substitutable Table and Column Examples". The first statement uses the root type `person_t`. The second insert uses the `employee_t` subtype of `person_t`, and the third insert uses the `part_time_emp_t` subtype of `employee_t`:

```
INSERT INTO persons VALUES (person_t('Bob', 1234));
INSERT INTO persons VALUES (employee_t('Joe', 32456, 12, 100000));
INSERT INTO persons VALUES (
   part_time_emp_t('Tim', 5678, 13, 1000, 20));
```

The following example inserts into the `books` table, which was created in "Substitutable Table and Column Examples". Notice that specification of the attribute values is identical to that for the substitutable table example:

```
INSERT INTO books VALUES (
   'An Autobiography', person_t('Bob', 1234));
INSERT INTO books VALUES (
   'Business Rules', employee_t('Joe', 3456, 12, 10000));
INSERT INTO books VALUES (
   'Mixing School and Work',
   part_time_emp_t('Tim', 5678, 13, 1000, 20));
```

You can extract data from substitutable tables and columns using built-in functions and conditions. For examples, see the functions TREAT and SYS_TYPEID , and "IS OF *type* Condition ".

Inserting Using the TO_LOB Function: Example

The following example copies `LONG` data to a LOB column in the following `long_tab` table:

```
CREATE TABLE long_tab (pic_id NUMBER, long_pics LONG RAW);
```

First you must create a table with a LOB.

```
CREATE TABLE lob_tab (pic_id NUMBER, lob_pics BLOB);
```

Next, use an `INSERT ... SELECT` statement to copy the data in all rows for the `LONG` column into the newly created LOB column:

```
INSERT INTO lob_tab
   SELECT pic_id, TO_LOB(long_pics) FROM long_tab;
```

When you are confident that the migration has been successful, you can drop the `long_pics` table. Alternatively, if the table contains other columns, then you can simply drop the `LONG` column from the table as follows:

```
ALTER TABLE long_tab DROP COLUMN long_pics;
```

Multi-Table Inserts: Examples

The following example uses the multi-table insert syntax to insert into the sample table `sh.sales` some data from an input table with a different structure.

> **Note:**
>
> A number of `NOT NULL` constraints on the `sales` table have been disabled for purposes of this example, because the example ignores a number of table columns for the sake of brevity.

The input table looks like this:

```
SELECT * FROM sales_input_table;

PRODUCT_ID CUSTOMER_ID WEEKLY_ST  SALES_SUN  SALES_MON  SALES_TUE  SALES_WED SALES_THU  SALES_FRI  SALES_SAT
---------- ----------- --------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
       111         222 01-OCT-00        100        200        300        400        500        600        700
       222         333 08-OCT-00        200        300        400        500        600        700        800
       333         444 15-OCT-00        300        400        500        600        700        800        900
```

The multi-table insert statement looks like this:

```
INSERT ALL
      INTO sales (prod_id, cust_id, time_id, amount)
      VALUES (product_id, customer_id, weekly_start_date, sales_sun)
      INTO sales (prod_id, cust_id, time_id, amount)
      VALUES (product_id, customer_id, weekly_start_date+1, sales_mon)
      INTO sales (prod_id, cust_id, time_id, amount)
      VALUES (product_id, customer_id, weekly_start_date+2, sales_tue)
      INTO sales (prod_id, cust_id, time_id, amount)
      VALUES (product_id, customer_id, weekly_start_date+3, sales_wed)
      INTO sales (prod_id, cust_id, time_id, amount)
      VALUES (product_id, customer_id, weekly_start_date+4, sales_thu)
      INTO sales (prod_id, cust_id, time_id, amount)
      VALUES (product_id, customer_id, weekly_start_date+5, sales_fri)
      INTO sales (prod_id, cust_id, time_id, amount)
      VALUES (product_id, customer_id, weekly_start_date+6, sales_sat)
   SELECT product_id, customer_id, weekly_start_date, sales_sun,
      sales_mon, sales_tue, sales_wed, sales_thu, sales_fri, sales_sat
      FROM sales_input_table;
```

Assuming these are the only rows in the `sales` table, the contents now look like this:

```
SELECT * FROM sales
   ORDER BY prod_id, cust_id, time_id;

   PROD_ID    CUST_ID TIME_ID   C   PROMO_ID QUANTITY_SOLD     AMOUNT       COST
---------- ---------- --------- - ---------- ------------- ---------- ----------
       111        222 01-OCT-00                                  100
       111        222 02-OCT-00                                  200
       111        222 03-OCT-00                                  300
       111        222 04-OCT-00                                  400
       111        222 05-OCT-00                                  500
       111        222 06-OCT-00                                  600
       111        222 07-OCT-00                                  700
       222        333 08-OCT-00                                  200
       222        333 09-OCT-00                                  300
       222        333 10-OCT-00                                  400
       222        333 11-OCT-00                                  500
       222        333 12-OCT-00                                  600
       222        333 13-OCT-00                                  700
       222        333 14-OCT-00                                  800
       333        444 15-OCT-00                                  300
```

```
333        444 16-OCT-00                                  400
333        444 17-OCT-00                                  500
333        444 18-OCT-00                                  600
333        444 19-OCT-00                                  700
333        444 20-OCT-00                                  800
333        444 21-OCT-00                                  900
```

The next examples insert into multiple tables. Suppose you want to provide to sales representatives some information on orders of various sizes. The following example creates tables for small, medium, large, and special orders and populates those tables with data from the sample table `oe.orders`:

```
CREATE TABLE small_orders
   (order_id       NUMBER(12)    NOT NULL,
    customer_id    NUMBER(6)     NOT NULL,
    order_total    NUMBER(8,2),
    sales_rep_id   NUMBER(6)
   );

CREATE TABLE medium_orders AS SELECT * FROM small_orders;

CREATE TABLE large_orders AS SELECT * FROM small_orders;

CREATE TABLE special_orders
   (order_id       NUMBER(12)     NOT NULL,
    customer_id    NUMBER(6)      NOT NULL,
    order_total    NUMBER(8,2),
    sales_rep_id   NUMBER(6),
    credit_limit   NUMBER(9,2),
    cust_email     VARCHAR2(40)
   );
```

The first multi-table insert populates only the tables for small, medium, and large orders:

```
INSERT ALL
   WHEN order_total <= 100000 THEN
      INTO small_orders
   WHEN order_total > 1000000 AND order_total <= 200000 THEN
      INTO medium_orders
   WHEN order_total > 200000 THEN
      INTO large_orders
   SELECT order_id, order_total, sales_rep_id, customer_id
      FROM orders;
```

You can accomplish the same thing using the `ELSE` clause in place of the insert into the `large_orders` table:

```
INSERT ALL
   WHEN order_total <= 100000 THEN
      INTO small_orders
   WHEN order_total > 100000 AND order_total <= 200000 THEN
      INTO medium_orders
   ELSE
      INTO large_orders
   SELECT order_id, order_total, sales_rep_id, customer_id
      FROM orders;
```

The next example inserts into the small, medium, and large tables, as in the preceding example, and also puts orders greater than 290,000 into the `special_orders` table. This table also shows how to use column aliases to simplify the statement:

```
INSERT ALL
   WHEN ottl <= 100000 THEN
      INTO small_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 100000 and ottl <= 200000 THEN
      INTO medium_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 200000 THEN
      into large_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 290000 THEN
      INTO special_orders
   SELECT o.order_id oid, o.customer_id cid, o.order_total ottl,
      o.sales_rep_id sid, c.credit_limit cl, c.cust_email cem
      FROM orders o, customers c
      WHERE o.customer_id = c.customer_id;
```

Finally, the next example uses the `FIRST` clause to put orders greater than 290,000 into the `special_orders` table and exclude those orders from the `large_orders` table:

```
INSERT FIRST
   WHEN ottl <= 100000 THEN
      INTO small_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 100000 and ottl <= 200000 THEN
      INTO medium_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 290000 THEN
      INTO special_orders
   WHEN ottl > 200000 THEN
      INTO large_orders
         VALUES(oid, ottl, sid, cid)
   SELECT o.order_id oid, o.customer_id cid, o.order_total ottl,
      o.sales_rep_id sid, c.credit_limit cl, c.cust_email cem
      FROM orders o, customers c
      WHERE o.customer_id = c.customer_id;
```

Inserting Multiple Rows Using a Single Statement: Example

The following statements create three tables named people, patients and staff:

```
CREATE TABLE people (
  person_id   INTEGER NOT NULL PRIMARY KEY,
  given_name  VARCHAR2(100) NOT NULL,
  family_name VARCHAR2(100) NOT NULL,
  title       VARCHAR2(20),
  birth_date  DATE
);

CREATE TABLE patients (
  patient_id          INTEGER NOT NULL PRIMARY KEY REFERENCES people (person_id),
  last_admission_date DATE
);

CREATE TABLE staff (
  staff_id   INTEGER NOT NULL PRIMARY KEY REFERENCES people (person_id),
  hired_date DATE
);
```

The following statement inserts a row into the people table:

```
INSERT INTO people
VALUES (1, 'Dave', 'Badger', 'Mr', date'1960-05-01');
```

The following statement returns an error as there is no value provided for the birth_date column:

```
INSERT INTO people
VALUES (2, 'Simon', 'Fox', 'Mr');
```

The following statement inserts a row into the people table:

```
INSERT INTO people (person_id, given_name, family_name, title)
VALUES (2, 'Simon', 'Fox', 'Mr');
```

The following statement inserts a row into the people table and the value for the title column is populated by selecting a static value from the dual table:

```
INSERT INTO people (person_id, given_name, family_name, title)
VALUES (3, 'Dave', 'Frog', (SELECT 'Mr' FROM dual));
```

The following statement inserts multiple rows into the people table using ‘SELECT’ statement:

```
INSERT INTO people (person_id, given_name, family_name, title)
  WITH names AS (
    SELECT 4, 'Ruth',     'Fox',      'Mrs'    FROM dual UNION ALL
    SELECT 5, 'Isabelle', 'Squirrel', 'Miss'   FROM dual UNION ALL
    SELECT 6, 'Justin',   'Frog',     'Master' FROM dual UNION ALL
    SELECT 7, 'Lisa',     'Owl',      'Dr'     FROM dual
  )
  SELECT * FROM names;
```

The following statement rolls back all the previous DML operations:

```
ROLLBACK;
```

The following statement inserts multiple rows into the people table using ‘SELECT’ statement with a ‘WHERE’ condition:

```
INSERT INTO people (person_id, given_name, family_name, title)
  WITH names AS (
    SELECT 4, 'Ruth',     'Fox' family_name,      'Mrs'    FROM dual UNION ALL
    SELECT 5, 'Isabelle', 'Squirrel' family_name, 'Miss'   FROM dual UNION ALL
    SELECT 6, 'Justin',   'Frog' family_name,     'Master' FROM dual UNION ALL
    SELECT 7, 'Lisa',     'Owl' family_name,      'Dr'     FROM dual
  )
  SELECT * FROM names
  WHERE  family_name LIKE 'F%';
```

The following statement rolls back all the previous DML operations:

```
ROLLBACK;
```

The following statement inserts multiple rows into people, patients and staff table using ‘INSERT ALL’ statement:

```
INSERT ALL
  /* Every one is a person */
  INTO people (person_id, given_name, family_name, title)
    VALUES (id, given_name, family_name, title)
  INTO patients (patient_id, last_admission_date)
    VALUES (id, admission_date)
  INTO staff (staff_id, hired_date)
    VALUES (id, hired_date)
  WITH names AS (
    SELECT 4 id, 'Ruth' given_name, 'Fox' family_name, 'Mrs' title,
           NULL hired_date, DATE'2009-12-31' admission_date
```

```
  FROM   dual UNION ALL
  SELECT 5 id, 'Isabelle' given_name, 'Squirrel' family_name, 'Miss' title ,
         NULL hired_date, DATE'2014-01-01' admission_date
  FROM   dual UNION ALL
  SELECT 6 id, 'Justin' given_name, 'Frog' family_name, 'Master' title,
         NULL hired_date, DATE'2015-04-22' admission_date
  FROM   dual UNION ALL
  SELECT 7 id, 'Lisa' given_name, 'Owl' family_name, 'Dr' title,
         DATE'2015-01-01' hired_date, NULL admission_date
  FROM   dual
)
SELECT * FROM names;
```

The following statement rolls back all the previous DML operations:

```
ROLLBACK;
```

The following statement inserts multiple rows into people, patients and staff table using 'INSERT ALL' statement with various conditions:

```
INSERT ALL
  /* Everyone is a person, so insert all rows into people */
  WHEN 1=1 THEN
    INTO people (person_id, given_name, family_name, title)
    VALUES (id, given_name, family_name, title)
  /* Only people with an admission date are patients */
  WHEN admission_date IS NOT NULL THEN
    INTO patients (patient_id, last_admission_date)
    VALUES (id, admission_date)
  /* Only people with a hired date are staff */
  WHEN hired_date IS NOT NULL THEN
    INTO staff (staff_id, hired_date)
    VALUES (id, hired_date)
  WITH names AS (
    SELECT 4 id, 'Ruth' given_name, 'Fox' family_name, 'Mrs' title,
           NULL hired_date, DATE'2009-12-31' admission_date
    FROM   dual UNION ALL
    SELECT 5 id, 'Isabelle' given_name, 'Squirrel' family_name, 'Miss' title ,
           NULL hired_date, DATE'2014-01-01' admission_date
    FROM   dual UNION ALL
    SELECT 6 id, 'Justin' given_name, 'Frog' family_name, 'Master' title,
           NULL hired_date, DATE'2015-04-22' admission_date
    FROM   dual UNION ALL
    SELECT 7 id, 'Lisa' given_name, 'Owl' family_name, 'Dr' title,
           DATE'2015-01-01' hired_date, NULL admission_date
    FROM   dual
  )
  SELECT * FROM names;
```

LOCK TABLE

Purpose

Use the `LOCK TABLE` statement to lock one or more tables, table partitions, or table subpartitions in a specified mode. This lock manually overrides automatic locking and permits or denies access to a table or view by other users for the duration of your operation.

Some forms of locks can be placed on the same table at the same time. Other locks allow only one lock for a table.

A locked table remains locked until you either commit your transaction or roll it back, either entirely or to a savepoint before you locked the table.

A lock never prevents other users from querying the table. A query never places a lock on a table. Readers never block writers and writers never block readers.

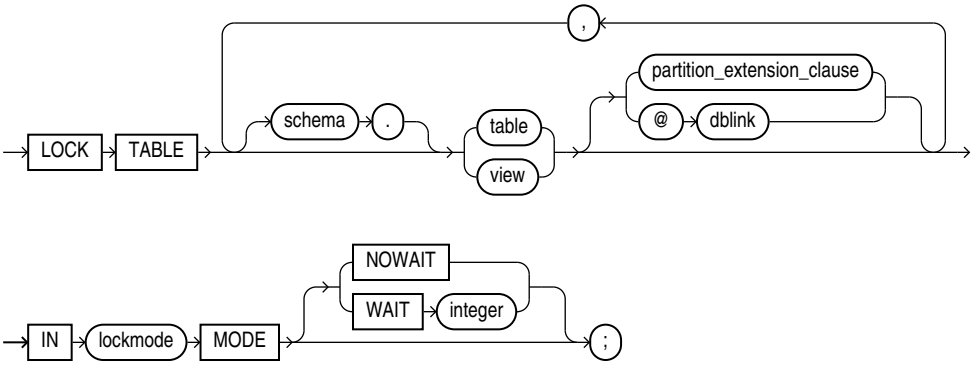
> **See Also:**
>
> - *Oracle Database Concepts* for a complete description of the interaction of lock modes
> - COMMIT
> - ROLLBACK
> - SAVEPOINT

Prerequisites

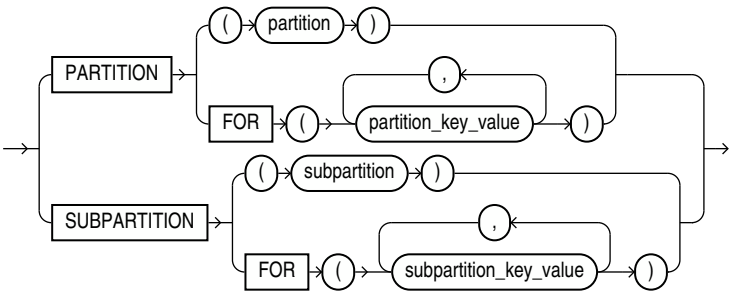
The table or view must be in your own schema, or you must have the `LOCK ANY TABLE` system privilege, or you must have any object privilege (except the `READ` object privilege) on the table or view.

Syntax

***lock_table*::=**



***partition_extension_clause*::=**



Semantics

schema

Specify the schema containing the table or view. If you omit *schema*, then Oracle Database assumes the table or view is in your own schema.

table | *view*

Specify the name of the table or view to be locked.

If you specify *view*, then Oracle Database locks the base tables of the view.

If you specify the *partition_extension_clause*, then Oracle Database first acquires an implicit lock on the table. The table lock is the same as the lock you specify for the partition or subpartition, with two exceptions:

- If you specify a SHARE lock for the subpartition, then the database acquires an implicit ROW SHARE lock on the table.
- If you specify an EXCLUSIVE lock for the subpartition, then the database acquires an implicit ROW EXCLUSIVE lock on the table.

If you specify PARTITION and *table* is composite-partitioned, then the database acquires locks on all the subpartitions of the partition.

Restrictions on Locking Tables

The following restrictions apply to locking tables:

- If *view* is part of a hierarchy, then it must be the root of the hierarchy.
- You can acquire locks on only the existing partitions in an automatic list-partitioned table. That is, when you specify the following statement, the partition key value must correspond to a partition that already exists in the table; it cannot correspond to a partition that might be created on-demand at a later time:

  ```
  LOCK TABLE ... PARTITION FOR (partition_key_value) ...
  ```

dblink

Specify a database link to a remote Oracle Database where the table or view is located. You can lock tables and views on a remote database only if you are using Oracle distributed functionality. All tables locked by a LOCK TABLE statement must be on the same database.

If you omit *dblink*, then Oracle Database assumes the table or view is on the local database.

> **See Also:**
>
> "References to Objects in Remote Databases " for information on specifying database links

lockmode Clause

Specify one of the following modes:

ROW SHARE

`ROW SHARE` permits concurrent access to the locked table but prohibits users from locking the entire table for exclusive access. `ROW SHARE` is synonymous with `SHARE UPDATE`, which is included for compatibility with earlier versions of Oracle Database.

ROW EXCLUSIVE

`ROW EXCLUSIVE` is the same as `ROW SHARE`, but it also prohibits locking in `SHARE` mode. `ROW EXCLUSIVE` locks are automatically obtained when updating, inserting, or deleting.

SHARE UPDATE

See ROW SHARE.

SHARE

`SHARE` permits concurrent queries but prohibits updates to the locked table.

SHARE ROW EXCLUSIVE

`SHARE ROW EXCLUSIVE` is used to look at a whole table and to allow others to look at rows in the table but to prohibit others from locking the table in `SHARE` mode or from updating rows.

EXCLUSIVE

`EXCLUSIVE` permits queries on the locked table but prohibits any other activity on it.

NOWAIT

Specify `NOWAIT` if you want the database to return control to you immediately if the specified table, partition, or table subpartition is already locked by another user. In this case, the database returns a message indicating that the table, partition, or subpartition is already locked by another user.

WAIT

Use the `WAIT` clause to indicate that the `LOCK TABLE` statement should wait up to the specified number of seconds to acquire a DML lock. There is no limit on the value of *`integer`*.

If you specify neither `NOWAIT` nor `WAIT`, then the database waits indefinitely until the table is available, locks it, and returns control to you. When the database is executing DDL statements concurrently with DML statements, a timeout or deadlock can sometimes result. The database detects such timeouts and deadlocks and returns an error.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about locking tables

Examples

Locking a Table: Example

The following statement locks the `employees` table in exclusive mode but does not wait if another user already has locked the table:

```
LOCK TABLE employees
   IN EXCLUSIVE MODE
   NOWAIT;
```

The following statement locks the remote `employees` table that is accessible through the database link `remote`:

```
LOCK TABLE employees@remote
   IN SHARE MODE;
```