11

SQL Statements: ALTER LIBRARY to ALTER SESSION

This chapter contains the following SQL statements:

ALTER LIBRARY

Purpose

The `ALTER LIBRARY` statement explicitly recompiles a library. Explicit recompilation eliminates the need for implicit run-time recompilation and prevents associated run-time compilation errors and performance overhead.

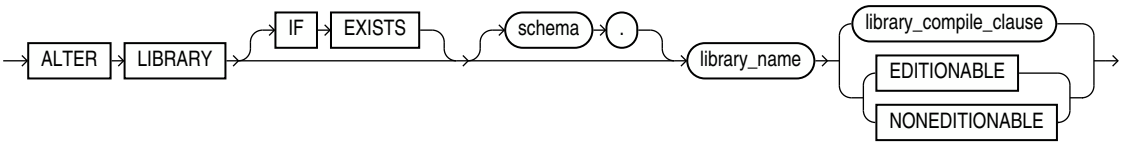
> **Note:**
>
> This statement does not change the declaration or definition of an existing library. To redeclare or redefine a library, use the "CREATE LIBRARY " with the `OR REPLACE` clause.

Prerequisites

If the library is in the `SYS` schema, you must be connected as `SYSDBA`. Otherwise, the library must be in your own schema or you must have the `ALTER ANY LIBRARY` system privilege.

Syntax

***alter_library*::=**



(*library_compile_clause*: See *Oracle Database PL/SQL Language Reference* for the syntax of this clause.)

Semantics

IF EXISTS

Specify `IF EXISTS` to alter an existing table.

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

schema

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

library_name

Specify the name of the library to be recompiled.

library_compile_clause

See *Oracle Database PL/SQL Language Reference* for the syntax and semantics of this clause and for complete information on creating and compiling libraries.

EDITIONABLE | NONEDITIONABLE

Use these clauses to specify whether the library becomes an editioned or noneditioned object if editioning is later enabled for the schema object type `LIBRARY` in *schema*. The default is `EDITIONABLE`. For information about altering editioned and noneditioned objects, see *Oracle Database Development Guide*.

ALTER LOCKDOWN PROFILE

Purpose

Use the `ALTER LOCKDOWN PROFILE` statement to alter a PDB lockdown profile. You can use PDB lockdown profiles in a multitenant environment to restrict user operations in pluggable databases (PDBs).

Immediately after you create a lockdown profile with the `CREATE LOCKDOWN PROFILE` statement, all user operations are enabled for the profile. You can then use the `ALTER LOCKDOWN PROFILE` statement to disable certain user operations for the profile. When a lockdown profile is applied to a CDB, application container, or PDB, users cannot perform the operations that are the disabled for the profile. If you later would like to reenable some of the disabled user operations, you can use the `ALTER LOCKDOWN PROFILE` statement to do so.

The `ALTER LOCKDOWN PROFILE` statement allows you to disable or enable:

- User operations associated with certain database features (using the *`lockdown_features`* clause)
- User operations associated with certain database options (using the *`lockdown_options`* clause)
- The issuance of certain SQL statements (using the *`lockdown_statements`* clause)

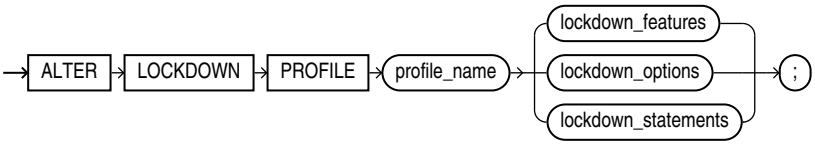
> **See Also:**
>
> - CREATE LOCKDOWN PROFILE and DROP LOCKDOWN PROFILE
> - *Oracle Database Security Guide* for more information on PDB lockdown profiles

Prerequisites

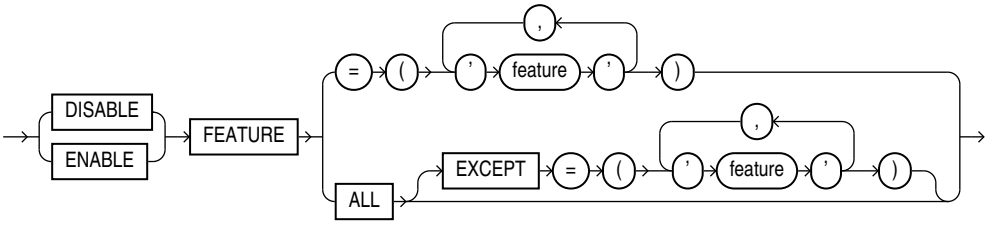
- You must issue the `ALTER LOCKDOWN PROFILE` statement from the CDB Root or Application Root.
- You must have the `ALTER LOCKDOWN PROFILE` system privilege in the container in which you issue the statement.

Syntax

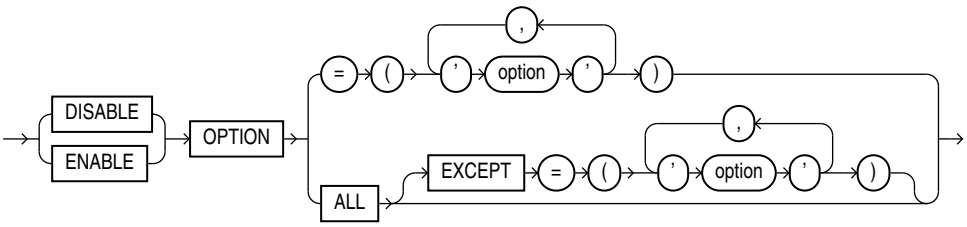
***alter_lockdown_profile*::=**



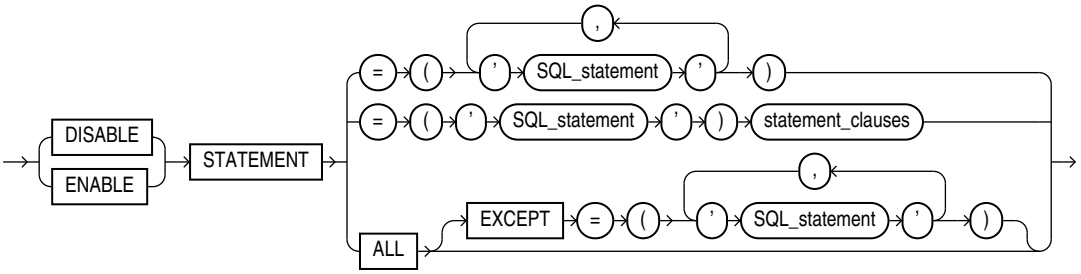
***lockdown_features*::=**



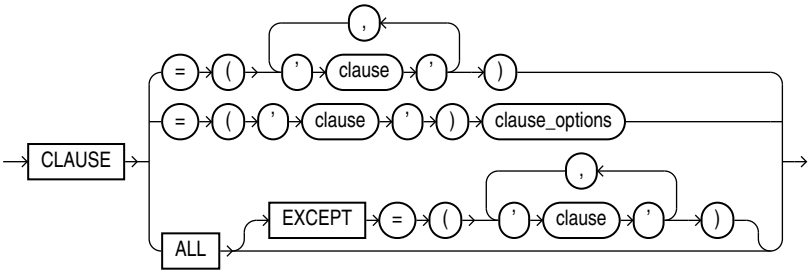
***lockdown_options*::=**



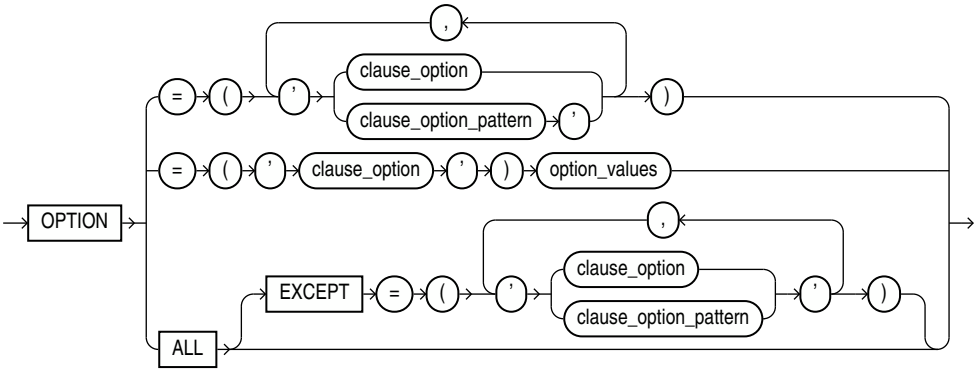
***lockdown_statements*::=**



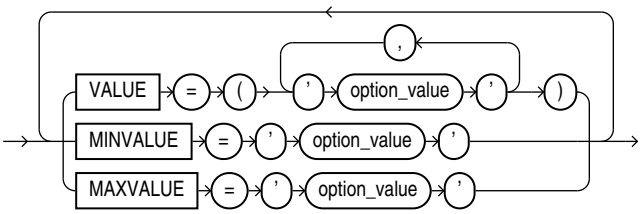
***statement_clauses*::=**



***clause_options*::=**



***option_values*::=**



Semantics

profile_name

Specify the name of the PDB lockdown profile to be altered.

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

lockdown_features

This clause lets you disable or enable user operations associated with certain database features.

- Specify `DISABLE` to add a restriction for the specified features. Users will be restricted from performing these operations in any PDB to which the profile applies.
- Specify `ENABLE` to remove a restriction for the specified features. Users will be allowed to perform these operations in any PDB to which the profile applies.
- Use *`feature`* to specify the features whose operations you want to disable or enable. Table 11-1 lists the features you can specify and describes the operations associated with each feature. The table also indicates a feature bundle for each feature. For *`feature`*, you can specify a feature bundle name to disable or enable user operations for all features in that bundle, or you can specify an individual feature name. You can specify feature bundle names and feature names in any combination of uppercase and lowercase letters.
- Use `ALL` to specify all features listed in the table.
- Use `ALL EXCEPT` to specify all features listed in the table except the specified features.

If you omit this clause, then the default is `ENABLE ALL`.

> **Note:**
>
> - The Oracle Text type `FILE_DATASTORE` is deprecated. Oracle recommends that you replace `FILE_DATASTORE` indexes with the `DIRECTORY_DATASTORE` index type for greater security as it enables file access to be based on directory objects.
> - The Oracle Text type `URL_DATASTORE` is deprecated. Oracle recommeds that you replace `URL_DATASTORE` with `NETWORK_DATASTORE`, which uses ACLs to control access to specific servers.

Table 11-1 PDB Lockdown Profile Features

Feature Bundle	Feature	Operations
AWR_ACCESS	AWR_ACCESS	The PDB taking manual and automatic Automatic Workload Repository (AWR) snapshots
COMMON_SCHEMA_ACCESS	COMMON_USER_LOCAL_SCHEMA_ACCESS	A common user invoking an invoker's rights code unit or accessing a BEQUEATH CURRENT_USER view owned by any local user in the PDB
COMMON_SCHEMA_ACCESS	LOCAL_USER_COMMON_SCHEMA_ACCESS	• A local user with an ANY system privilege (for example, CREATE ANY TABLE) creating or accessing objects in a common user's schema for which the privilege applies. **Note:** Disabling the LOCAL_USER_COMMON_SCHEMA_ACCESS feature does not prevent a local user with the SYSDBA privilege or specific object privileges from creating or accessing objects in a common user's schema. Therefore, Oracle recommends against granting such privileges to local users. • A local user with the BECOME USER system privilege becoming a common user • A local user altering a common user by issuing an ALTER USER statement • A local user using a common user for proxy connections
COMMON_SCHEMA_ACCESS	SECURITY_POLICIES	Creation of certain security policies by a local user on a common object, including: • Data Redaction • Fine Grained Auditing (FGA) • Real Application Security (RAS) • Virtual Private Database (VPD)
CONNECTIONS	COMMON_USER_CONNECT	A common user connecting to the PDB directly. If this feature is disabled, then in order to connect to the PDB, a common user must first connect to the CDB root and then switch to the desired PDB using the ALTER SESSION SET CONTAINER statement.
CONNECTIONS	LOCAL_SYSOPER_RESTRICTED_MODE_CONNECT	A local user with the SYSOPER privilege connecting to a PDB that is open in RESTRICTED mode
CTX_LOGGING	CTX_LOGGING	Use logging in Oracle Text PL/SQL procedures such as CTX_OUTPUT.START_LOG and CTX_OUTPUT.START_QUERY_LOG
JAVA	JAVA	Java as a whole. If this feature is disabled, then all options and features of the database that depend on Java will be disabled.
JAVA_RUNTIME	JAVA_RUNTIME	Operations through Java that require java.lang.RuntimePermission

Table 11-1 (Cont.) PDB Lockdown Profile Features

Feature Bundle	Feature	Operations
NETWORK_ACCESS	AQ_PROTOCOLS	Using HTTP, SMTP, and OCI notification features.
NETWORK_ACCESS	CTX_PROTOCOLS	• Operations that access the Oracle Text datastore types DIRECTORY_DATASTORE and NETWORK_DATASTORE. The type DIRECTORY_DATASTORE has an attribute called DIRECTORY which is the directory object whose data is to be indexed. The default value of this attribute is null. The DIRECTORY_DATASTORE type replaces the FILE_DATASTORE type, which is deprecated. The NETWORK_DATASTORE type replaces the URL_DATASTORE type, which is deprecated. The type NETWORK_DATASTORE conforms to the standard database security model for providing URL access based on access control lists (ACLs), which support the HTTP and HTTPS protocols. The URL_DATASTORE type did not support HTTPS. • Printing tokens as part of CTX logging with events EVENT_INDEX_PRINT_TOKEN and EVENT_OPT_PRINT_TOKEN
NETWORK_ACCESS	DBMS_DEBUG_JDWP	Using the DBMS_DEBUG_JDWP PL/SQL package
NETWORK_ACCESS	UTL_HTTP	Using the UTL_HTTP PL/SQL package
NETWORK_ACCESS	UTL_INADDR	Using the UTL_INADDR PL/SQL package
NETWORK_ACCESS	UTL_SMTP	Using the UTL_SMTP PL/SQL package
NETWORK_ACCESS	UTL_TCP	Using the UTL_TCP PL/SQL package
NETWORK_ACCESS	XDB_PROTOCOLS	Using HTTP, FTP, and other network protocols through XDB
OS_ACCESS	DROP_TABLESPACE_KEEP_DATAFILES	Dropping a tablespace in the PDB without specifying the INCLUDING CONTENTS AND DATAFILES clause in DROP TABLESPACE statement
OS_ACCESS	EXTERNAL_FILE_ACCESS	Using external files or directory objects in the PDB when PATH_PREFIX is not set for the PDB
OS_ACCESS	EXTERNAL_PROCEDURES	Using external procedure agent extproc in the PDB
OS_ACCESS	FILE_TRANSFER	Using the DBMS_FILE_TRANSFER package
OS_ACCESS	JAVA_OS_ACCESS	Using java.io.FilePermission from Java
OS_ACCESS	LOB_FILE_ACCESS	Using BFILE and CFILE data types

Table 11-1 (Cont.) PDB Lockdown Profile Features

Feature Bundle	Feature	Operations
OS_ACCESS	TRACE_VIEW_ACCESS	Using the following trace views: • [G]V$DIAG_OPT_TRACE_RECORDS • [G]V$DIAG_SQL_TRACE_RECORDS • [G]V$DIAG_TRACE_FILE_CONTENTS • V$DIAG_SESS_OPT_TRACE_RECORDS • V$DIAG_SESS_SQL_TRACE_RECORDS
OS_ACCESS	UTL_FILE	Using UTL_FILE. If this feature is disabled, then the database blocks use of the UTL_FILE.FOPEN function.

lockdown_options

This clause lets you disable or enable user operations associate with certain database options.

- Specify DISABLE to disable user operations for the specified options. Users will be restricted from performing these operations in any PDB to which the profile applies.
- Specify ENABLE to enable user operations for the specified options. Users will be allowed to perform these operations in any PDB to which the profile applies.
- For *option*, you can specify the following database options in any combination of uppercase and lowercase letters:
  - DATABASE QUEUING – Represents user operations associated with the Oracle Database Advanced Queuing option
  - PARTITIONING – Represents user operations associated with the Oracle Partitioning option
- Use ALL to specify all options in the preceding list.
- Use ALL EXCEPT to specify all options in the preceding list except the specified options.

If you omit this clause, then the default is ENABLE OPTION ALL.

lockdown_statements

This clause lets you disable or enable the issuance of certain SQL statements.

- Specify DISABLE to disable the issuance of the specified SQL statements. Users will be restricted from issuing these statements in any PDB to which the profile applies.
- Specify ENABLE to enable the issuance of the specified SQL statements. Users will be allowed to issue these statements in any PDB to which the profile applies.
- For *SQL_statement*, you can specify the following statements in any combination of uppercase and lowercase letters:
  - ADMINISTER KEY MANAGEMENT
  - ALTER DATABASE
  - ALTER PLUGGABLE DATABASE
  - ALTER SESSION
  - ALTER SYSTEM

  - ALTER TABLE
  - ALTER INDEX
  - ALTER TABLESPACE
  - ALTER PROFILE
  - CREATE TABLE
  - CREATE INDEX
  - CREATE TABLESPACE
  - CREATE PROFILE
  - CREATE DATABASE LINK
  - DROP TABLE
  - DROP INDEX
  - DROP TABLESPACE
  - DROP PROFILE
- Use ALL to specify all statements in the preceding list.
- Use ALL EXCEPT to specify all statements in the preceding list except the specified statements.

If you omit this clause, then the default is ENABLE STATEMENT ALL.

statement_clauses

This clause lets you disable or enable specific clauses of the specified SQL statement.

- Use *clause* to specify the SQL keywords that form the clause you want to disable or enable. You can specify a clause in any combination of uppercase and lowercase letters.
- Use ALL to specify all clauses for the SQL statement.
- Use ALL EXCEPT to specify all clauses for the SQL statement except the specified clauses.

For *clause*, you must specify at least enough keywords to unambiguously identify a single clause for the SQL statement. The following are some examples of how to specify *clause* for the ALTER SYSTEM statement:

- To specify the *archive_log_clause*::=, specify ARCHIVE. This is sufficient because no other ALTER SYSTEM clause begins with the keyword ARCHIVE. Alternatively, you can specify ARCHIVE LOG for semantic clarity, but the LOG keyword is unnecessary.
- To specify either of the *rolling_migration_clauses*::=, you must specify START ROLLING MIGRATION or STOP ROLLING MIGRATION in order to distinguish these clauses from the similarly named *rolling_patch_clauses*::= START ROLLING PATCH and STOP ROLLING PATCH.
- You cannot specify the single keyword FLUSH, because several ALTER SYSTEM clauses begin with this keyword. You must instead specify each clause separately, such as FLUSH SHARED_POOL or FLUSH GLOBAL CONTEXT.

There is no need to specify optional keywords within a clause, because they have no effect. For example:

- The *archive_log_clause*::= has an optional INSTANCE keyword. However, you cannot enable or disable only ARCHIVE LOG clauses that contain the INSTANCE keyword. Specifying ARCHIVE LOG INSTANCE is equivalent to specifying ARCHIVE or ARCHIVE LOG.

There is no need to specify parameter values within a clause, because they have no effect. For example:

- The *shutdown_dispatcher_clause*::= requires you to specify a *dispatcher_name*. However, you cannot enable or disable SHUTDOWN clauses that contain a specific dispatcher name. Specifying SHUTDOWN dispatcher1 is equivalent to specifying SHUTDOWN.

> **See Also:**
>
> ALTER DATABASE , ALTER PLUGGABLE DATABASE, ALTER SESSION , and ALTER SYSTEM for complete information on the clauses for these statements

clause_options

This clause is valid only when you specify one of the following for *lockdown_statements* and *statement_clauses*:

```
{ DISABLE | ENABLE } STATEMENT = ('ALTER SESSION') CLAUSE = ('SET')
{ DISABLE | ENABLE } STATEMENT = ('ALTER SYSTEM') CLAUSE = ('SET')
```

This clause lets you disable or enable the setting or modification of specific options with the ALTER SESSION SET or ALTER SYSTEM SET statements.

- Use *clause_option* to specify the option you want to disable or enable.
- Use *clause_option_pattern* to specify a pattern that matches multiple options. Within the pattern, specify a percent sign (%) to match zero or more characters in an option name. For example, specifying 'QUERY_REWRITE_%' is equivalent to specifying both the QUERY_REWRITE_ENABLED and QUERY_REWRITE_INTEGRITY options.
- You can specify *clause_option* and *clause_option_pattern* in any combination of uppercase and lowercase letters.
- Use ALL to specify all options.
- Use ALL EXCEPT to specify all options except the specified options.

> **See Also:**
>
> The *alter_session_set_clause* clause of ALTER SESSION and the *alter_system_set_clause* clause of ALTER SYSTEM for complete information on the options you can specify for these statements

option_values

This clause is valid only when you specify one of the following for *lockdown_statements*, *statement_clauses*, and *clause_options*:

```
DISABLE STATEMENT = ('ALTER SESSION') CLAUSE = ('SET') OPTION = clause_option
DISABLE STATEMENT = ('ALTER SYSTEM') CLAUSE = ('SET') OPTION = clause_option
```

This clause lets you specify a default value for an option when disabling the setting of that option. For options that take numeric values, this clause also lets you restrict users from setting an option to certain values.

- The `VALUE` clause lets you specify a default *`option_value`* for *`clause_option`*, which will go into effect for any PDB to which the profile applies after you close and reopen the PDB. If *`clause_option`* accepts multiple default values, then you can specify more than one *`option_value`* in a comma-separated list. The purpose of using this clause is to simultaneously set a default value for an option and restrict users from setting or modifying the value.
- The `MINVALUE` clause lets you restricts users from setting the value of *`clause_option`* to a value less than *`option_value`*. You can specify this clause only for options that take a numeric value.
- The `MAXVALUE` clause lets you restricts users from setting the value of *`clause_option`* to a value greater than *`option_value`*. You can specify this clause only for options that take a numeric value.
- You can specify both the `MINVALUE` and `MAXVALUE` clauses together to restrict users from setting the value of *`clause_options`* to any value less than `MINVALUE` or greater than `MAXVALUE`.
- `MINVALUE` and `MAXVALUE` settings take effect immediately when the lockdown profile is assigned to a PDB; you need not close and reopen the PDB.

> **See Also:**
>
> *Oracle Database Reference* for complete information on the values allowed for the various options

USERS Clause

As a CDB administrator or an Application administrator you can use the `USERS` clause to configure lockdown rules for a specific set of users.

The values for `USERS` in a `CDB$ROOT` lockdown profile are as follows:

- `USERS = ALL` means that the lockdown rule applies to all users in the PDB.
- `USERS = COMMON` means that the lockdown rule applies only to CDB `COMMON` users in the PDB.
- `USERS = LOCAL` means that the lockdown rule applies only to local users in the PDB. Application common users are considered local users at the CDB level.

The values for `USERS` in an Application `ROOT` lockdown profile are as follows:

- `USERS = ALL` means that the lockdown rule applies to all users in the PDB.
- `USERS = COMMON` means that the lockdown rule applies only to Application `COMMON` users in the PDB.
- `USERS = LOCAL` means that the lockdown rule applies only to local users in the PDB.

Note that the Application lockdown profile rules should not affect CDB common users.

- `ALL` users means Application common users and local users in the PDB.
- `COMMON` users means Application common users in the PDB.

Examples

The following statement creates PDB lockdown profile `hr_prof`:

```
CREATE LOCKDOWN PROFILE hr_prof;
```

The remaining examples in this section alter `hr_prof`.

Disabling Features for PDB Lockdown Profiles: Examples

The following statement disables all features in the feature bundle `NETWORK_ACCESS`:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE FEATURE = ('NETWORK_ACCESS');
```

The following statement disables the `LOB_FILE_ACCESS` and `TRACE_VIEW ACCESS` features:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE FEATURE = ('LOB_FILE_ACCESS', 'TRACE_VIEW_ACCESS');
```

The following statement disables all features except the `COMMON_USER_LOCAL_SCHEMA_ACCESS` and `LOCAL_USER_COMMON_SCHEMA_ACCESS` features:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE FEATURE ALL EXCEPT = ('COMMON_USER_LOCAL_SCHEMA_ACCESS',
'LOCAL_USER_COMMON_SCHEMA_ACCESS');
```

The following statement disables all features:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE FEATURE ALL;
```

Enabling Features for PDB Lockdown Profiles: Examples

The following statement enables the `UTL_HTTP` and `UTL_SMTP` features, as well as all features in the feature bundle `OS_ACCESS`:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE FEATURE = ('UTL_HTTP', 'UTL_SMTP', 'OS_ACCESS');
```

The following statement enables all features except the `AQ_PROTOCOLS` and `CTX_PROTOCOLS` features:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE FEATURE ALL EXCEPT = ('AQ_PROTOCOLS', 'CTX_PROTOCOLS');
```

The following statement enables all features:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE FEATURE ALL;
```

Disabling Options for PDB Lockdown Profiles: Examples

The following statement disables user operations associated with the Oracle Database Advanced Queuing option:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE OPTION = ('DATABASE QUEUING');
```

The following statement disables user operations associated with the Oracle Partitioning option:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE OPTION = ('PARTITIONING');
```

Enabling Options for PDB Lockdown Profiles: Examples

The following statement enables user operations associated with the Oracle Database Advanced Queuing option:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE OPTION = ('DATABASE QUEUING');
```

The following statement enables user operations associated both with the Oracle Database Advanced Queuing option and the Oracle Partitioning option:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE OPTION ALL;
```

Disabling SQL Statements for PBB Lockdown Profiles: Examples

The following statement disables the `ALTER DATABASE` statement:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER DATABASE');
```

The following statement disables the `ALTER SYSTEM SUSPEND` and `ALTER SYSTEM RESUME` statements:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER SYSTEM')
          CLAUSE = ('SUSPEND', 'RESUME');
```

The following statement disables all clauses of the `ALTER PLUGGABLE DATABASE` statement, except `DEFAULT TABLESPACE` and `DEFAULT TEMPORARY TABLESPACE`:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER PLUGGABLE DATABASE')
          CLAUSE ALL EXCEPT = ('DEFAULT TABLESPACE', 'DEFAULT TEMPORARY
TABLESPACE');
```

The following statement disables using the `ALTER SESSION` statement to set or modify `COMMIT_WAIT` or `CURSOR_SHARING`:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER SESSION')
          CLAUSE = ('SET')
          OPTION = ('COMMIT_WAIT', 'CURSOR_SHARING');
```

The following statement disables using the `ALTER SYSTEM` statement to set or modify the value of `PDB_FILE_NAME_CONVERT`. It also sets the default value for `PDB_FILE_NAME_CONVERT` to `'cdb1_pdb0', 'cdb1_pdb1'`. This default value will take effect the next time the PDB is closed and reopened.

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER SYSTEM')
          CLAUSE = ('SET')
          OPTION = ('PDB_FILE_NAME_CONVERT')
          VALUE = ('cdb1_pdb0', 'cdb1_pdb1');
```

The following statement disables using the `ALTER SYSTEM` statement to set or modify the value of `CPU_COUNT` to a value less than `8`:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER SYSTEM')
          CLAUSE = ('SET')
          OPTION = ('CPU_COUNT')
          MINVALUE = '8';
```

The following statement disables using the `ALTER SYSTEM` statement to set or modify the value of `CPU_COUNT` to a value greater than `2`:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER SYSTEM')
          CLAUSE = ('SET')
          OPTION = ('CPU_COUNT')
          MAXVALUE = '2';
```

The following statement disables using the `ALTER SYSTEM` statement to set or modify the value of `CPU_COUNT` to a value less than `2` or greater than `6`:

```
ALTER LOCKDOWN PROFILE hr_prof
  DISABLE STATEMENT = ('ALTER SYSTEM')
          CLAUSE = ('SET')
          OPTION = ('CPU_COUNT')
          MINVALUE = '2'
          MAXVALUE = '6';
```

Enabling SQL Statements for PBB Lockdown Profiles: Examples

The following statement enables all statements except `ALTER DATABASE`:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE STATEMENT ALL EXCEPT = ('ALTER DATABASE');
```

The following statement enables the `ALTER DATABASE MOUNT` and `ALTER DATABASE OPEN` statements:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE STATEMENT = ('ALTER DATABASE')
           CLAUSE = ('MOUNT', 'OPEN');
```

The following statement enables all clauses of the `ALTER PLUGGABLE DATABASE` statement, except `DEFAULT TABLESPACE` and `DEFAULT TEMPORARY TABLESPACE`:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE STATEMENT = ('ALTER PLUGGABLE DATABASE')
         CLAUSE ALL EXCEPT = ('DEFAULT TABLESPACE', 'DEFAULT TEMPORARY
TABLESPACE');
```

The following statement enables using the `ALTER SESSION` statement to set or modify `COMMIT_WAIT` or `CURSOR_SHARING`:

```
ALTER LOCKDOWN PROFILE hr_prof
  ENABLE STATEMENT = ('ALTER SESSION')
         CLAUSE = ('SET')
         OPTION = ('COMMIT_WAIT', 'CURSOR_SHARING');
```

ALTER MATERIALIZED VIEW

Purpose

A materialized view is a database object that contains the results of a query. The `FROM` clause of the query can name tables, views, and other materialized views. Collectively these source objects are called **master tables** (a replication term) or **detail tables** (a data warehousing term). This reference uses the term master tables for consistency. The databases containing the master tables are called the **master databases**.

Use the `ALTER MATERIALIZED VIEW` statement to modify an existing materialized view in one or more of the following ways:

- To change its storage characteristics
- To change its refresh method, mode, or time
- To alter its structure so that it is a different type of materialized view
- To enable or disable query rewrite

> **Note:**
>
> The keyword `SNAPSHOT` is supported in place of `MATERIALIZED VIEW` for backward compatibility.

> **See Also:**
>
> - CREATE MATERIALIZED VIEW for more information on creating materialized views
> - *Oracle Database Administrator's Guide* for information on materialized views in a replication environment
> - *Oracle Database Data Warehousing Guide* for information on materialized views in a data warehousing environment

Prerequisites

The materialized view must be in your own schema, or you must have the `ALTER ANY MATERIALIZED VIEW` system privilege.

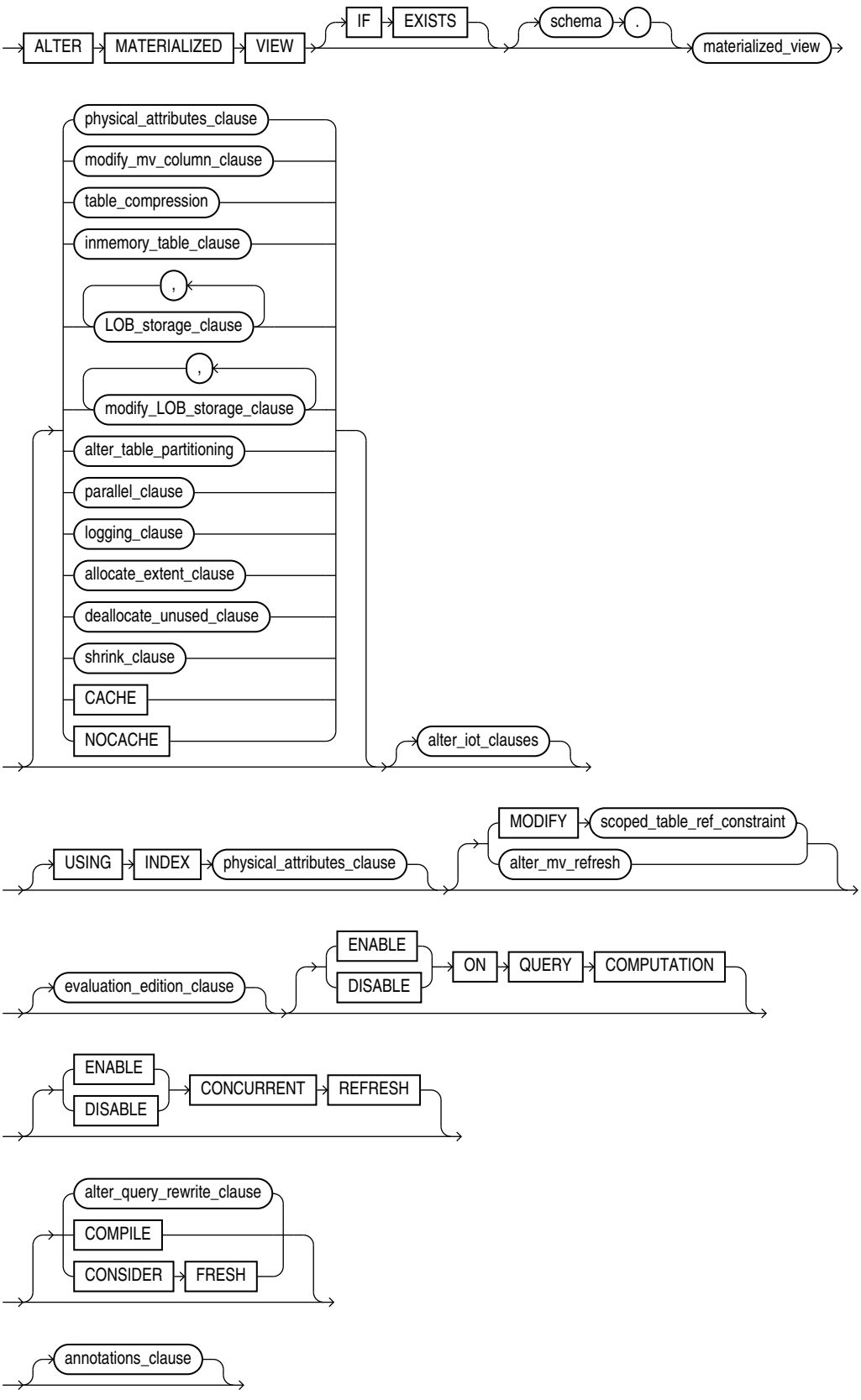
To enable a materialized view for query rewrite:

- If all of the master tables in the materialized view are in your schema, then you must have the `QUERY REWRITE` privilege.
- If any of the master tables are in another schema, then you must have the `GLOBAL QUERY REWRITE` privilege.
- If the materialized view is in another user's schema, then both you and the owner of that schema must have the appropriate `QUERY REWRITE` privilege, as described in the preceding two items. In addition, the owner of the materialized view must have `SELECT` access to any master tables that the materialized view owner does not own.

To specify an edition in the *`evaluation_edition_clause`* or the *`unusable_editions_clause`*, you must have the `USE` privilege on the edition.

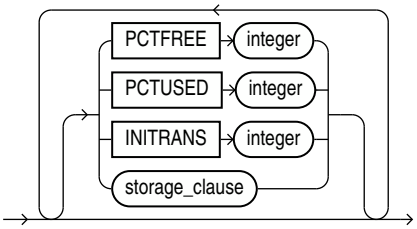
Syntax

***alter_materialized_view*::=**



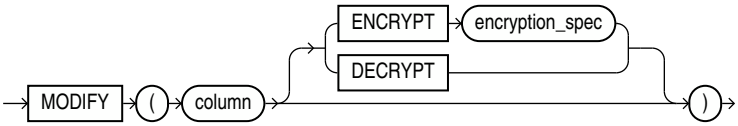
(*physical_attributes_clause*::=, *modify_mv_column_clause*::=, *table_compression*::=, *inmemory_table_clause*::=, *LOB_storage_clause*::= , *modify_LOB_storage_clause*::=, *alter_table_partitioning*::= (part of `ALTER TABLE`), *parallel_clause*::=, *logging_clause*::=, *allocate_extent_clause*::=, *deallocate_unused_clause*::=, *shrink_clause*::=, *alter_iot_clauses*::=, *scoped_table_ref_constraint*::=, *alter_mv_refresh*::=, *evaluation_edition_clause*::=, *alter_query_rewrite_clause*::=, *annotations_clause*)

***physical_attributes_clause*::=**

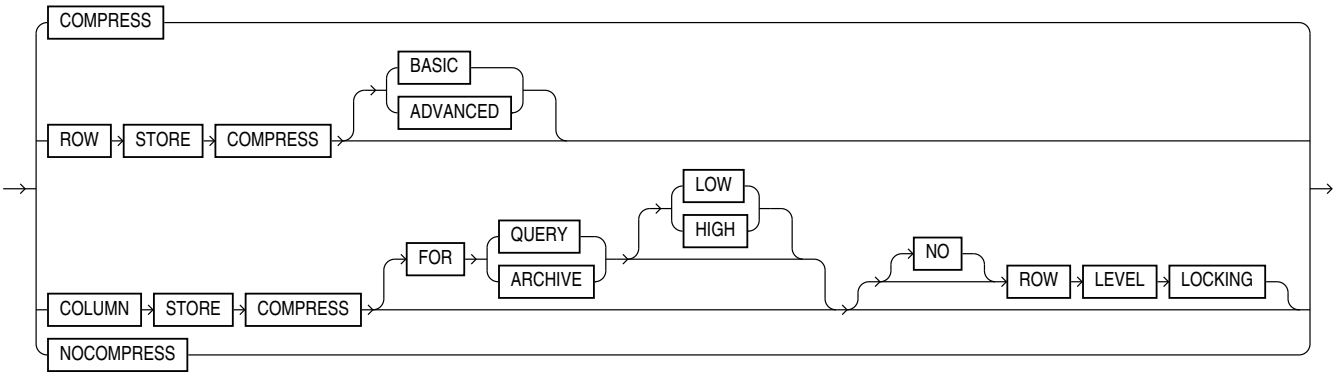


(*storage_clause*::=)

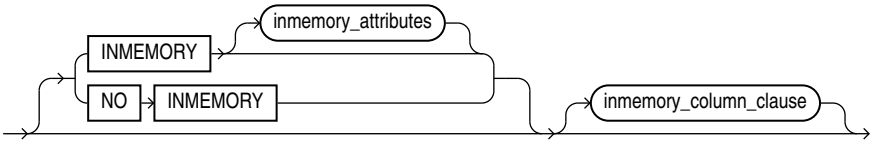
***modify_mv_column_clause*::=**



***table_compression*::=**

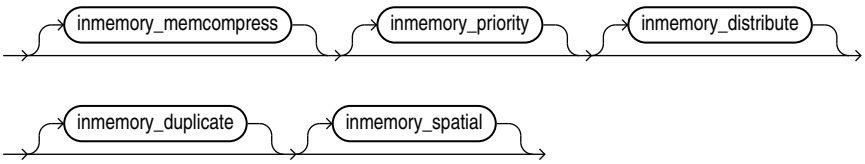


***inmemory_table_clause*::=**



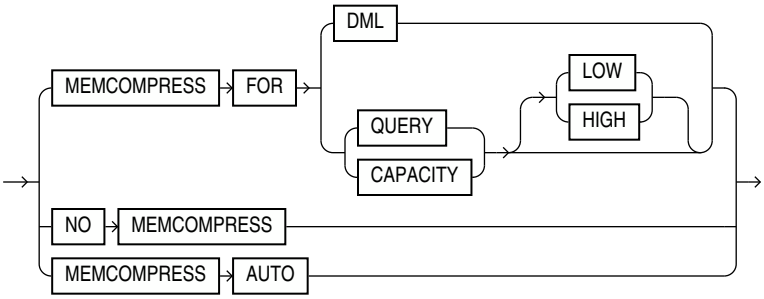
(*inmemory_attributes*::=, *inmemory_column_clause*::=)

***inmemory_attributes*::=**

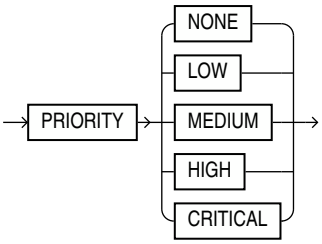


(*inmemory_memcompress*::=, *inmemory_priority*::=, *inmemory_distribute*::=, *inmemory_duplicate*::=)

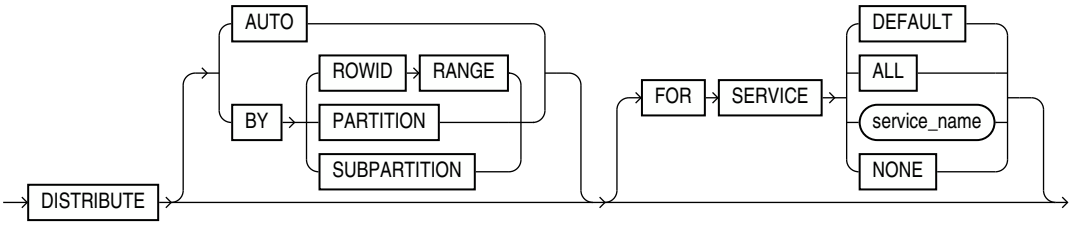
***inmemory_memcompress*::=**



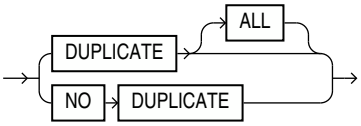
***inmemory_priority*::=**



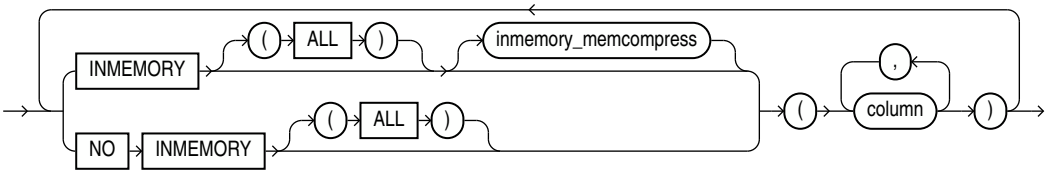
***inmemory_distribute*::=**



***inmemory_duplicate*::=**

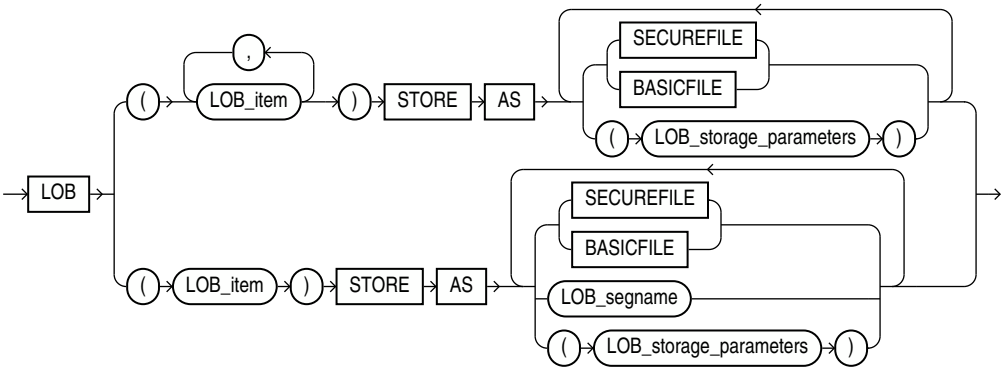


***inmemory_column_clause*::=**



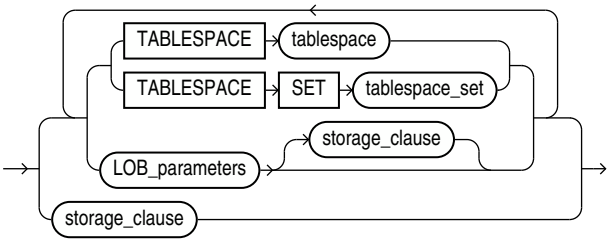
(*inmemory_memcompress*::=)

***LOB_storage_clause*::=**



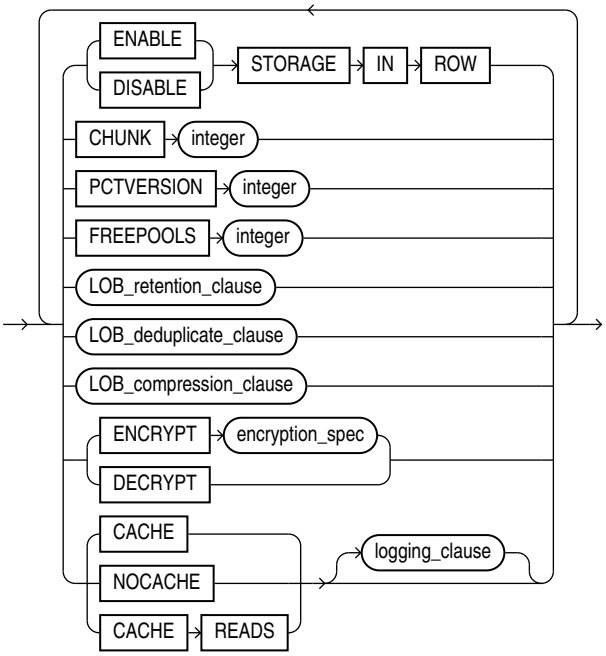
(*LOB_storage_parameters*::=)

***LOB_storage_parameters*::=**



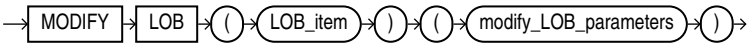
(`TABLESPACE SET`: not supported with `ALTER MATERIALIZED VIEW`, *LOB_parameters*::=, *storage_clause*::=)

***LOB_parameters*::=**



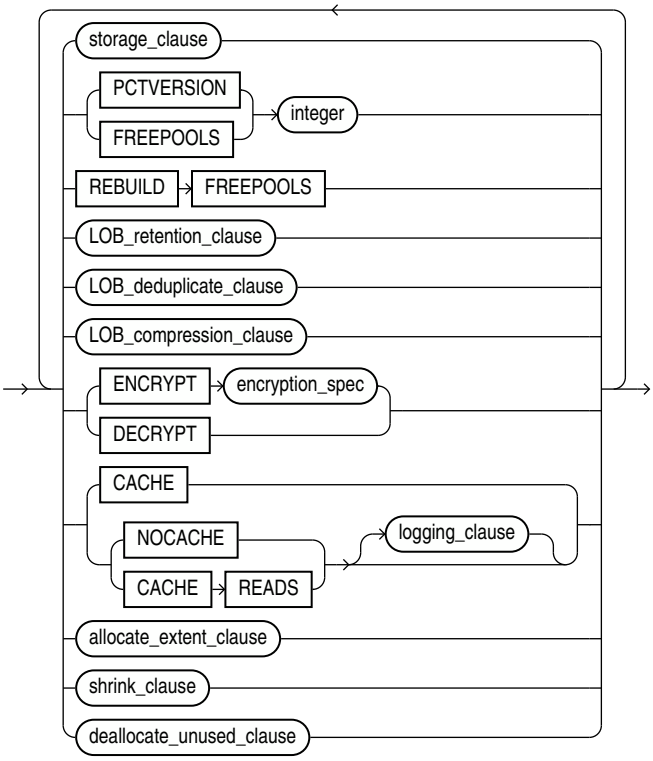
(*storage_clause*::=, *logging_clause*::=)

***modify_LOB_storage_clause*::=**



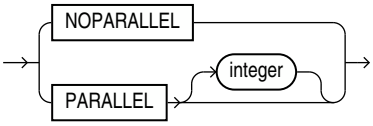
(*modify_LOB_parameters*::=)

***modify_LOB_parameters*::=**

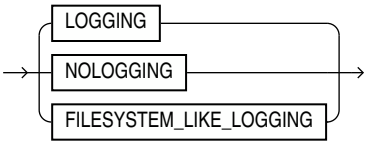


(*storage_clause*::=, *LOB_retention_clause*::=, *LOB_compression_clause*::=, *logging_clause*::=, *allocate_extent_clause*::=, *shrink_clause*::=, *deallocate_unused_clause*::=)

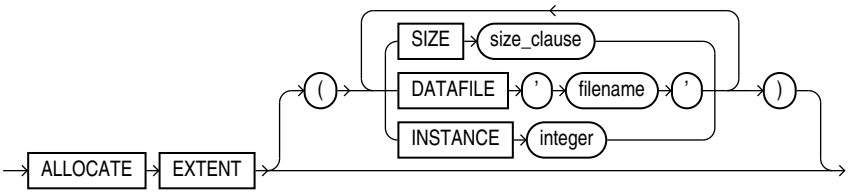
***parallel_clause*::=**



***logging_clause*::=**

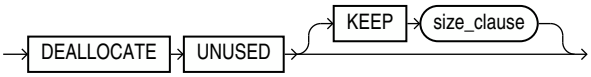


***allocate_extent_clause*::=**



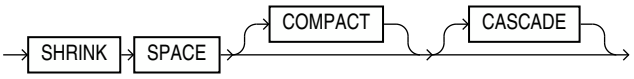
(*size_clause*::=)

***deallocate_unused_clause*::=**

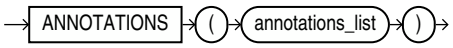


(*size_clause*::=)

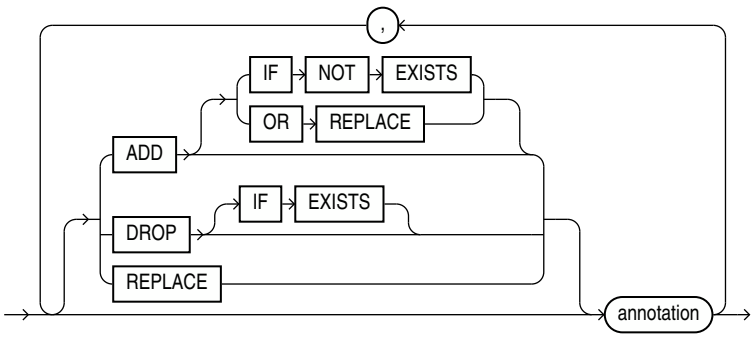
***shrink_clause*::=**



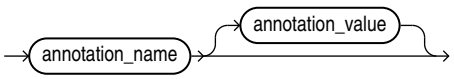
***annotations_clause*::=**



***annotations_list*::=**

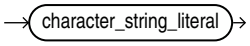


***annotation*::=**

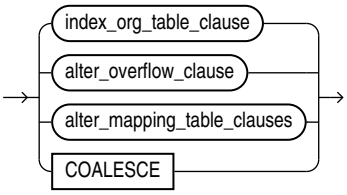


***annotation_name*::=**

***annotation_value*::=**

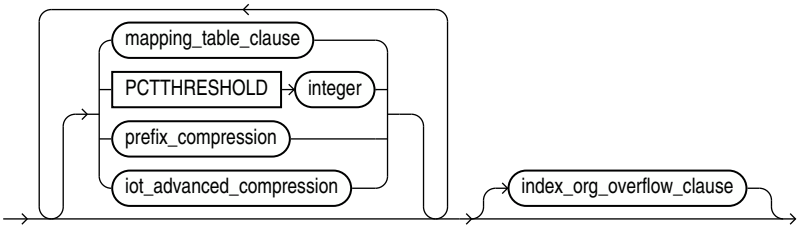


***alter_iot_clauses*::=**



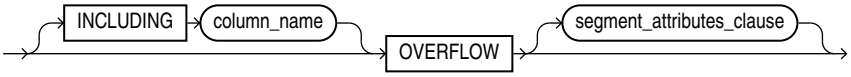
(*index_org_table_clause*::=, *alter_overflow_clause*::=, `alter_mapping_table_clauses`: not supported with materialized views)

***index_org_table_clause*::=**



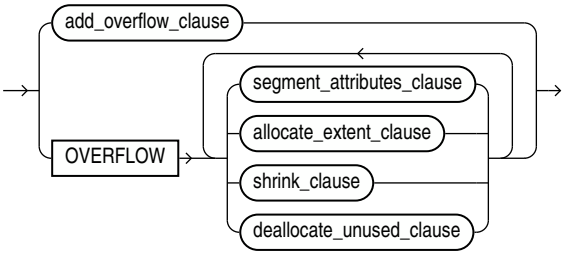
(`mapping_table_clause`: not supported with materialized views, `prefix_compression`: not supported for altering materialized views, *index_org_overflow_clause*::=)

***index_org_overflow_clause*::=**



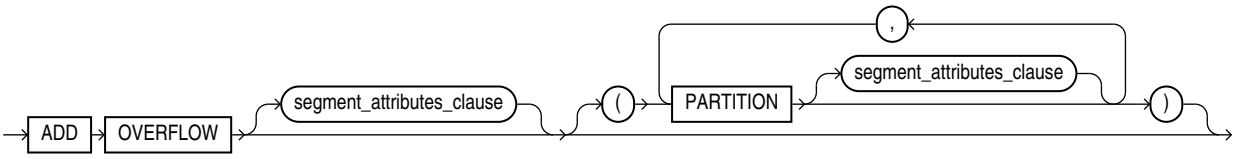
(*segment_attributes_clause*::=—part of `ALTER TABLE`)

***alter_overflow_clause*::=**



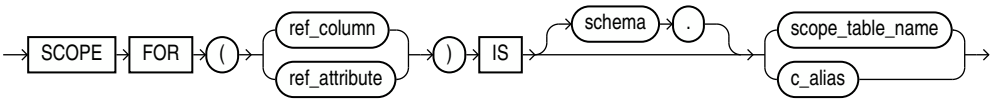
(*allocate_extent_clause*::=, *shrink_clause*::=, *deallocate_unused_clause*::=)

***add_overflow_clause*::=**

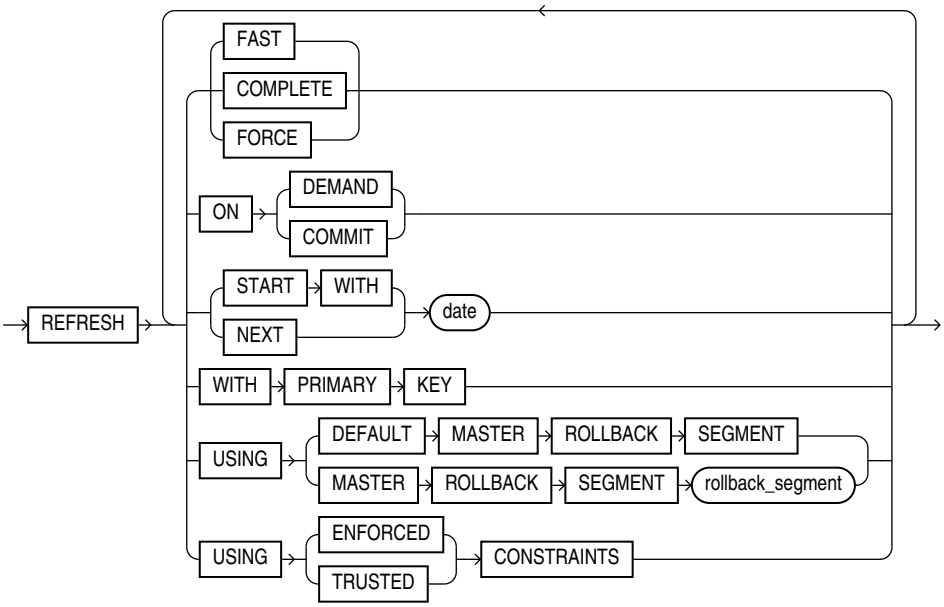


(*segment_attributes_clause*::=--part of `ALTER TABLE`)

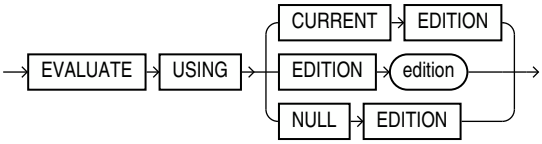
***scoped_table_ref_constraint*::=**



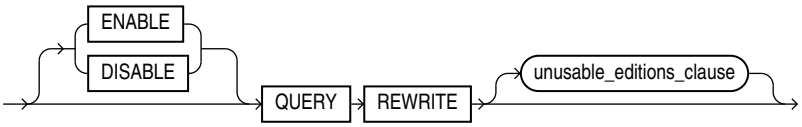
***alter_mv_refresh*::=**



***evaluation_edition_clause*::=**

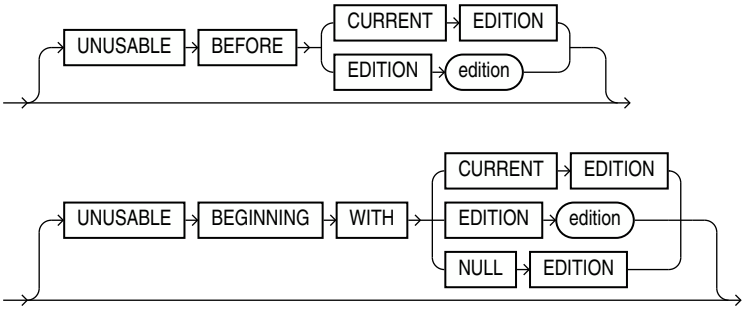


***alter_query_rewrite_clause*::=**



> **Note:**
>
> You cannot specify only `QUERY REWRITE`. You must specify at least one of the following: `ENABLE`, `DISABLE`, or a subclause of the *`unusable_editions_clause`*.

***unusable_editions_clause*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to alter an existing materialized view.

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

schema

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

materialized_view

Specify the name of the materialized view to be altered.

physical_attributes_clause

Specify new values for the `PCTFREE`, `PCTUSED`, and `INITRANS` parameters (or, when used in the `USING INDEX` clause, for the `INITRANS` parameter only) and the storage characteristics for the materialized view. Refer to ALTER TABLE for information on the `PCTFREE`, `PCTUSED`, and `INITRANS` parameters and to storage_clause for information about storage characteristics.

modify_mv_column_clause

Use this clause to encrypt or decrypt this column of the materialized view. Refer to the `CREATE TABLE` clause *encryption_spec* for information on this clause.

table_compression

Use the *`table_compression`* clause to instruct Oracle Database whether to compress data segments to reduce disk and memory use. Refer to the *table_compression* clause of `CREATE TABLE` for the full semantics of this clause.

inmemory_table_clause

Use the *`inmemory_table_clause`* to enable or disable the materialized view or its columns for the In-Memory Column Store (IM column store), or to change the In-Memory attributes for the materialized view or its columns. This clause has the same semantics here as it has for the `ALTER TABLE` statement. Refer to the *inmemory_table_clause* of `ALTER TABLE` for the full semantics of this clause.

LOB_storage_clause

The *LOB_storage_clause* lets you specify the storage characteristics of a new LOB. LOB storage behaves for materialized views exactly as it does for tables. Refer to the *LOB_storage_clause* (in CREATE TABLE) for information on the LOB storage parameters.

modify_LOB_storage_clause

The *modify_LOB_storage_clause* lets you modify the physical attributes of the LOB attribute *LOB_item* or the LOB object attribute. Modification of LOB storage behaves for materialized views exactly as it does for tables.

> **See Also:**
>
> The *modify_LOB_storage_clause* of ALTER TABLE for information on the LOB storage parameters that can be modified

alter_table_partitioning

The syntax and general functioning of the partitioning clauses for materialized views is the same as for partitioned tables. Refer to *alter_table_partitioning* in the documentation on ALTER TABLE.

Restriction on Altering Materialized View Partitions

You cannot specify the *LOB_storage_clause* or *modify_LOB_storage_clause* within any of the *partitioning_clauses*.

> **Note:**
>
> If you want to keep the contents of the materialized view synchronized with those of the master table, then Oracle recommends that you manually perform a complete refresh of all materialized views dependent on the table after dropping or truncating a table partition.

MODIFY PARTITION UNUSABLE LOCAL INDEXES

Use this clause to mark UNUSABLE all the local index partitions associated with *partition*.

MODIFY PARTITION REBUILD UNUSABLE LOCAL INDEXES

Use this clause to rebuild the unusable local index partitions associated with *partition*.

parallel_clause

The *parallel_clause* lets you change the default degree of parallelism for the materialized view.

For complete information on this clause, refer to *parallel_clause* in the documentation on CREATE TABLE.

logging_clause

Specify or change the logging characteristics of the materialized view. Refer to the *logging_clause* for a full description of this clause.

allocate_extent_clause

The *allocate_extent_clause* lets you explicitly allocate a new extent for the materialized view. Refer to the *allocate_extent_clause* for a full description of this clause.

deallocate_unused_clause

Use the *deallocate_unused_clause* to explicitly deallocate unused space at the end of the materialized view and make the freed space available for other segments. Refer to the *deallocate_unused_clause* for a full description of this clause.

shrink_clause

Use this clause to compact the materialized view segments. For complete information on this clause, refer to *shrink_clause* in the documentation on CREATE TABLE.

CACHE | NOCACHE

For data that will be accessed frequently, CACHE specifies that the blocks retrieved for this table are placed at the most recently used end of the LRU list in the buffer cache when a full table scan is performed. This attribute is useful for small lookup tables. NOCACHE specifies that the blocks are placed at the least recently used end of the LRU list. Refer to "CACHE | NOCACHE | CACHE READS" in the documentation on CREATE TABLE for more information about this clause.

annotations_clause

The *annotation_name* is an identifier that can have up to 4000 characters. If the annotation name is a reserved word it must be provided in double quotes. When a double quoted identifier is used, the identifier can also contain whitespace characters. However, identifiers that contain only whitespace characters are not accepted.

You can only change annotations at the view level with the ALTER statement. To drop column-level annotations, you must drop and recreate the view.

For the full semantics of the annotations clause see *annotations_clause* of CREATE TABLE.

alter_iot_clauses

Use the *alter_iot_clauses* to change the characteristics of an index-organized materialized view. The keywords and parameters of the components of the *alter_iot_clauses* have the same semantics as in ALTER TABLE, with the restrictions that follow.

Restrictions on Altering Index-Organized Materialized Views

You cannot specify the *mapping_table_clause* or the *prefix_compression* clause of the *index_org_table_clause*.

> **See Also:**
>
> *index_org_table_clause* of `CREATE MATERIALIZED VIEW` for information on creating an index-organized materialized view

USING INDEX Clause

Use this clause to change the value of `INITRANS` and `STORAGE` parameters for the index Oracle Database uses to maintain the materialized view data.

Restriction on the USING INDEX clause

You cannot specify the `PCTUSED` or `PCTFREE` parameters in this clause.

MODIFY *scoped_table_ref_constraint*

Use the `MODIFY` *`scoped_table_ref_constraint`* clause to rescope a `REF` column or attribute to a new table or to an alias for a new column.

Restrictions on Rescoping REF Columns

You can rescope only one `REF` column or attribute in each `ALTER MATERIALIZED VIEW` statement, and this must be the only clause in this statement.

alter_mv_refresh

Use the *`alter_mv_refresh`* clause to change the default method and mode and the default times for automatic refreshes. If the contents of the master tables of a materialized view are modified, then the data in the materialized view must be updated to make the materialized view accurately reflect the data currently in its master table(s). This clause lets you schedule the times and specify the method and mode for Oracle Database to refresh the materialized view.

> **See Also:**
>
> - This clause only sets the default refresh options. For instructions on actually implementing the refresh, refer to *Oracle Database Administrator's Guide* and *Oracle Database Data Warehousing Guide*.
> - *Oracle Database Data Warehousing Guide* to learn how to use refresh statistics to monitor the performance of materialized view refresh operations

FAST Clause

Specify `FAST` for the fast refresh method, which performs the refresh according to the changes that have occurred to the master tables. The changes are stored either in the materialized view log associated with the master table (for conventional DML changes) or in the direct loader log (for direct-path `INSERT` operations).

For both conventional DML changes and for direct-path `INSERT` operations, other conditions may restrict the eligibility of a materialized view for fast refresh.

When you change the refresh method to `FAST` in an `ALTER MATERIALIZED VIEW` statement, Oracle Database does not perform this verification. If the materialized view is not eligible for fast refresh, then Oracle Database returns an error when you attempt to refresh this view.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for restrictions on fast refresh in replication environments
> - *Oracle Database Data Warehousing Guide* for restrictions on fast refresh in data warehouse environments
> - "Automatic Refresh: Examples"

COMPLETE Clause

Specify `COMPLETE` for the complete refresh method, which is implemented by executing the defining query of the materialized view. If you specify a complete refresh, then Oracle Database performs a complete refresh even if a fast refresh is possible.

> **See Also:**
>
> "Complete Refresh: Example"

FORCE Clause

Specify `FORCE` if, when a refresh occurs, you want Oracle Database to perform a fast refresh if one is possible or a complete refresh otherwise.

ON COMMIT Clause

Specify `ON COMMIT` if you want a refresh to occur whenever Oracle Database commits a transaction that operates on a master table of the materialized view.

You cannot specify both `ON COMMIT` and `ON DEMAND`. If you specify `ON COMMIT`, then you cannot also specify `START WITH` or `NEXT`.

Restriction on ON COMMIT

This clause is supported only for materialized join views and single-table materialized aggregate views.

ON DEMAND Clause

Specify `ON DEMAND` if you want the materialized view to be refreshed on demand by calling one of the three `DBMS_MVIEW` refresh procedures. If you omit both `ON COMMIT` and `ON DEMAND`, then `ON DEMAND` is the default.

You cannot specify both `ON COMMIT` and `ON DEMAND`. `START WITH` and `NEXT` take precedence over `ON DEMAND`. Therefore, in most circumstances it is not meaningful to specify `ON DEMAND` when you have specified `START WITH` or `NEXT`.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information on these procedures
> - *Oracle Database Data Warehousing Guide* on the types of materialized views you can create by specifying `REFRESH ON DEMAND`

START WITH Clause

Specify `START WITH` *`date`* to indicate a date for the first automatic refresh time.

NEXT Clause

Specify `NEXT` to indicate a date expression for calculating the interval between automatic refreshes.

Both the `START WITH` and `NEXT` values must evaluate to a time in the future. If you omit the `START WITH` value, then Oracle Database determines the first automatic refresh time by evaluating the `NEXT` expression with respect to the creation time of the materialized view. If you specify a `START WITH` value but omit the `NEXT` value, then Oracle Database refreshes the materialized view only once. If you omit both the `START WITH` and `NEXT` values, or if you omit the *`alter_mv_refresh`* entirely, then Oracle Database does not automatically refresh the materialized view.

WITH PRIMARY KEY Clause

Specify `WITH PRIMARY KEY` to change a rowid materialized view to a primary key materialized view. Primary key materialized views allow materialized view master tables to be reorganized without affecting the ability of the materialized view to continue to fast refresh.

For you to specify this clause, the master table must contain an enabled primary key constraint and must have defined on it a materialized view log that logs primary key information.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for detailed information about primary key materialized views
> - "Primary Key Materialized View: Example"

USING ROLLBACK SEGMENT Clause

This clause is not valid if your database is in automatic undo mode, because in that mode Oracle Database uses undo tablespaces instead of rollback segments. Oracle strongly recommends that you use automatic undo mode. This clause is supported for backward compatibility with replication environments containing older versions of Oracle Database that still use rollback segments.

For complete information on this clause, refer to `CREATE MATERIALIZED VIEW` ... "USING ROLLBACK SEGMENT Clause".

USING ... CONSTRAINTS Clause

This clause has the same semantics in CREATE MATERIALIZED VIEW and ALTER MATERIALIZED VIEW statements. For complete information, refer to "USING ... CONSTRAINTS Clause" in the documentation on CREATE MATERIALIZED VIEW.

evaluation_edition_clause

Use this clause to change the evaluation edition for the materialized view. This clause has the same semantics in CREATE MATERIALIZED VIEW and ALTER MATERIALIZED VIEW statements. For complete information on this clause, refer to *evaluation_edition_clause* in the documentation on CREATE MATERIALIZED VIEW.

Notes on Changing the Evaluation Edition of a Materialized View

The following notes apply when changing the evaluation edition of a materialized view:

- If you change the evaluation edition of a refresh-on-commit materialized view, then Oracle Database performs a complete refresh of the materialized view unless you specify CONSIDER FRESH.
- If you change the evaluation edition of a refresh-on-demand materialized view, then Oracle Database sets the staleness state of the materialized view to STALE unless you specify CONSIDER FRESH.
- For both refresh-on-commit and refresh-on-demand materialized views: If you change the evaluation edition and specify CONSIDER FRESH, then Oracle Database does not update the staleness state of the materialized view and does not rebuild the materialized view. Therefore, you can specify CONSIDER FRESH to indicate that, although the evaluation edition has changed, there is no difference in the results that *subquery* will produce. If the materialized view is stale and in need of either a fast refresh or a complete refresh before this statement is issued, then the state will not be changed and the materialized view may contain bad data.

{ ENABLE | DISABLE } ON QUERY COMPUTATION

This clause lets you control whether the materialized view is a real-time materialized view or a regular materialized view.

- Specify ENABLE ON QUERY COMPUTATION to convert a regular materialized view into a real-time materialized view by enabling on-query computation.
- Specify DISABLE ON QUERY COMPUTATION to convert a real-time materialized view into a regular materialized view by disabling on-query computation.

This clause has the same semantics in CREATE MATERIALIZED VIEW and ALTER MATERIALIZED VIEW statements. For complete information on this clause, refer to { ENABLE | DISABLE } ON QUERY COMPUTATION in the documentation on CREATE MATERIALIZED VIEW.

alter_query_rewrite_clause

Use this clause to specify whether the materialized view is eligible to be used for query rewrite.

ENABLE Clause

Specify ENABLE to enable the materialized view for query rewrite. If you currently specify, or previously specified, the *unusable_editions_clause* for the materialized view, then it is not enabled for query rewrite in the unusable editions.

> **See Also:**
>
> - "Enabling Query Rewrite: Example"
> - *Oracle Database Data Warehousing Guide* to learn how to use refresh statistics to monitor the performance of materialized view refresh operations

Restrictions on Enabling Materialized Views

Enabling materialized views is subject to the following restrictions:

- If the materialized view is in an invalid or unusable state, then it is not eligible for query rewrite in spite of the `ENABLE` mode.
- You cannot enable query rewrite if the materialized view was created totally or in part from a view.
- You can enable query rewrite only if all user-defined functions in the materialized view are `DETERMINISTIC`.

  > **See Also:**
  >
  > CREATE FUNCTION

- You can enable query rewrite only if expressions in the statement are repeatable. For example, you cannot include `CURRENT_TIME` or `USER`.

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

DISABLE Clause

Specify `DISABLE` if you do not want the materialized view to be eligible for use by query rewrite. If a materialized view is in the invalid state, then it is not eligible for use by query rewrite, whether or not it is disabled. However, a disabled materialized view can be refreshed.

unusable_editions_clause

Use this clause to specify the editions in which the materialized view is not eligible for query rewrite. This clause has the same semantics in `CREATE MATERIALIZED VIEW` and `ALTER MATERIALIZED VIEW` statements. For complete information on this clause, refer to *unusable_editions_clause* in the documentation on `CREATE MATERIALIZED VIEW`.

Cursors that use the materialized view for query rewrite and were compiled in an edition that is made unusable will be invalidated.

ENABLE | DISABLE CONCURRENT REFRESH

Enable concurrent refresh to refresh the same on-commit atomic materialized view across mulitple sessions. Multiple sessions which have DML on a base table can refresh the MV concurrently. There are no limitations on how many materialized views can be refreshed.

Concurrent refresh is disabled by default. You must explictly enable it on the materialized view. Note that you can enable it only on on-commit materialized views.

> **See Also:**
>
> *Refreshing Materialized Views* of the *Oracle Database Data Warehousing Guide*.

COMPILE

Specify `COMPILE` to explicitly revalidate a materialized view. If an object upon which the materialized view depends is dropped or altered, then the materialized view remains accessible, but it is invalid for query rewrite. You can use this clause to explicitly revalidate the materialized view to make it eligible for query rewrite.

If the materialized view fails to revalidate, then it cannot be refreshed or used for query rewrite.

> **See Also:**
>
> "Compiling a Materialized View: Example"

CONSIDER FRESH

This clause lets you manage the staleness state of a materialized view after changes have been made to its master tables. `CONSIDER FRESH` directs Oracle Database to consider the materialized view fresh and therefore eligible for query rewrite in the `TRUSTED` or `STALE_TOLERATED` modes.

> **Caution:**
>
> The `CONSIDER FRESH` clause also directs Oracle Database to no longer apply any rows in a materialized view log or Partition Change Tracking changes to the materialized view prior to the issuance of the `CONSIDER FRESH` clause. In other words, the pending changes will be ignored and deleted, not applied to the materialized view. This may result in the materialized view containing more or less data than the base table.

Because Oracle Database cannot guarantee the freshness of the materialized view, query rewrite in `ENFORCED` mode is not supported. This clause also sets the staleness state of the materialized view to `UNKNOWN`. The staleness state is displayed in the `STALENESS` column of the `ALL_MVIEWS`, `DBA_MVIEWS`, and `USER_MVIEWS` data dictionary views.

A materialized view is stale if changes have been made to the contents of any of its master tables. This clause directs Oracle Database to assume that the materialized view is fresh and that no such changes have been made. Therefore, actual updates to those tables pending refresh are purged with respect to the materialized view.

> **See Also:**
>
> - *Oracle Database Data Warehousing Guide* for more information on query rewrite and the implications of performing partition maintenance operations on master tables
> - "CONSIDER FRESH: Example"

Examples

Automatic Refresh: Examples

The following statement changes the default refresh method for the `sales_by_month_by_state` materialized view (created in "Creating Materialized Aggregate Views: Example") to `FAST`:

```
ALTER MATERIALIZED VIEW sales_by_month_by_state
   REFRESH FAST;
```

The next automatic refresh of the materialized view will be a fast refresh provided it is a simple materialized view and its master table has a materialized view log that was created before the materialized view was created or last refreshed.

Because the `REFRESH` clause does not specify `START WITH` or `NEXT` values, Oracle Database will use the refresh intervals established by the `REFRESH` clause when the `sales_by_month_by_state` materialized view was created or last altered.

The following statement establishes a new interval between automatic refreshes for the `sales_by_month_by_state` materialized view:

```
ALTER MATERIALIZED VIEW sales_by_month_by_state
   REFRESH NEXT SYSDATE+7;
```

Because the `REFRESH` clause does not specify a `START WITH` value, the next automatic refresh occurs at the time established by the `START WITH` and `NEXT` values specified when the `sales_by_month_by_state` materialized view was created or last altered.

At the time of the next automatic refresh, Oracle Database refreshes the materialized view, evaluates the `NEXT` expression `SYSDATE`+7 to determine the next automatic refresh time, and continues to refresh the materialized view automatically once a week. Because the `REFRESH` clause does not explicitly specify a refresh method, Oracle Database continues to use the refresh method specified by the `REFRESH` clause of the `CREATE MATERIALIZED VIEW` or most recent `ALTER MATERIALIZED VIEW` statement.

CONSIDER FRESH: Example

The following statement instructs Oracle Database that materialized view `sales_by_month_by_state` should be considered fresh. This statement allows `sales_by_month_by_state` to be eligible for query rewrite in `TRUSTED` mode even after you have performed partition maintenance operations on the master tables of `sales_by_month_by_state`:

```
ALTER MATERIALIZED VIEW sales_by_month_by_state CONSIDER FRESH;
```

As a result of the preceding statement, any partition maintenance operations that were done to the base table since the last refresh of the materialized view will not be applied to the materialized view. For example, the add, drop, or change of data in a partition in the base table

will not be reflected in the materialized view if `CONSIDER FRESH` is used before the next refresh of the materialized view. Refer to CONSIDER FRESH for more information.

> **See Also:**
>
> "Splitting Table Partitions: Examples" for a partitioning maintenance example that would require this `ALTER MATERIALIZED VIEW` example

Complete Refresh: Example

The following statement specifies a new refresh method, a new `NEXT` refresh time, and a new interval between automatic refreshes of the `emp_data` materialized view (created in "Periodic Refresh of Materialized Views: Example"):

```
ALTER MATERIALIZED VIEW emp_data
   REFRESH COMPLETE
   START WITH TRUNC(SYSDATE+1) + 9/24
   NEXT SYSDATE+7;
```

The `START WITH` value establishes the next automatic refresh for the materialized view to be 9:00 a.m. tomorrow. At that point, Oracle Database performs a complete refresh of the materialized view, evaluates the `NEXT` expression, and subsequently refreshes the materialized view every week.

Enabling Query Rewrite: Example

The following statement enables query rewrite on the materialized view `emp_data` and implicitly revalidates it:

```
ALTER MATERIALIZED VIEW emp_data
   ENABLE QUERY REWRITE;
```

Primary Key Materialized View: Example

The following statement changes the rowid materialized view `order_data` (created in "Creating Rowid Materialized Views: Example") to a primary key materialized view. This example requires that you have already defined a materialized view log with a primary key on `order_data`.

```
ALTER MATERIALIZED VIEW order_data
   REFRESH WITH PRIMARY KEY;
```

Compiling a Materialized View: Example

The following statement revalidates the materialized view `store_mv`:

```
ALTER MATERIALIZED VIEW order_data COMPILE;
```

Drop Annotation from View: Example

The following example drops annotation `Snapshot` from view `MView1`:

```
ALTER MATERIALIZED VIEW MView1 ANNOTATIONS(DROP Snapshot);
```

ALTER MATERIALIZED VIEW LOG

Purpose

A **materialized view log** is a table associated with the master table of a materialized view. Use the `ALTER MATERIALIZED VIEW LOG` statement to alter the storage characteristics or type of an existing materialized view log.

> **Note:**
>
> The keyword `SNAPSHOT` is supported in place of `MATERIALIZED VIEW` for backward compatibility.

> **See Also:**
>
> - CREATE MATERIALIZED VIEW LOG for information on creating a materialized view log
> - ALTER MATERIALIZED VIEW for more information on materialized views, including refreshing them
> - CREATE MATERIALIZED VIEW for a description of the various types of materialized views

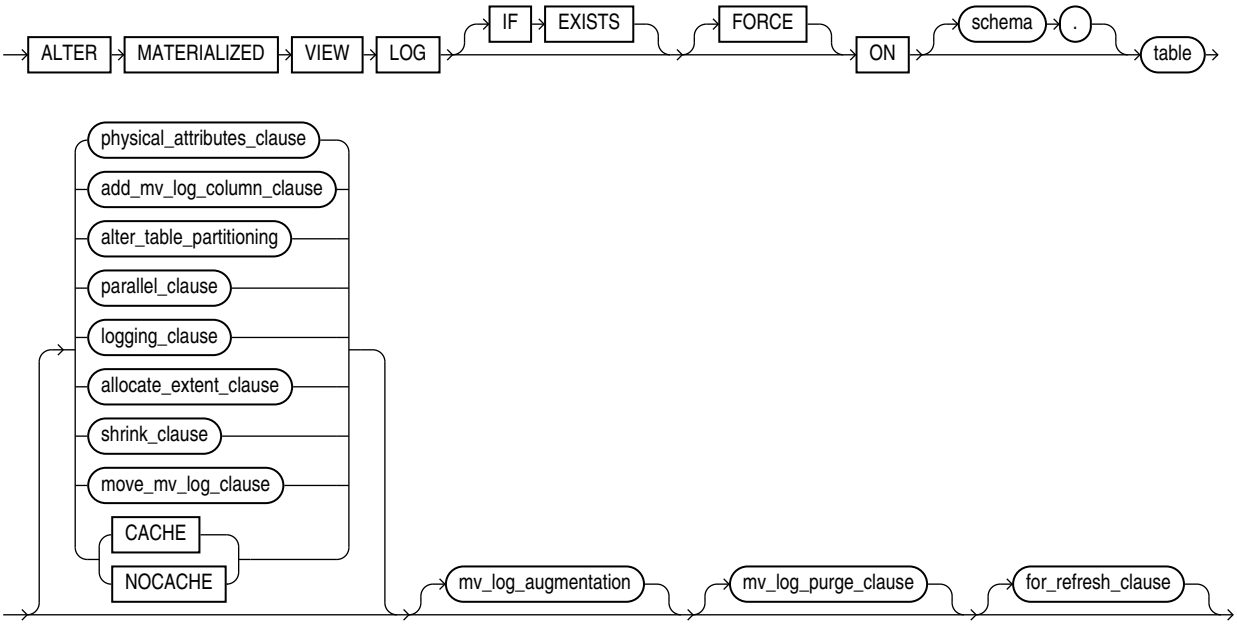
Prerequisites

You must be the owner of the master table, or you must have the `READ` or `SELECT` privilege on the master table and the `ALTER` privilege on the materialized view log.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for detailed information about the prerequisites for `ALTER MATERIALIZED VIEW LOG`

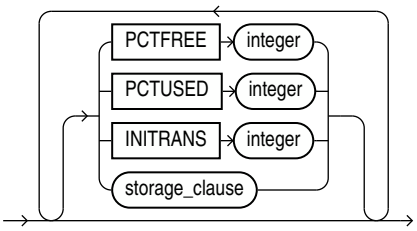
Syntax

***alter_materialized_view_log*::=**



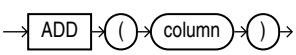
(*physical_attributes_clause*::=, *add_mv_log_column_clause*::=, *alter_table_partitioning*::= (in `ALTER TABLE`), *parallel_clause*::=, *logging_clause*::=, *allocate_extent_clause*::=, *shrink_clause*::=, *move_mv_log_clause*::=, *mv_log_augmentation*::=, *mv_log_purge_clause*::=, *for_refresh_clause*::=)

***physical_attributes_clause*::=**

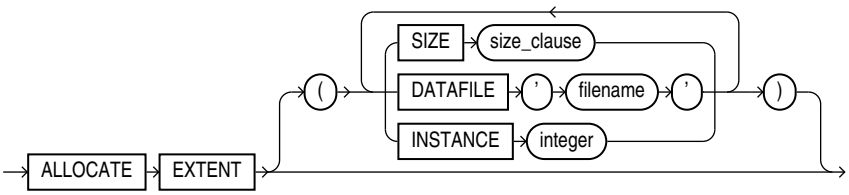


storage_clause::=

***add_mv_log_column_clause*::=**

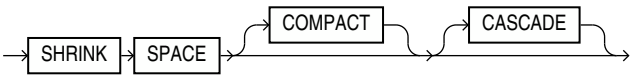


***allocate_extent_clause*::=**

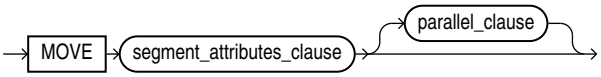


(*size_clause*::=)

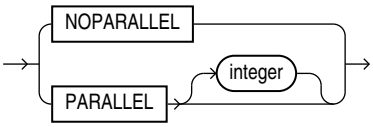
***shrink_clause*::=**



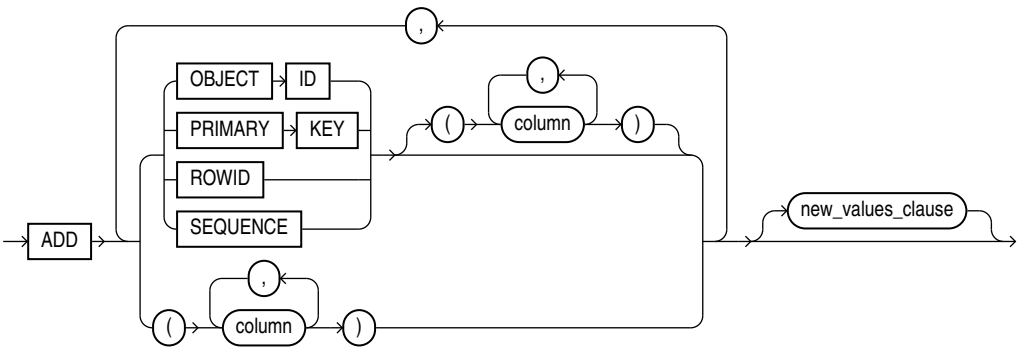
***move_mv_log_clause*::=**



***parallel_clause*::=**

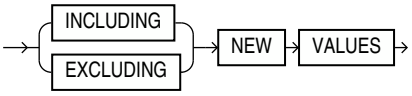


***mv_log_augmentation*::=**

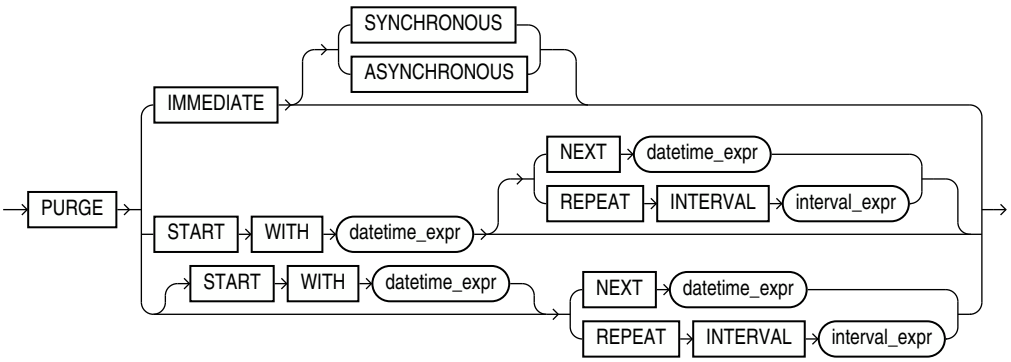


(*new_values_clause*::=

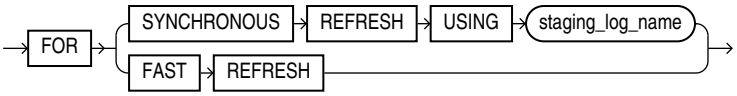
***new_values_clause*::=**



***mv_log_purge_clause*::=**



***for_refresh_clause*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to alter an existing table.

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

FORCE

If you specify `FORCE` and any items specified with the `ADD` clause have already been specified for the materialized view log, then Oracle Database does not return an error, but silently ignores the existing elements and adds to the materialized view log any items that do not already exist in the log. Likewise, if you specify `INCLUDING NEW VALUES` and that attribute has already been specified for the materialized view log, Oracle Database ignores the redundancy and does not return an error.

schema

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

table

Specify the name of the master table associated with the materialized view log to be altered.

physical_attributes_clause

The *physical_attributes_clause* lets you change the value of the PCTFREE, PCTUSED, and INITRANS parameters and the storage characteristics for the materialized view log, the partition, the overflow data segment, or the default characteristics of a partitioned materialized view log.

Restriction on Materialized View Log Physical Attributes

You cannot use the *storage_clause* to modify extent parameters if the materialized view log resides in a locally managed tablespace. Refer to CREATE TABLE for a description of these parameters.

add_mv_log_column_clause

When you add a column to the master table of the materialized view log, the database does not automatically add a column to the materialized view log. Therefore, use this clause to add a column to the materialized view log. Oracle Database will encrypt the newly added column if the corresponding column of the master table is encrypted.

alter_table_partitioning

The syntax and general functioning of the partitioning clauses is the same as described for the ALTER TABLE statement. Refer to *alter_table_partitioning* in the documentation on ALTER TABLE.

Restrictions on Altering Materialized View Log Partitions

Altering materialized view log partitions is subject to the following restrictions:

- You cannot use the *LOB_storage_clause* or *modify_LOB_storage_clause* when modifying partitions of a materialized view log.
- If you attempt to drop, truncate, or exchange a materialized view log partition, then Oracle Database raises an error.

parallel_clause

The *parallel_clause* lets you specify whether parallel operations will be supported for the materialized view log.

For complete information on this clause, refer to *parallel_clause* in the documentation on CREATE TABLE.

logging_clause

Specify the logging attribute of the materialized view log. Refer to the *logging_clause* for a full description of this clause.

allocate_extent_clause

Use the *allocate_extent_clause* to explicitly allocate a new extent for the materialized view log. Refer to *allocate_extent_clause* for a full description of this clause.

shrink_clause

Use this clause to compact the materialized view log segments. For complete information on this clause, refer to *shrink_clause* in the documentation on `CREATE TABLE`.

move_mv_log_clause

Use the `MOVE` clause to move the materialized view log table to a different tablespace, to change other segment or storage attributes of the materialized view log, or to change the parallelism of the materialized view log.

Restriction on Moving Materialized View Logs

The `ENCRYPT` clause of the *storage_clause* of *segment_attributes* is not valid for materialized view logs.

CACHE | NOCACHE Clause

For data that will be accessed frequently, `CACHE` specifies that the blocks retrieved for this log are placed at the most recently used end of the LRU list in the buffer cache when a full table scan is performed. This attribute is useful for small lookup tables. `NOCACHE` specifies that the blocks are placed at the least recently used end of the LRU list. Refer to "CACHE | NOCACHE | CACHE READS" in the documentation on `CREATE TABLE` for more information about this clause.

mv_log_augmentation

Use the `ADD` clause to augment the materialized view log so that it records the primary key values, rowid values, object ID values, or a sequence when rows in the materialized view master table are changed. This clause can also be used to record additional columns.

To stop recording any of this information, you must first drop the materialized view log and then re-create it. Dropping the materialized view log and then re-creating it forces a complete refresh for each of the existing materialized views that depend on the master table on its next refresh.

Restriction on Augmenting Materialized View Logs

You can specify only one `PRIMARY KEY`, one `ROWID`, one `OBJECT ID`, one `SEQUENCE`, and each column in the column list once for each materialized view log. You can specify only a single occurrence of `PRIMARY KEY`, `ROWID`, `OBJECT ID`, `SEQUENCE`, and column list within this `ALTER` statement. Also, if any of these values was specified at create time (either implicitly or explicitly), you cannot specify that value in this `ALTER` statement unless you use the `FORCE` option.

OBJECT ID

Specify `OBJECT ID` if you want the appropriate object identifier of all rows that are changed to be recorded in the materialized view log.

Restriction on the OBJECT ID clause

You can specify `OBJECT ID` only for logs on object tables, and you cannot specify it for storage tables.

PRIMARY KEY

Specify `PRIMARY KEY` if you want the primary key values of all rows that are changed to be recorded in the materialized view log.

ROWID

Specify `ROWID` if you want the rowid values of all rows that are changed to be recorded in the materialized view log.

SEQUENCE

Specify `SEQUENCE` to indicate that a sequence value providing additional ordering information should be recorded in the materialized view log.

column

Specify the additional columns whose values you want to be recorded in the materialized view log for all rows that are changed. Typically these columns are filter columns (non-primary-key columns referenced by subquery materialized views) and join columns (non-primary-key columns that define a join in the `WHERE` clause of the subquery).

> **See Also:**
>
> - CREATE MATERIALIZED VIEW for details on explicit and implicit inclusion of materialized view log values
> - *Oracle Database Administrator's Guide* for more information about filter columns and join columns
> - "Rowid Materialized View Log: Example"

NEW VALUES Clause

The `NEW VALUES` clause lets you specify whether Oracle Database saves both old and new values for update DML operations in the materialized view log. The value you set in this clause applies to all columns in the log, not only to columns you may have added in this `ALTER MATERIALIZED VIEW LOG` statement.

INCLUDING

Specify `INCLUDING` to save both new and old values in the log. If this log is for a table on which you have a single-table materialized aggregate view, and if you want the materialized view to be eligible for fast refresh, then you must specify `INCLUDING`.

EXCLUDING

Specify `EXCLUDING` to disable the recording of new values in the log. You can use this clause to avoid the overhead of recording new values.

If you have a fast-refreshable single-table materialized aggregate view defined on this table, then do not specify `EXCLUDING NEW VALUES` unless you first change the refresh mode of the materialized view to something other than `FAST`.

> **See Also:**
>
> "Materialized View Log EXCLUDING NEW VALUES: Example"

mv_log_purge_clause

Use this clause alter the purge attributes of the materialized view log in the following ways:

- Change the purge from `IMMEDIATE SYNCHRONOUS` to `IMMEDIATE ASYNCHRONOUS` or from `IMMEDIATE ASYNCHRONOUS` to `IMMEDIATE SYNCHRONOUS`
- Change the purge from `IMMEDIATE` to scheduled or from scheduled to `IMMEDIATE`
- Specify a new start time and a new `next` time and `interval`

If you are altering purge from scheduled to `IMMEDIATE`, then the scheduled purged job associated with that materialized view log is dropped. If you are altering purge from `IMMEDIATE` to scheduled, then a purge job is created with the attributes provided. If you are altering scheduled purge attributes, then only those attributes specified will be changed in the scheduler purge job.

You must specify `FORCE` if you are altering log purge to its current state (that is, you are not making any change), unless you are changing scheduled purge attributes.

To learn whether the purge time or interval has already been set for this materialized view log, query the `*_MVIEW_LOGS` data dictionary views. See the `CREATE MATERIALIZED VIEW LOG` clause *mv_log_purge_clause* for the full semantics of this clause.

for_refresh_clause

Use this clause to change the refresh method for which the materialized view log will be used.

FOR SYNCHRONOUS REFRESH

Specify this clause to change from fast refresh to synchronous refresh, or complete refresh to synchronous refresh. A staging log will be created.

If you are changing from fast refresh, then ensure that the following conditions are satisfied before using this clause:

- All changes in the materialized view log have been consumed.
- Any refresh-on-demand materialized views associated with the master table have been refreshed.
- Any refresh-on-commit materialized views associated with the master table have been converted to refresh-on-demand materialized views.

After you use this clause, you cannot perform DML operations directly on the master table. You must use the procedures in the `DBMS_SYNC_REFRESH` package to prepare and execute change data operations.

FOR FAST REFRESH

Specify this clause to change from synchronous refresh to fast refresh, or complete refresh to fast refresh. A materialized view log will be created.

If you are changing from synchronous refresh to fast refresh, then ensure that all changes in the staging log have been consumed before using this clause.

After you use this clause, you can perform DML operations directly on the master table.

See the `CREATE MATERIALIZED VIEW LOG` clause *for_refresh_clause* for the full semantics of this clause.

Examples

Rowid Materialized View Log: Example

The following statement alters an existing primary key materialized view log to also record rowid information:

```
ALTER MATERIALIZED VIEW LOG ON order_items ADD ROWID;
```

Materialized View Log EXCLUDING NEW VALUES: Example

The following statement alters the materialized view log on `hr.employees` by adding a filter column and excluding new values. Any materialized aggregate views that use this log will no longer be fast refreshable. However, if fast refresh is no longer needed, this action avoids the overhead of recording new values:

```
ALTER MATERIALIZED VIEW LOG ON employees
   ADD (commission_pct)
   EXCLUDING NEW VALUES;
```

ALTER MATERIALIZED ZONEMAP

Purpose

Use the `ALTER MATERIALIZED ZONEMAP` statement to modify an existing zone map in one of the following ways:

- To change its attributes
- To change its default refresh method and mode
- To enable or disable its use for pruning
- To compile it, rebuild it, or make it unusable

> **See Also:**
>
> - CREATE MATERIALIZED ZONEMAP for information on creating zone maps
> - *Oracle Database Data Warehousing Guide* for more information on zone maps

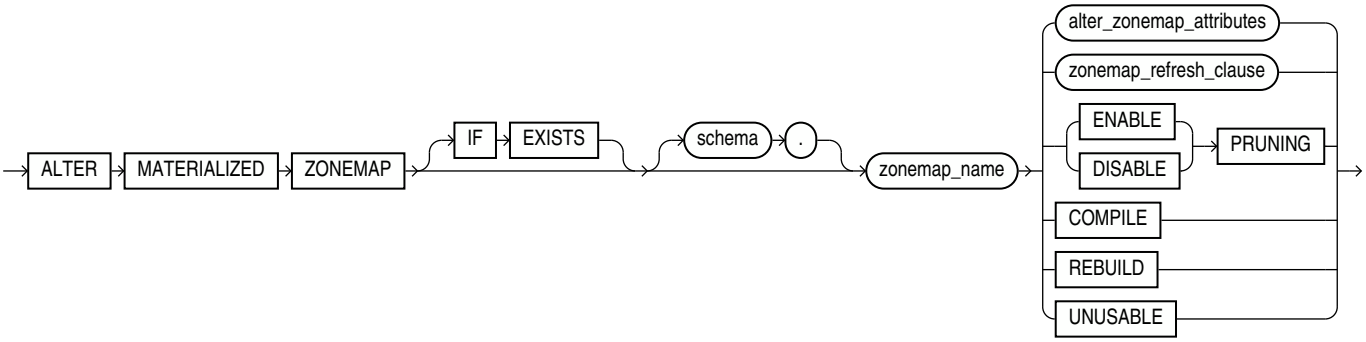
Prerequisites

The zone map must be in your own schema or you must have the `ALTER ANY MATERIALIZED VIEW` system privilege.

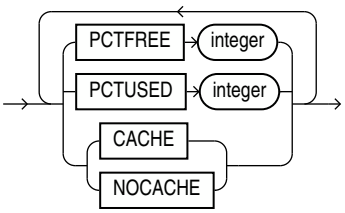
The user who owns the schema containing the zone map must have access to any base tables of the zone map that reside outside of that schema, either through a `READ` or `SELECT` object privilege on each of the tables, or through the `READ ANY TABLE` or `SELECT ANY TABLE` system privilege.

Syntax

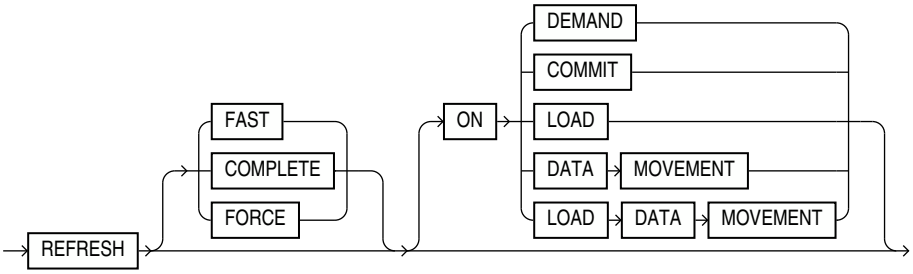
***alter_materialized_zonemap*::=**



***alter_zonemap_attributes*::=**



***zonemap_refresh_clause*::=**



> **Note:**
>
> When specifying the *zonemap_refresh_clause*, you must specify at least one clause after the REFRESH keyword.

Semantics

IF EXISTS

Specify IF EXISTS to alter an existing table.

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

schema

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

zonemap_name

Specify the name of the zone map to be altered.

alter_zonemap_attributes

Use this clause to modify the following attributes for the zone map: `PCTFREE`, `PCTUSED`, and `CACHE` or `NOCACHE`. These attributes have the same semantics for `ALTER MATERIALIZED ZONEMAP` and `CREATE MATERIALIZED ZONEMAP`. For complete information on these attributes, refer to PCTFREE, PCTUSED, and CACHE | NOCACHE in the documentation on `CREATE MATERIALIZED ZONEMAP`.

zonemap_refresh_clause

Use this clause to modify the default refresh method and mode for the zone map. This clause has the same semantics for `ALTER MATERIALIZED ZONEMAP` and `CREATE MATERIALIZED ZONEMAP`. For complete information on this clause, refer to *zonemap_refresh_clause* in the documentation on `CREATE MATERIALIZED ZONEMAP`.

ENABLE | DISABLE PRUNING

Use this clause to enable or disable use of the zone map for pruning. This clause has the same semantics for `ALTER MATERIALIZED ZONEMAP` and `CREATE MATERIALIZED ZONEMAP`. For complete information on this clause, refer to ENABLE | DISABLE PRUNING in the documentation on `CREATE MATERIALIZED ZONEMAP`

COMPILE

This clause lets you explicitly compile the zone map. This operation validates the zone map after a DDL operation changes the structure of one or more of its base tables. It is usually not necessary to issue this clause because Oracle database automatically compiles a zone map that requires compilation before using it. However, if you would like to explicitly compile a zone map, then you can use this clause to do so.

The result of compiling a zone map depends on whether a base table is changed in a way that affects the zone map. For example, if a column is added to a base table, then the zone map will be valid after compilation because the change does not affect the zone map. However, if a column that is included in the defining subquery of the zone map is dropped from a base table, then the zone map will be invalid after compilation.

You can determine if a zone map requires compilation by querying the `COMPILE_STATE` column of the `ALL_`, `DBA_`, and `USER_ZONEMAPS` data dictionary views. If the value of the column is `NEEDS_COMPILE`, then the zone map requires compilation.

REBUILD

This clause lets you explicitly rebuild the zone map. This operation refreshes the data in the zone map. This clause is useful in the following situations:

- You can use this clause to refresh the data for a refresh-on-demand zone map. Refer to the ON DEMAND clause in the documentation on CREATE MATERIALIZED ZONEMAP for more information.
- You must issue this clause after an EXCHANGE PARTITION operation on one of the base tables of a zone map, regardless of the default refresh mode of the zone map.
- If a zone map is marked unusable, then you must issue this clause to mark it usable. You can determine if a zone map is marked unusable by querying the UNUSABLE column of the ALL_, DBA_, and USER_ZONEMAPS data dictionary views.

UNUSABLE

Specify this clause to make the zone map unusable. Subsequent queries will not use the zone map and the database will no longer maintain the zone map. You can make the zone map usable again by issuing an ALTER MATERIALIZED ZONEMAP ... REBUILD statement.

Examples

Modifying Zone Map Attributes: Example

The following statement modifies the PCTFREE and PCTUSED attributes of zone map sales_zmap, and modifies the zone map so that it does not use caching:

```
ALTER MATERIALIZED ZONEMAP sales_zmap
  PCTFREE 20 PCTUSED 50 NOCACHE;
```

Modifying the Default Refresh Method and Mode for a Zone Map: Example

The following statement changes the default refresh method to FAST and the default refresh mode to ON COMMIT for zone map sales_zmap:

```
ALTER MATERIALIZED ZONEMAP sales_zmap
  REFRESH FAST ON COMMIT;
```

Disabling Use of a Zone Map for Pruning: Example

The following statement disables use of zone map sales_zmap for pruning:

```
ALTER MATERIALIZED ZONEMAP sales_zmap
  DISABLE PRUNING;
```

Compiling a Zone Map: Example

The following statement compiles zone map sales_zmap:

```
ALTER MATERIALIZED ZONEMAP sales_zmap
  COMPILE;
```

Rebuilding a Zone Map: Example

The following statement rebuilds zone map sales_zmap:

```
ALTER MATERIALIZED ZONEMAP sales_zmap
  REBUILD;
```

Making a Zone Map Unusable: Example

The following statement makes zone map sales_zmap unusable:

```
ALTER MATERIALIZED ZONEMAP sales_zmap
  UNUSABLE;
```

ALTER MLE ENV

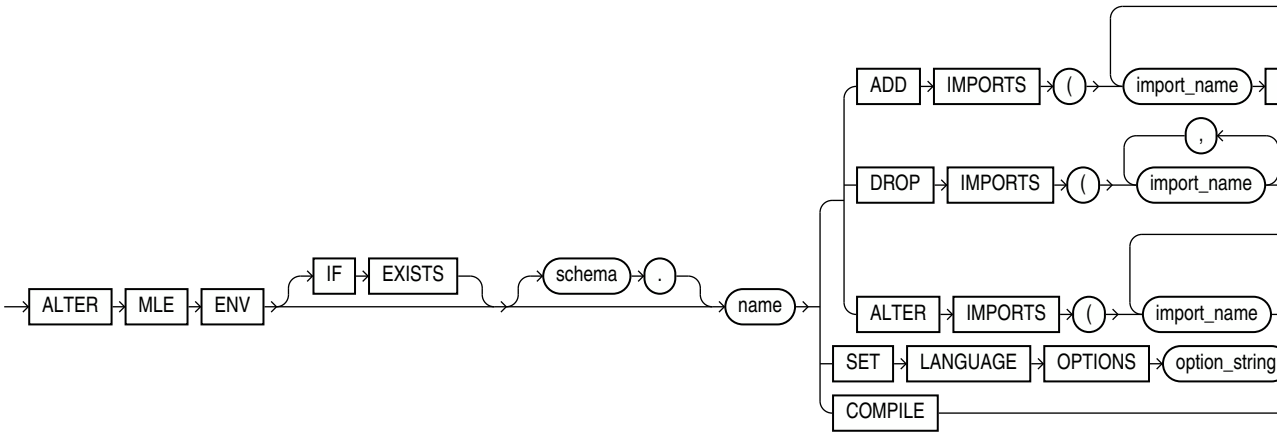
Purpose

Use `ALTER MLE ENV` to alter an exisiting MLE environment. You can add, remove, and alter mappings for import names and set language options.

Prerequisites

You must have the `ALTER ANY MLE` privilege to alter an environment in schemas other than your own. No privilege is needed to alter an environment in your own schema.

Syntax



Semantics

IF EXISTS

The `ALTER MLE MODULE` statement raises an `ORA-04103` error if the module does not exist, or an `ORA-00922` error if an invalid attribute is specified.

schema

Specify the schema containing the MLE module. If you do not specify the schema, then Oracle Database assumes that the module is in your own schema.

ADD IMPORTS Clause

Use `ADD IMPORTS` to add new mappings from import names to MLE module schema objects. Mappings to be added are specified as a comma-separated list enclosed in parentheses. Each element in the list is of the form: *`import-name MODULE [schema]. mle-module-name`*.

The following cases produce errors:

- If the environment already contains one or more of the import names, an `ORA-04109` error is thrown.
- If one or more of the MLE modules does not reside in the same schema as the environment, an `ORA-01031` error is thrown.

DROP IMPORTS Clause

Use `DROP IMPORTS` to remove import names from the environment.

If the environment does nnot contain one or more of the specified import names, an `ORA-04110` error is thrown.

ALTER IMPORTS Clause

Use `ALTER IMPORTS` to update import mappings for each of the specified import names.

The following cases produce errors:

- If the environment does not contain one or more of the import names, an `ORA-04110` error is thrown.
- If one or more of the new MLE modules does not reside in the same schema as the environment, an `ORA-01031` error is thrown.

SET LANGUAGE OPTIONS Clause

Use `LANGUAGE OPTIONS` to specify language options for all execution contexts created with this environment. Language options are specified as a string literal consisting of comma-separated key-value pairs. Language options are only parsed at runtime when an execution context is created using the MLE environment.

If at context creation the language options string turns out to be invalid (invalid format, unsupported options), an `ORA-04152` error is thrown.

Example

The following example modifies an exisiting environment myenv by enabling JavaScript in strict mode:

```
ALTER MLE ENV scott."myenv" SET LANGUAGE OPTIONS 'js.strict=
true ';
```

COMPILE Clause

Use `COMPILE` to explicitly recompile an MLE environment. You can use this clause to revalidate an environment that has become invalid and thereby catch errors before run time.

> **See Also:**
>
> - *MLE JavaScript Modules and Environments*
> - CREATE MLE ENV
> - DROP MLE ENV

ALTER MLE MODULE

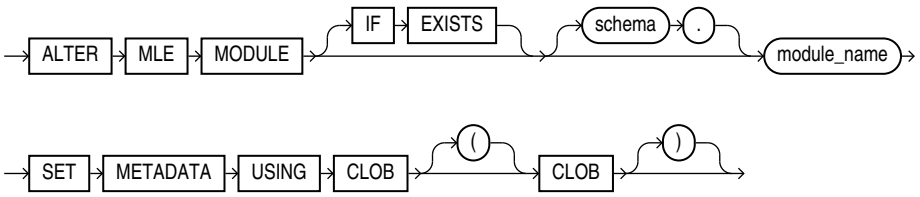
Purpose

Use `ALTER MLE MODULE` to add metadata to existing MLE modules in the database.

Prerequisites

To alter MLE modules in another schema you need the `ALTER ANY MLE` system privilege. No privileges are required to alter MLE modules in your own schema.

Syntax



Semantics

IF EXISTS

The `ALTER MLE MODULE` statement raises an `ORA-04103` error if the module does not exist, or an `ORA-00922` error if an invalid attribute is specified.

schema

Specify the schema containing the MLE module. If you do not specify the schema, then Oracle Database assumes that the module is in your own schema.

module_name

`module_name` refers to the name of the MLE module.

CLOB

`CLOB` refers to the text you can attach to an MLE module. You can use it to refer to a commit in a version control system or as a part of a Software Bill of Materials. The `CLOB` contents are freeform metadata that can be attached to the MLE module. The metadata does not impact module functionality in any way. Oracle recommends that the metadata be used to record version information for the MLE module, e.g., the commit in a version control system that corresponds to the deployed version of the module, or a Software Bill of Materials for the module, for example the contents of `package-lock.json` for a bundled JavaScript module.

Examples

The following example attaches metadata as `JSON` to MLE module `myMLEModule`:

```
ALTER MLE MODULE myMLEModule
SET METADATA USING CLOB (
SELECT JSON(
  '{
       "name": "value",
       "version": "1.2.0",
       "commitHash": "23fas4h",
       "projectName": "Database Backend"
  }')
)
```

> **See Also:**
>
> - *MLE JavaScript Modules and Environments*
> - CREATE MLE MODULE
> - DROP MLE MODULE

ALTER OPERATOR

Purpose

Use the `ALTER OPERATOR` statement to add bindings to, drop bindings from, or compile an existing operator.

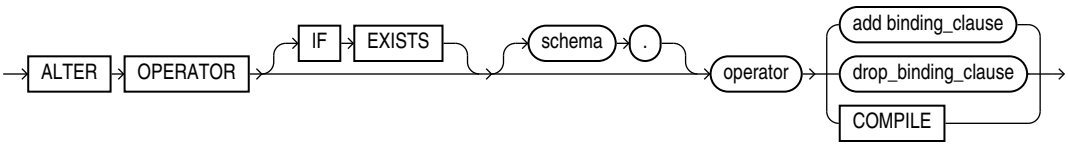
> **See Also:**
>
> CREATE OPERATOR

Prerequisites

The operator must already have been created by a previous `CREATE OPERATOR` statement. The operator must be in your own schema or you must have the `ALTER ANY OPERATOR` system privilege. You must have the `EXECUTE` object privilege on the operators and functions referenced in the `ALTER OPERATOR` statement.

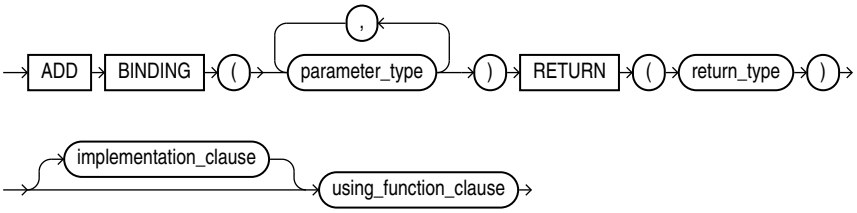
Syntax

***alter_operator*::=**



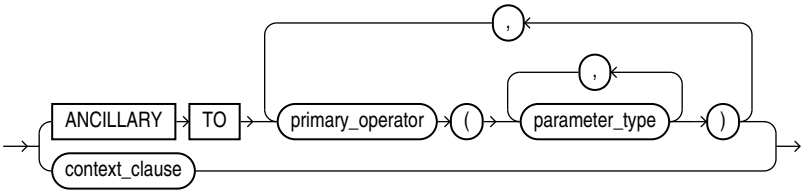
(*add_binding_clause*::=, *drop_binding_clause*::=)

***add_binding_clause*::=**



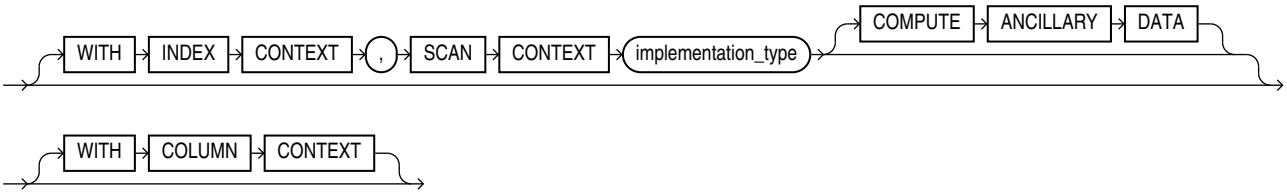
(*implementation_clause*::=, *using_function_clause*::=)

***implementation_clause*::=**

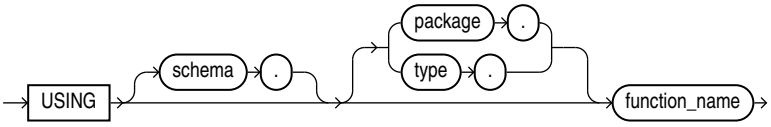


(*context_clause*::=)

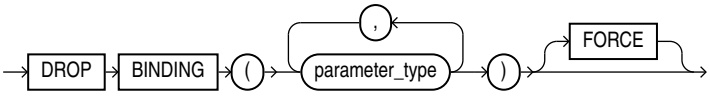
***context_clause*::=**



***using_function_clause*::=**



***drop_binding_clause*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to alter an existing table.

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

schema

Specify the schema containing the operator. If you omit this clause, then Oracle Database assumes the operator is in your own schema.

operator

Specify the name of the operator to be altered.

add_binding_clause

Use this clause to add an operator binding and specify its parameter data types and return type. The signature must be different from the signature of any existing binding for this operator.

If a binding of an operator is associated with an indextype and you add another binding to the operator, then Oracle Database does not automatically associate the new binding with the indextype. If you want to make such an association, then you must issue an explicit `ALTER INDEXTYPE ... ADD OPERATOR` statement.

implementation_clause

This clause has the same semantics in `CREATE OPERATOR` and `ALTER OPERATOR` statements. For full information, refer to *implementation_clause* in the documentation on `CREATE OPERATOR`.

context_clause

This clause has the same semantics in `CREATE OPERATOR` and `ALTER OPERATOR` statements. For full information, refer to context_clause in the documentation on `CREATE OPERATOR`.

using_function_clause

This clause has the same semantics in `CREATE OPERATOR` and `ALTER OPERATOR` statements. For full information, refer to *using_function_clause* in the documentation on `CREATE OPERATOR`.

drop_binding_clause

Use this clause to specify the list of parameter data types of the binding you want to drop from the operator. You must specify `FORCE` if the binding has any dependent objects, such as an indextype or an ancillary operator binding. If you specify `FORCE`, then Oracle Database marks `INVALID` all objects that are dependent on the binding. The dependent objects are revalidated the next time they are referenced in a DDL or DML statement or a query.

You cannot use this clause to drop the only binding associated with this operator. Instead you must use the `DROP OPERATOR` statement. Refer to DROP OPERATOR for more information.

COMPILE

Specify `COMPILE` to cause Oracle Database to recompile the operator.

Examples

Compiling a User-defined Operator: Example

The following example compiles the operator `eq_op` (which was created in "Creating User-Defined Operators: Example"):

```
ALTER OPERATOR eq_op COMPILE;
```

ALTER OUTLINE

Purpose

> **Note:**
>
> Stored outlines are deprecated. They are still supported for backward compatibility. However, Oracle recommends that you use SQL plan management instead. SQL plan management creates SQL plan baselines, which offer superior SQL performance stability compared with stored outlines.
>
> You can migrate existing stored outlines to SQL plan baselines by using the `MIGRATE_STORED_OUTLINE` function of the `DBMS_SPM` package or Enterprise Manager Cloud Control. When the migration is complete, the stored outlines are marked as migrated and can be removed. You can drop all migrated stored outlines on your system by using the `DROP_MIGRATED_STORED_OUTLINE` function of the `DBMS_SPM` package.
>
> See Also: *Oracle Database SQL Tuning Guide* for more information about SQL plan management and *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SPM` package

Use the `ALTER OUTLINE` statement to rename a stored outline, reassign it to a different category, or regenerate it by compiling the outline's SQL statement and replacing the old outline data with the outline created under current conditions.

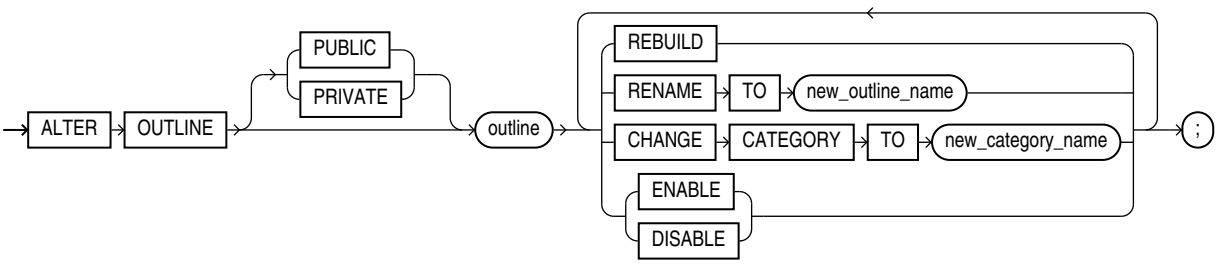
> **See Also:**
>
> CREATE OUTLINE for information on creating an outline

Prerequisites

To modify an outline, you must have the `ALTER ANY OUTLINE` system privilege.

Syntax

***alter_outline*::=**



Semantics

PUBLIC | PRIVATE

Specify `PUBLIC` if you want to modify the public version of this outline. This is the default.

Specify `PRIVATE` if you want to modify an outline that is private to the current session and whose data is stored in the current parsing schema.

outline

Specify the name of the outline to be modified.

REBUILD

Specify `REBUILD` to regenerate the execution plan for *`outline`* using current conditions.

> **See Also:**
>
> "Rebuilding an Outline: Example"

RENAME TO Clause

Use the `RENAME TO` clause to specify an outline name to replace *`outline`*.

CHANGE CATEGORY TO Clause

Use the `CHANGE CATEGORY TO` clause to specify the name of the category into which the *`outline`* will be moved.

ENABLE | DISABLE

Use this clause to selectively enable or disable this outline. Outlines are enabled by default. The `DISABLE` keyword lets you disable one outline without affecting the use of other outlines.

Examples

Rebuilding an Outline: Example

The following statement regenerates a stored outline called `salaries` by compiling the text of the outline and replacing the old outline data with the outline created under current conditions.

```
ALTER OUTLINE salaries REBUILD;
```

ALTER PACKAGE

Purpose

Packages are defined using PL/SQL. Therefore, this section provides some general information but refers to *Oracle Database PL/SQL Language Reference* for details of syntax and semantics.

Use the `ALTER PACKAGE` statement to explicitly recompile a package specification, body, or both. Explicit recompilation eliminates the need for implicit run-time recompilation and prevents associated run-time compilation errors and performance overhead.

Because all objects in a package are stored as a unit, the `ALTER PACKAGE` statement recompiles all package objects together. You cannot use the `ALTER PROCEDURE` statement or `ALTER FUNCTION` statement to recompile individually a procedure or function that is part of a package.

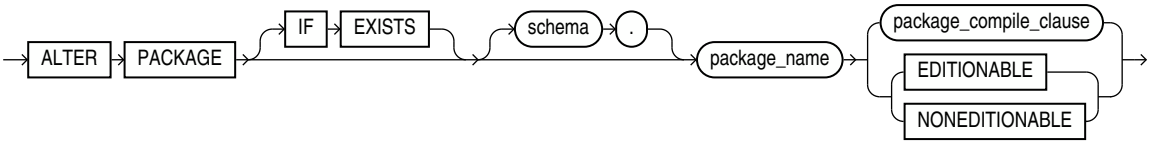
> **Note:**
>
> This statement does not change the declaration or definition of an existing package. To redeclare or redefine a package, use the CREATE PACKAGE or the CREATE PACKAGE BODY statement with the `OR REPLACE` clause.

Prerequisites

For you to modify a package, the package must be in your own schema or you must have `ALTER ANY PROCEDURE` system privilege.

Syntax

***alter_package*::=**



(*package_compile_clause*: See *Oracle Database PL/SQL Language Reference* for the syntax of this clause.)

Semantics

IF EXISTS

Specify `IF EXISTS` to alter an existing table.

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

schema

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

package_name

Specify the name of the package to be recompiled.

package_compile_clause

See *Oracle Database PL/SQL Language Reference* for the syntax and semantics of this clause and for complete information on creating and compiling packages.

EDITIONABLE | NONEDITIONABLE

Use these clauses to specify whether the package becomes an editioned or noneditioned object if editioning is later enabled for the schema object type `PACKAGE` in *schema*. The default is

EDITIONABLE. For information about altering editioned and noneditioned objects, see *Oracle Database Development Guide*.

ALTER PLUGGABLE DATABASE

Purpose

Use the ALTER PLUGGABLE DATABASE statement to modify a pluggable database (PDB). The PDB can be a traditional PDB, an application container, or an application PDB.

This statement enables you to perform the following tasks:

- Unplug a PDB from a multitenant container database (CDB) (using the *pdb_unplug_clause*)
- Modify the settings of a PDB (using the *pdb_settings_clauses*)
- Bring PDB data files online or take them offline (using the *pdb_datafile_clause*)
- Back up and recover a PDB (using the *pdb_recovery_clauses*)
- Modify the state of a PDB (using the *pdb_change_state* clause)
- Modify the state of multiple PDBs within a CDB (using the *pdb_change_state_from_root* clause)
- Perform operations on applications in an application container (using the *application_clauses*)
- Create and manage PDB snapshots using the *snapshot_clauses*

> **Note:**
>
> You can perform all ALTER PLUGGABLE DATABASE tasks by connecting to a PDB and running the corresponding ALTER DATABASE statement. This functionality is provided to maintain backward compatibility for applications that have been migrated to a CDB environment. The exception is modifying PDB storage limits, for which you must use the *pdb_storage_clause* of ALTER PLUGGABLE DATABASE.

> **See Also:**
>
> CREATE PLUGGABLE DATABASE for information on creating PDBs

Prerequisites

You must be connected to a CDB.

To specify the *pdb_unplug_clause*, the current container must be the root or the application root, you must be authenticated AS SYSDBA or AS SYSOPER, and the SYSDBA or SYSOPER privilege must be either granted to you commonly, or granted to you locally in the root and locally in the PDB you want to unplug.

To specify the *pdb_settings_clauses*, the current container must be the PDB whose settings you want to modify and you must have the ALTER DATABASE privilege, either granted commonly

or granted locally in the PDB. To specify the *pdb_logging_clauses* or the RENAME GLOBAL_NAME clause, you must also have the RESTRICTED SESSION privilege, either granted commonly or granted locally in the PDB being renamed, and the PDB must be in READ WRITE RESTRICTED mode.

To specify the *pdb_datafile_clause*, the current container must be the PDB whose datafiles you want to bring online or take offline and you must have the ALTER DATABASE privilege, either granted commonly or granted locally in the PDB.

To specify the *pdb_recovery_clauses*, the current container must be the PDB you want to back up or recover and you must have the ALTER DATABASE privilege, either granted commonly or granted locally in the PDB.

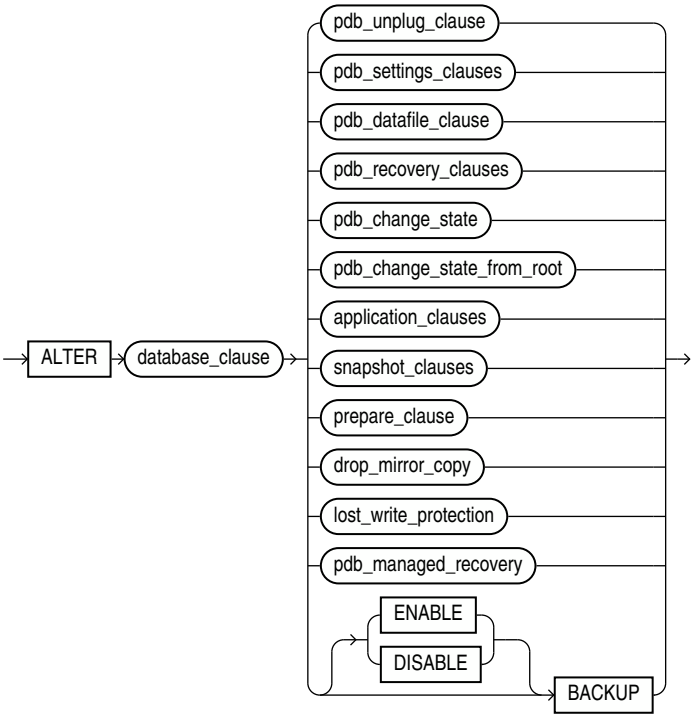
To specify the *pdb_change_state* clause, the current container must be the PDB whose state you want to change and you must be authenticated AS SYSBACKUP, AS SYSDBA, AS SYSDG, or AS SYSOPER.

To specify the *pdb_change_state_from_root* clause, the current container must be the root or the application root, you must be authenticated AS SYSBACKUP, AS SYSDBA, AS SYSDG, or AS SYSOPER, and the SYSBACKUP, SYSDBA, SYSDG, or SYSOPER privilege must be either granted to you commonly, or granted to you locally in the root or application root, and locally in the PDB(s) whose state(s) you want to change.

To specify the *application_clauses*, the current container must be an application container, you must be authenticated AS SYSBACKUP or AS SYSDBA, and the SYSBACKUP or SYSDBA privilege must be either granted to you commonly, or granted to you locally in the application root and locally in the application PDB(s) in which you want to perform application operations.

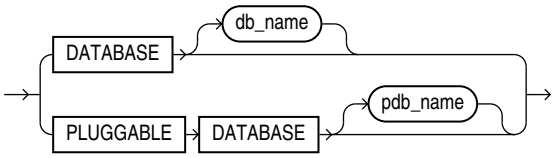
Syntax

***alter_pluggable_database*::=**



(*pdb_unplug_clause*::=, *pdb_settings_clauses*::=, *pdb_datafile_clause*::=, *pdb_recovery_clauses*, *pdb_change_state*::=, *pdb_change_state_from_root*::=, *application_clauses*::=)

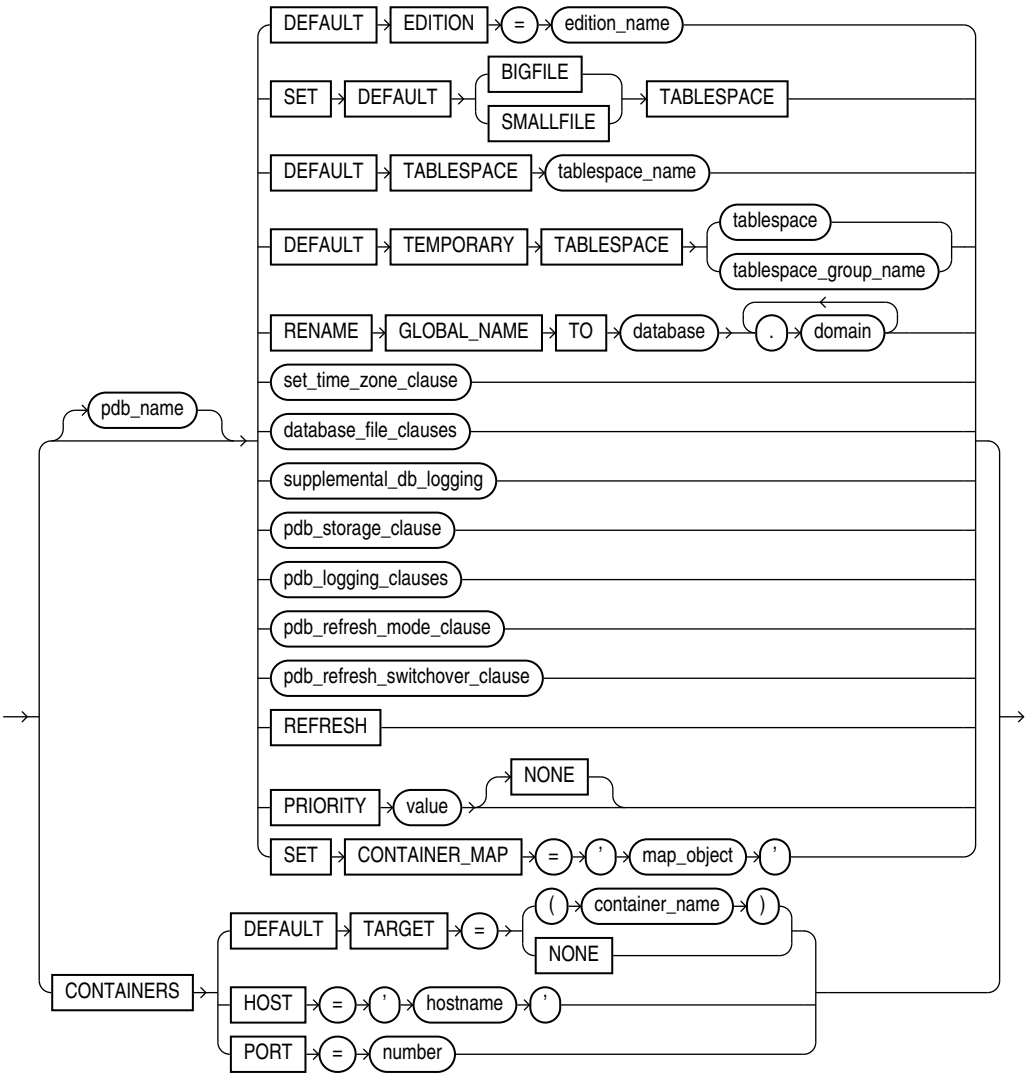
***database_clause*::=**



***pdb_unplug_clause*::=**

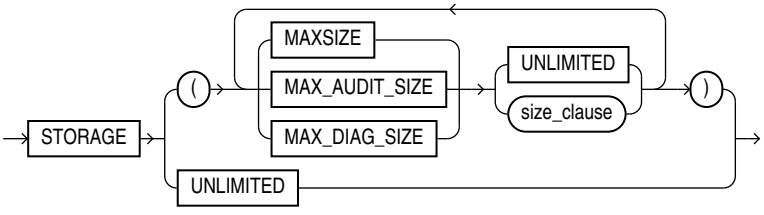
***pdb_unplug_encrypt*::=**

***pdb_settings_clauses*::=**



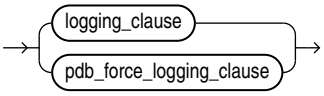
(*set_time_zone_clause*::=, *database_file_clauses*::=, *supplemental_db_logging*::=, *pdb_storage_clause*::=, *pdb_logging_clauses*::=, *pdb_refresh_mode_clause*::=)

***pdb_storage_clause*::=**

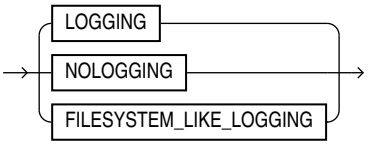


(*size_clause*::=)

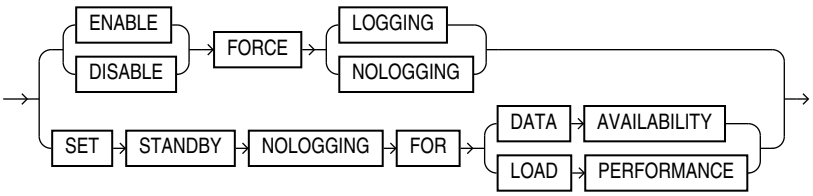
***pdb_logging_clauses*::=**



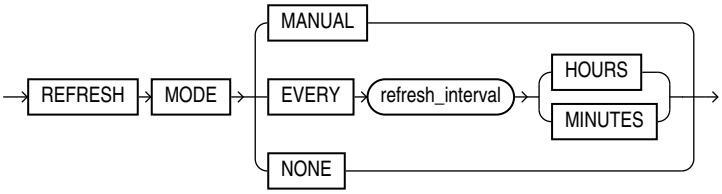
***logging_clause*::=**



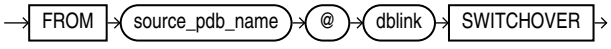
***pdb_force_logging_clause*::=**



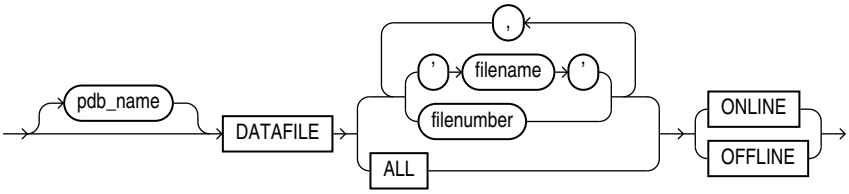
***pdb_refresh_mode_clause*::=**



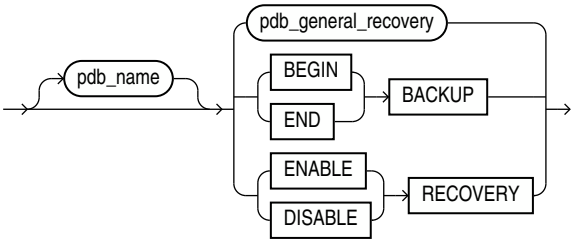
***pdb_refresh_switchover_clause* ::=**



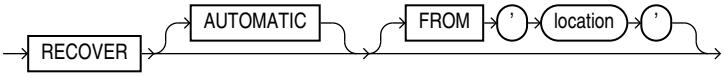
***pdb_datafile_clause*::=**

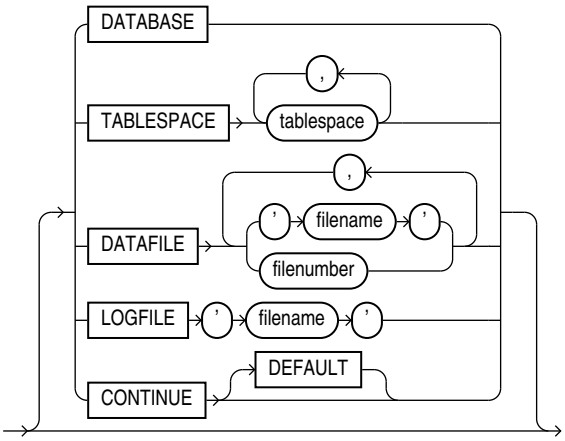


pdb_recovery_clauses

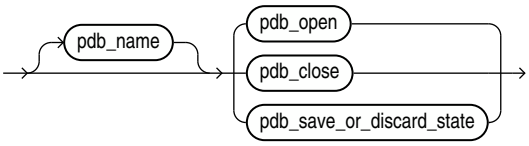


***pdb_general_recovery*::=**



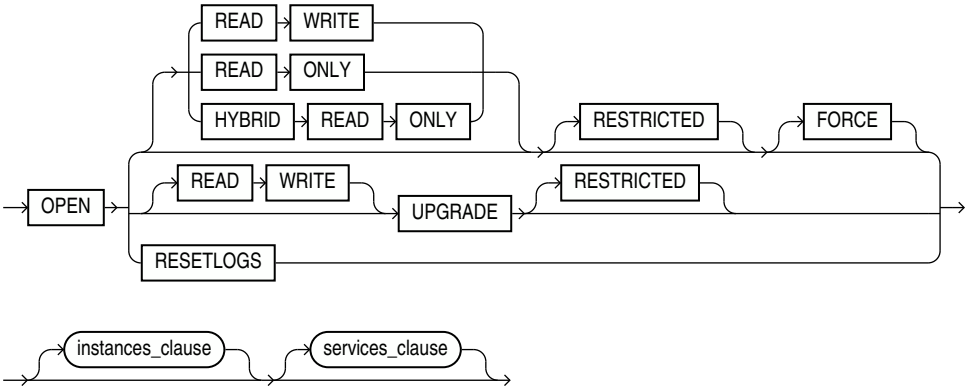


***pdb_change_state*::=**

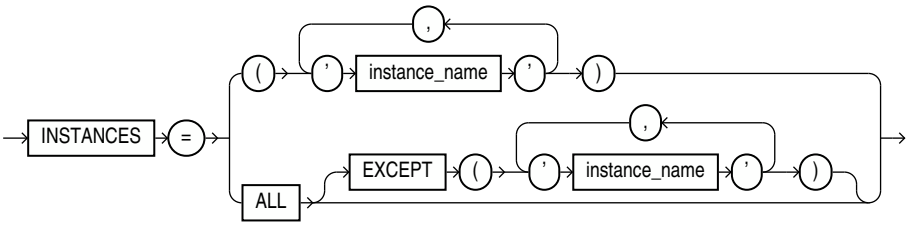


(*pdb_open*::=, *pdb_close*::=, *pdb_save_or_discard_state*::=)

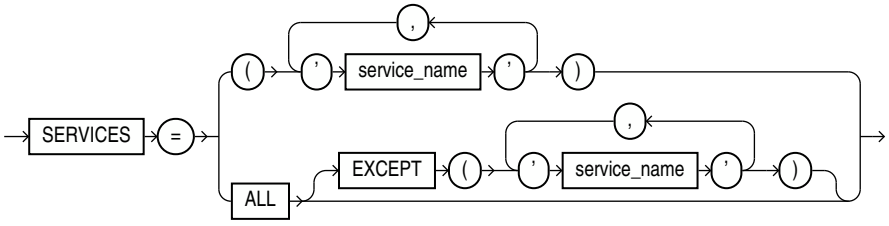
***pdb_open*::=**



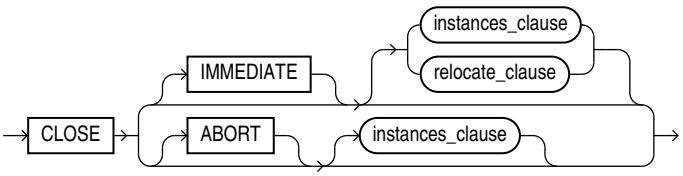
***instances_clause*::=**



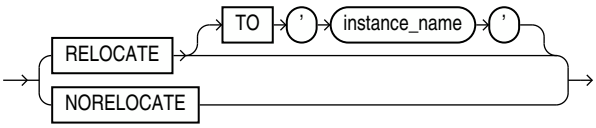
***services_clause*::=**



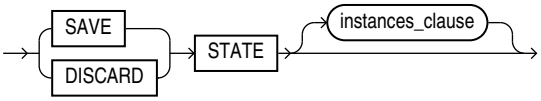
***pdb_close*::=**



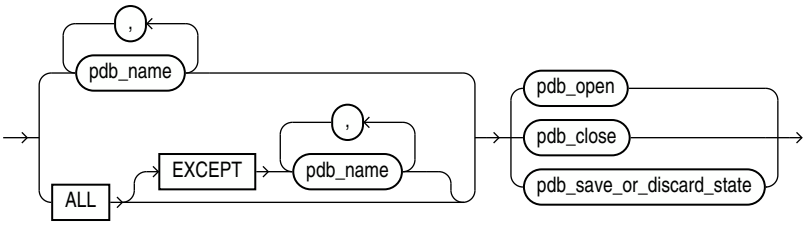
***relocate_clause*::=**



***pdb_save_or_discard_state*::=**

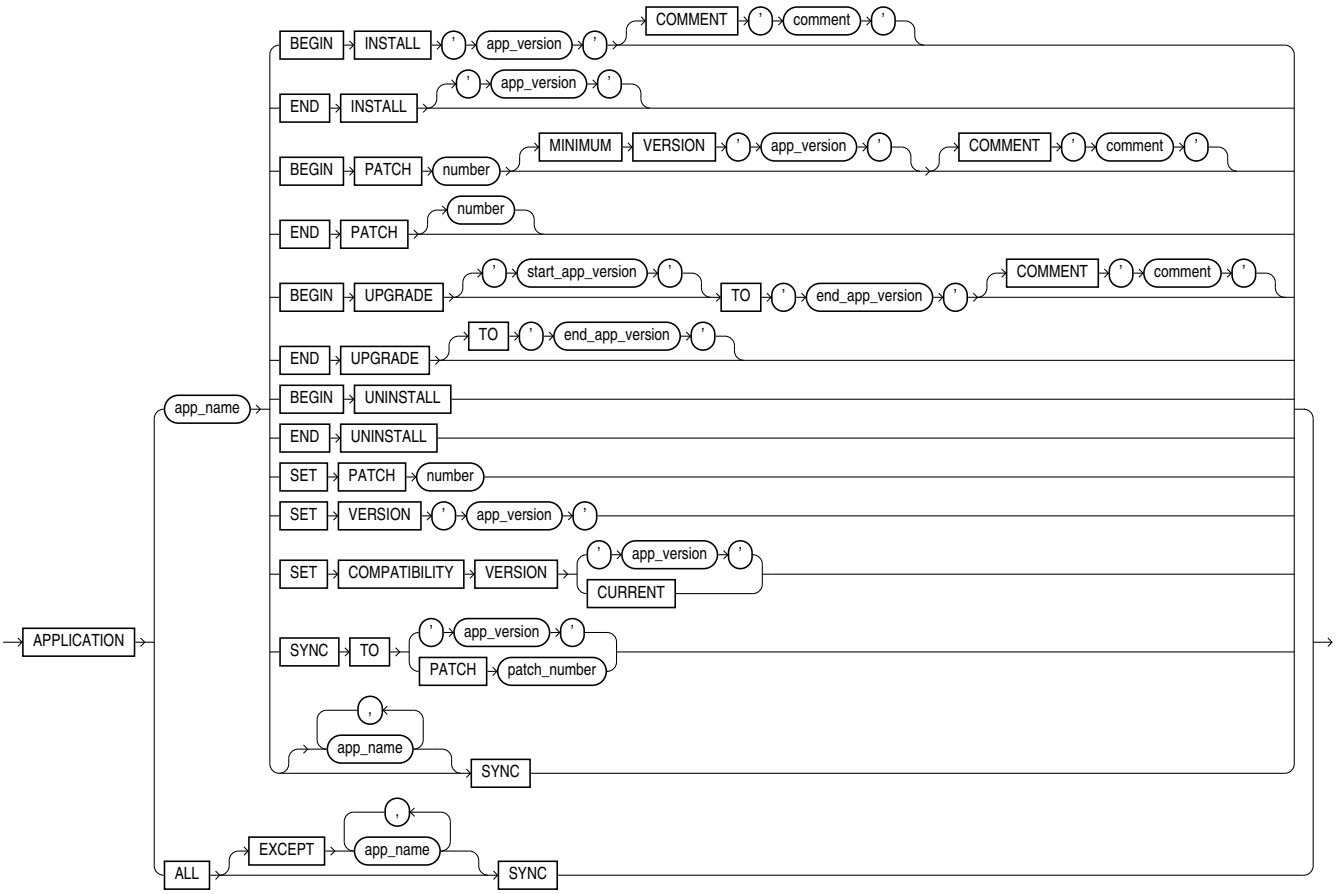


***pdb_change_state_from_root*::=**

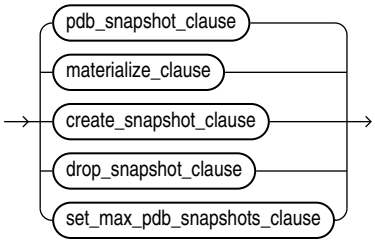


(*pdb_open*::=, *pdb_close*::=, *pdb_save_or_discard_state*::=)

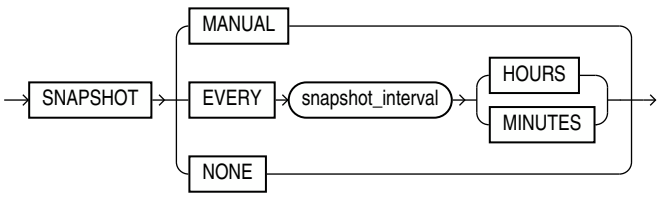
***application_clauses*::=**



***snapshot_clauses* ::=**

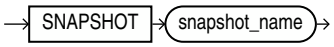


***pdb_snapshot_clause* ::=**

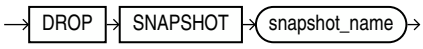


***materialize_clause*::=**

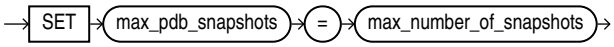
***create_snapshot_clause*::=**



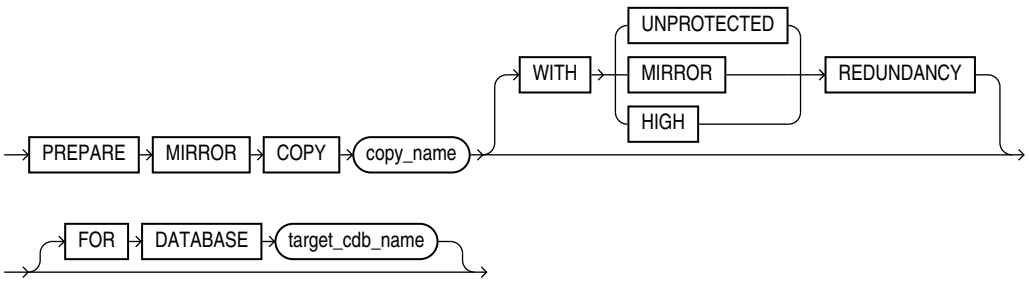
***drop_snapshot_clause*::=**



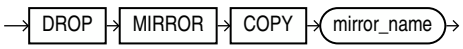
***set_max_pdb_snapshots*::=**



***prepare_clause*::=**



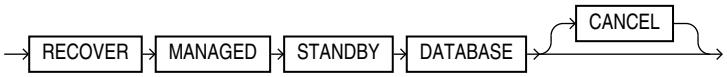
***drop_mirror_copy*::=**



***lost_write_protection* ::=**

The usage for the `lost_write_protection` clause with the `ALTER PLUGGABLE DATABASE` statement is identical to the `ALTER DATABASE` statement. Look here ALTER DATABASE for syntax details.

***pdb_managed_recovery* ::=**



Semantics

database_clause

Specify the `PLUGGABLE DATABASE` option for a container database.

pdb_name

Specify the name of the database to be altered. If you omit *db_name*, then Oracle Database alters the database identified by the value of the initialization parameter `DB_NAME`. You can alter only the database whose control files are specified by the initialization parameter `CONTROL_FILES`. The database identifier is not related to the Oracle Net database specification.

pdb_unplug_clause

This clause lets you unplug a PDB from a CDB. When you unplug a PDB, Oracle stores information about the PDB in a file on your operating system. You can subsequently use this file to plug the PDB into a CDB.

For *pdb_name*, specify the name of the PDB you want to unplug. The PDB must be closed—that is, the open mode must be `MOUNTED`. In an Oracle Real Application Clusters (Oracle RAC) environment, the PDB must be closed in all Oracle RAC instances

For *filename*, specify the full path name of the operating system file in which to store information about the PDB. The file name that you specify determines the type of information stored and how it is stored.

- If you specify a file name that ends with the extension .xml, then Oracle creates an XML file containing metadata about the PDB. You can then copy the XML file and the PDB's data files to a new location and specify the XML file name when plugging the PDB into a CDB. In this case, you must copy the PDB's data files separately.
- If you specify a file name that ends with the extension .pdb, then Oracle creates a .pdb archive file. This is a compressed file that includes an XML file containing metadata about the PDB, as well as the PDB's data files. You can then copy this single archive file to a new location and specify the archive file name when plugging the PDB into a CDB. This eliminates having to copy the PDB's data files separately. When you use a .pdb archive file when plugging in a PDB, this file is extracted when you plug in the PDB, and the PDB’s files are placed in the same directory as the .pdb archive file.

After a PDB is unplugged, it remains in the CDB with an open mode of `MOUNTED` and a status of `UNPLUGGED`. The only operation you can perform on an unplugged PDB is `DROP PLUGGABLE DATABASE`, which will remove it from the CDB. You must drop the PDB before you can plug it into the same CDB or another CDB.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information on unplugging a PDB
> - The *create_pdb_from_xml* clause of `CREATE PLUGGABLE DATABASE` for information on plugging a PDB into a CDB

pdb_unplug_encrypt

You must have the `SYSKM` privilege to execute this command.

United PDBs

- `ENCRYPT USING transport_secret` is optional.
- If TDE is in use, you must specify this clause. If TDE is not in use, the statement throws the following error `ORA-46680:master keys of the container database must be exported.`
- The wallet must be open in `ROOT` if TDE is in use.
- Keys are encrypted using the provided *`transport secret`* and exported into the `.XML` or archive file

Unplugging a PDB Into an XML Metadata File: Example

```
ALTER PLUGGABLE DATABASE CDB1_PDB2 UNPLUG INTO '/tmp/cdb1_pdb2.xml' ENCRYPT USING
transport_secret
```

Unplugging a PDB Into an Archive File: Example

```
ALTER PLUGGABLE DATABASE CDB1_PDB1_1 UNPLUG INTO '/tmp/CDB1_PDB1_1.pdb' ENCRYPT USING
transport_secret
```

For PDBs in **isolated** mode, you need not specify `ENCRYPT USING transport_secret`. This is not required because the wallet file of the PDB is copied during the creation of the pluggable database from an XML file. If you are unplugging a PDB as an archive file, the wallet file of the PDB is added to the zipped archive with the `.pdb` extension.

If the `ewallet.p12` file already exists at the destination, a backup is automatically initiated. The backup file has the following format: `ewallet_PLGDB_2017090517455564.p12`.

pdb_settings_clauses

These clauses lets you modify various settings for a PDB.

pdb_name

You can optionally use *`pdb_name`* to specify the name of the PDB whose settings you want to modify.

DEFAULT EDITION Clause

Use this clause to designate the specified edition as the default edition for the PDB. For the full semantics of this clause, refer to "DEFAULT EDITION Clause" in the `ALTER DATABASE` documentation.

SET DEFAULT TABLESPACE Clause

Use this clause to specify or change the default type of tablespaces subsequently created in the PDB. For the full semantics of this clause, refer to "SET DEFAULT TABLESPACE Clause" in the `ALTER DATABASE` documentation.

DEFAULT TABLESPACE Clause

Use this clause to establish or change the default permanent tablespace of the PDB. For the full semantics of this clause, refer to "DEFAULT TABLESPACE Clause" in the `ALTER DATABASE` documentation.

DEFAULT TEMPORARY TABLESPACE Clause

Use this clause to change the default temporary tablespace of the PDB to a new tablespace or tablespace group. For the full semantics of this clause, refer to "DEFAULT [LOCAL] TEMPORARY TABLESPACE Clause" in the `ALTER DATABASE` documentation.

RENAME GLOBAL_NAME TO Clause

Use this clause to change the global name of the PDB. The new global name must be unique within the CDB. For an Oracle Real Application Clusters (Oracle RAC) database, the PDB must be open in `READ WRITE RESTRICTED` mode on the current instance only. The PDB must be closed on all other instances. For the full semantics of this clause, refer to "RENAME GLOBAL_NAME Clause" in the `ALTER DATABASE` documentation.

> **Note:**
>
> When you change the global name of a PDB, be sure to change the `PLUGGABLE DATABASE` property for database services that are used to connect to the PDB.

set_time_zone_clause

Use this clause to modify the time zone setting for the PDB. For the full semantics of this clause, refer to *set_time_zone_clause* in the `ALTER DATABASE` documentation.

database_file_clauses

Use this clause to modify data files and temp files for the PDB. For the full semantics of this clause, refer to *database_file_clauses* in the `ALTER DATABASE` documentation.

supplemental_db_logging

Use these clauses to instruct Oracle Database to add or stop adding supplemental data into the log stream for the PDB.

- Specify the `ADD SUPPLEMENTAL LOG` clause to add supplemental data into the log stream for the PDB. In order to issue this clause, supplemental logging must have been enabled for the CDB root with the `ALTER DATABASE ... ADD SUPPLEMENTAL LOG ...` statement. The level of supplemental logging that you specify for the PDB does not need to match that of the CDB root. That is, you can specify any of the clauses `DATA`, *`supplemental_id_key_clause`*, or *`supplemental_plsql_clause`* for the PDB, regardless of which clause was specified when enabling supplemental logging for the CDB root.

- Specify the DROP SUPPLEMENTAL LOG clause to stop adding supplemental data into the log stream for the PDB.

ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION, of ALTER PLUGGABLE DATABASE enables low impact minimal supplemental logging on the PDB.

- You can only execute this DDL on a pluggable database.
- You can execute this DDL only when the *enable_goldengate_replication* parameter is TRUE, and database compatible is 19.0 or higher.
- You must enable minimal supplemental logging in CDB$ROOT to run this command.
- After you execute this DDL, minimal supplemental logging will become low impact for the pluggable database. SYS.PROP$ will be updated to indicate that low impact minimal supplemental logging is enabled at the PDB level for this pluggable database.

DROP SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION, of ALTER PLUGGABLE DATABASE disables low impact minimal supplemental logging on the PDB.

- You can only execute this DDL on a pluggable database.
- You can execute this DDL only when the *enable_goldengate_replication* parameter is TRUE, and database compatible is 19.0 or higher.
- You must enable minimal supplemental logging in CDB$ROOT to run this command.
- SYS.PROP$ will be updated to indicate that supplemental logging for subset database replication is disabled at the PDB level for this pluggable database. If supplemental logging for subset database replication is also disabled at CDB$ROOT (CDB level), then low impact minimal supplemental logging will be disabled for this pluggable database.

For the full semantics of this clause, refer to *supplemental_db_logging* in the ALTER DATABASE documentation.

pdb_storage_clause

Use this clause to modify the storage limits for a PDB.

This clause has the same semantics as the *pdb_storage_clause* in the CREATE PLUGGABLE DATABASE documentation, with the following additions:

- If you specify MAXSIZE *size_clause*, then the value you specify for *size_clause* must be greater than or equal to the combined size of the existing tablespaces belonging to the PDB. Otherwise, an error occurs.
- If you specify MAX_AUDIT_SIZE *size_clause*, then the value you specify for *size_clause* must be greater than or equal to the amount of storage used by the existing unified audit OS spillover (.bin format) files in the PDB. Otherwise, an error occurs.
- If you specify MAX_DIAG_SIZE *size_clause*, then the value you specify for *size_clause* must be greater than or equal to the amount of storage for diagnostics in the Automatic Diagnostic Repository (ADR) that is currently used by the PDB. Otherwise an error occurs.

pdb_logging_clauses

Use these clauses to set or change the logging characteristics of the PDB.

logging_clause

Use this clause to change the default logging attribute for tablespaces subsequently created within the PDB. This clause has the same semantics as the *logging_clause* in the CREATE PLUGGABLE DATABASE documentation.

pdb_force_logging_clause

Use this clause to place a PDB into, or take it out of, one of four logging modes.

Force logging mode instructs the database to log all changes in the PDB, except changes in temporary tablespaces and temporary segments. Force nologging mode instructs the database to not log any changes in the PDB.

Standby nologging instructs the database to not log operations that qualify to be done without logging. The database sends the data blocks that were created by the operation to each qualifying standby database in the Data Guard configuration, typically resulting in those standbys not having invalid blocks.

CDB-wide force logging mode takes precedence over any other setting. PDB-level force logging mode and force nologging mode take precedence over and are independent of any `LOGGING`, `NOLOGGING`, or `FORCE LOGGING` settings you specify for individual tablespaces in the PDB and any `LOGGING` or `NOLOGGING` settings you specify for individual database objects in the PDB.

- Specify `ENABLE FORCE LOGGING` to place the PDB in force logging mode. If the PDB is currently in force nologging mode, then specifying this clause results in an error. You must first specify `DISABLE FORCE NOLOGGING`.
- Specify `DISABLE FORCE LOGGING` to take the PDB out of force logging mode. If the PDB is not currently in force logging mode, then specifying this clause results in an error.
- Specify `ENABLE FORCE NOLOGGING` to place the PDB in force nologging mode. If the PDB is currently in force logging mode, then specifying this clause results in an error. You must first specify `DISABLE FORCE LOGGING`. The nonlogged operations will use classic invalidation redo, even if the CDB has a standby nologging mode set.
- Specify `DISABLE FORCE NOLOGGING` to take the PDB out of force nologging mode. If the PDB is not currently in force nologging mode, then specifying this clause results in an error.
- Specify `SET STANDBY NOLOGGING FOR LOAD PERFORMANCE` to put the PDB into standby nologging for load performance mode. In this mode the data loaded as part of the nonlogged task is sent to the qualifying standbys via a private network connection, provided that doing so will not slow down the load process. If a slow down occurs, then the data is not sent but fetched automatically from the primary as each standby encounters the invalidation redo and will be retried until the data blocks are eventually received.
- Specify `SET STANDBY NOLOGGING FOR DATA AVAILABILITY` to put the PDB into standby nologging for data availability mode. In this mode the data loaded as part of the nonlogged task is sent to the qualifying standbys either via a network connection to them, or if that fails, via block images in the redo. That is to say, in this mode the load will switch to be done in a logged fashion if the network connection or related processes prevent the sending of the data over the private network connection.

For the standby nologging modes a qualifying standby is one that is open for read, running managed recovery, and receiving redo into standby redo logs.

This clause does not change the default `LOGGING` or `NOLOGGING` mode of the PDB specified by the *logging_clause*.

pdb_refresh_mode_clause

Use this clause to change the refresh mode of a PDB. You can specify this clause only for a refreshable PDB, that is, a PDB whose current refresh mode is `MANUAL` or `EVERY` *`refresh_interval`* `MINUTES` or `HOURS`. You can switch a PDB from manual refresh to automatic refresh, or from automatic refresh to manual refresh. You can also use this clause to change

the number of minutes between automatic refreshes. You can switch a PDB from manual or automatic refresh to no refresh, but you cannot enable manual or automatic refresh for a PDB that is not refreshable. For the complete semantics of this clause, refer to the *pdb_refresh_mode_clause* in the documentation on `CREATE PLUGGABLE DATABASE`.

REFRESH

Specify this clause to perform a manual refresh of a refreshable PDB, that is, a PDB whose current refresh mode is `MANUAL` or `EVERY` *`number`* `MINUTES`. The PDB must be closed. For more information on refreshable PDBs, refer to the *pdb_refresh_mode_clause* in the documentation on `CREATE PLUGGABLE DATABASE`.

pdb_refresh_switchover_clause

Use this clause to reverse roles between a refreshable clone PDB and a primary PDB. This clause makes the refreshable clone PDB into a primary PDB, which can be opened in read write mode. The former primary PDB becomes the refreshable clone..

- This command must be executed from the primary PDB .
- `REFRESH MODE NONE` may not be specified when issuing this statement.
- The `dblink` should point to the Root of the CDB where the refreshable clone PDB currently resides.
- After this operation, the current PDB will become the refreshable clone and can only be opened in READ ONLY mode.
- The database link user must exist in the primary PDB, if the refreshable clone exists in a different CDB.

PRIORITY

You can control the order of operations on PDBs by assigning a `PRIORITY` to the PDB. The priority*`value`* should be a whole number greater than 0 and less than 4099. A value outside this range throws an error.

The following ordering rules apply to manage different kinds of PDBs:

- PDBs are processed in an ascending order of priority. A PDB with a lower priority value will be processed before a PDB with a higher priority value.
- PDBs with the same priority may be processed in any order. However, if App PDBs and the App Root have the same priority or have no priority, App PDBs will still be opened after the App Root.
- PDBs have no priority are considered to be the lowest priority.
- PDB priority for a given PDB is applicable to all RAC instances, i.e priority is NOT specific to a given RAC instance.
- Priority will not be copied from source PDB to target PDB by plug, unplug or refreshable clone.
- App PDBs cannot have a higher priority than App Root.
- App Root clones have the same priority as App Roots, and cannot be explicitly given a PDB priority.
- The priority of `CDB$ROOT` and `PDB$SEED` is determined internally by Oracle RDBMS and is not subject to PDB priority .

Use `PRIORITY NONE` to reset priority settings on the PDB.

SET CONTAINER_MAP

Use this clause to specify the `CONTAINER_MAP` database property for an application container. The current container must be the application root. The *map_object* is of the form *[schema.]table*. For *schema*, specify the schema containing *table*. If you omit *schema*, then the database assumed that the table is in your own schema. For *table*, specify a range-, list-, or hash-partitioned table.

CONTAINERS DEFAULT TARGET

Use this clause to specify the default container for DML statements in an application container. You must be connect to the application root.

- For *container_name*, specify the name of the default container. The default container can be any container in the application container, including the application root or an application PDB. You can specify only one default container.
- If you specify `NONE`, then the default container is the CDB root. This is the default.

When a DML statement is issued in the application root without specifying containers in the `WHERE` clause, the DML statement affects the default container for the application container.

CONTAINERS HOST and PORT

Use the `HOST` and `PORT` clauses if you want to create a PDB that you plan to reference from a proxy PDB. This type of PDB is called a referenced PDB.

The following statements can be executed within a PDB:

```
ALTER PLUGGABLE DATABASE CONTAINERS HOST='myhost.example.com';

ALTER PLUGGABLE DATABASE CONTAINERS PORT=1599;
```

The following statements can be executed within CDB Root, Application Root, or within a PDB :

```
ALTER PLUGGABLE DATABASE <pdbname> CONTAINERS HOST='myhost.example.com';

ALTER PLUGGABLE DATABASE <pdbname> CONTAINERS PORT=1599;
```

pdbname must meet the following criteria:

- If the statement is executed in Application Root, then *pdbname* has to match the name of Application Root or the name of one of its Application PDBs.
- If the statement is executed in CDB Root, then *pdbname* has to match the name of one of the PDBs in the CDB.
- If the statement is executed in a PDB, then *pdbname* has to match the name of the current PDB.

> **See Also:**
>
> HOST and PORT of `CREATE PLUGGABLE DATABASE` for the full semantics of `HOST` and `PORT`

pdb_datafile_clause

This clause lets you bring data files associated with a PDB online or take them offline. The PDB must be closed when you issue this clause.

- For *pdb_name*, specify the name of the PDB. If the current container is the PDB, then you can omit *pdb_name*.
- The DATAFILE clauses let you specify the data files you want to bring online or take offline. Use *filename* or *filenumber* to identify specific data files by name or by number. You can view data file names and numbers by querying the NAME and FILE# columns of the V$DATAFILE dynamic performance view. Use ALL to specify all datafiles associated with the PDB.
- Specify ONLINE to bring the data files online or OFFLINE to take the data files offline.

pdb_recovery_clauses

Use the *pdb_recovery_clauses* to back up and recover a PDB.

pdb_name

You can optionally use *pdb_name* to specify the name of the PDB you want to back up or recover.

pdb_general_recovery

This clause lets you control media recovery for the PDB or standby database or for specified tablespaces or files. The *pdb_general_recovery* clause has the same semantics as the *general_recovery* clause of ALTER DATABASE. Refer to the *general_recovery* clause of ALTER DATABASE for more information.

BACKUP Clauses

Use these clauses to move all of the data files in the PDB into or out of online backup mode (also called hot backup mode). These clauses have the same semantics in ALTER PLUGGABLE DATABASE and ALTER DATABASE. Refer to the "BACKUP Clauses" of ALTER DATABASE for more information.

RECOVERY Clauses

Use these clauses to enable or disable a PDB for recovery. The PDB must be closed—that is, the open mode must be MOUNTED.

- Specify ENABLE RECOVERY to bring all data files that belong to a PDB online and enable the PDB for recovery.
- Specify DISABLE RECOVERY to take all data files that belong to a PDB offline and disable the PDB for recovery.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for more information on the RECOVERY clauses

pdb_change_state

This clause enables you to change the state, or open mode, of a PDB. Table 11-2 lists the open modes of a PDB.

- Specify the *pdb_open* clause to change the open mode to READ WRITE, READ ONLY, or MIGRATE.
- Specify the *pdb_close* clause to change the open mode to MOUNTED.

Table 11-2 PDB Open Modes

Open Mode	Description
READ WRITE	A PDB in open read/write mode allows queries and user transactions to proceed and allows users to generate redo logs.
READ ONLY	A PDB in open read-only mode allows queries but does not allow user changes.
MIGRATE	When a PDB is in open migrate mode, you can run database upgrade scripts on the PDB.
MOUNTED	When a PDB is in mounted mode, it behaves like a non-CDB in mounted mode. It does not allow changes to any objects, and it is accessible only to database administrators. It cannot read from or write to data files. Information about the PDB is removed from memory caches. Cold backups of the PDB are possible.

You can view the open mode of a PDB by querying the OPEN_MODE column of the V$PDBS view.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for a complete description of PDB open modes

pdb_name

You can optionally use *pdb_name* to specify the name of the PDB whose open mode you want to change.

pdb_open

This clause lets you change the open mode of a PDB to READ WRITE, READ ONLY, or MIGRATE. When you specify this clause, the PDB must be in MOUNTED mode unless you specify the FORCE keyword.

If you do not specify READ WRITE or READ ONLY, then the default is READ WRITE. The exception is when the PDB belongs to a CDB that is used as a physical standby database, in which case the default is READ ONLY.

READ WRITE

Specify this clause to change the open mode to READ WRITE.

READ ONLY

Specify this clause to change the open mode to READ ONLY.

HYBRID READ ONLY

Specify this clause to open a PDB in `READ ONLY` and `READ WRITE` mode depending on the type of user that connects to it. When a local user connects to the PDB, the PDB operates in `READ ONLY` mode. The PDB operates in `READ WRITE` mode for common users.

[READ WRITE] UPGRADE

Specify this clause to change the open mode to `MIGRATE`. The `READ WRITE` keywords are optional and are provided for semantic clarity.

RESTRICTED

If you specify the optional `RESTRICTED` keyword, then the PDB is accessible only to users with the `RESTRICTED SESSION` privilege in the PDB.

If the PDB is in `READ WRITE` or `READ ONLY` mode, and you specify the `RESTRICTED` and `FORCE` keywords while changing the open mode, then all sessions connected to the PDB that do not have the `RESTRICTED SESSION` privilege in the PDB are terminated, and their transactions are rolled back.

FORCE

Specify this keyword to change the open mode of a PDB from `READ WRITE` to `READ ONLY`, or from `READ ONLY` to `READ WRITE`. The `FORCE` keyword allows users to remain connected to the PDB while the open mode is changed.

When you specify `FORCE` to change the open mode of a PDB from `READ WRITE` to `READ ONLY`, any `READ WRITE` transaction that is open when you change the open mode will not be allowed to perform any more DML operations or to `COMMIT`.

Restriction on FORCE

You cannot specify the `FORCE` keyword if the PDB is currently in `MIGRATE` mode, and you cannot specify the `FORCE` keyword to change a currently open PDB to `MIGRATE` mode.

RESETLOGS

Specify this clause to create a new PDB incarnation and open the PDB in `READ WRITE` mode after point-in-time recovery of the PDB.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information on performing point-in-time recovery of CDBs and PDBs

instances_clause

In an Oracle Real Application Clusters environment, use this clause to modify the state of the PDB in the specified Oracle RAC instances. If you omit this clause, then the state of the PDB is modified only in the current instance.

- Use *`instance_name`* to specify one or more instance names, in a comma-separated list enclosed in parenthesis. This modifies the state of the PDB only in those instances.
- Specify `ALL` to modify the state of the PDB in all instances.

- Specify `ALL EXCEPT` to modify the state of the PDB in all instances except the specified instances.

If the PDB is already open in one or more instances, then you can open it in additional instances, but it must be opened in the same mode as in the instances in which it is already open.

services_clause

- Use *`service_name`* to specify one or more services, in a comma-separated list enclosed in parenthesis.
- Specify `ALL` to specify all services on the PDB.
- Specify `ALL EXCEPT` to specify all services except the services specifed.

pdb_close

This clause lets you change the open mode of a PDB to `MOUNTED`. When you specify this clause, the PDB must be in `READ WRITE`, `READ ONLY`, or `MIGRATE` mode. This clause is the PDB equivalent of the SQL*Plus `SHUTDOWN` command.

IMMEDIATE

If you specify the optional `IMMEDIATE` keyword, then this clause is the PDB equivalent of the SQL*Plus `SHUTDOWN` command with the immediate mode. Otherwise, the PDB is shut down with the normal mode.

> **See Also:**
>
> *SQL*Plus User's Guide and Reference* for more information on the SQL*Plus `SHUTDOWN` command

ABORT

Specify `ABORT` to forcibly shut down the PDB.

instances_clause

In an Oracle Real Application Clusters environment, use this clause to modify the state of the PDB in the specified Oracle RAC instances. You can close a PDB in some instances and leave it open in others. Refer to the instances_clause for the full semantics of this clause.

relocate_clause

In an Oracle Real Application Clusters environment, use this clause to instruct the database to reopen the PDB on a different Oracle RAC instance.

- Specify `RELOCATE` to reopen the PDB on a different instance that is selected by Oracle Database.
- Specify `RELOCATE TO '`*`instance_name`*`'` to reopen the PDB in the specified instance.
- Specify `NORELOCATE` to close the PDB in the current instance. This is the default.

pdb_save_or_discard_state

Use this clause to instruct the database to save or discard the open mode of the PDB when the CDB restarts.

- If you specify `SAVE`, then the PDB's open mode after the CDB restarts will be identical to its open mode just before the CDB restarted.
- If you specify `DISCARD`, then the PDB's open mode after the CDB restarts will be `MOUNTED`. This is the default.

instances_clause

In an Oracle Real Application Clusters environment, use this clause to instruct the database to save or discard the open mode of the PDB in the specified Oracle RAC instances. If you omit this clause, then the database applies the `SAVE` or `DISCARD` setting only to the PDB in the current instance.

- Use *`instance_name`* to specify one or more instance names, in a comma-separated list enclosed in parenthesis. This applies the `SAVE` or `DISCARD` setting to the PDB only in those instances.
- Specify `ALL` to apply the `SAVE` or `DISCARD` setting to the PDB in all instances.
- Specify `ALL EXCEPT` to apply the `SAVE` or `DISCARD` setting to the PDB in all instances except the specified instances.

pdb_change_state_from_root

This clause enables you to modify the state of one or more PDBs.

- Specify the *`pdb_name`* for one or more PDBs whose state you want to modify.
- Specify `ALL` to modify the state of all PDBs in the CDB.
- Specify `ALL EXCEPT` to modify the state of all PDBs in the CDB except those specified by using *`pdb_name`*.

If a PDB is already in the specified state, then the PDB's state is unchanged and no error is returned. If the state of a PDB cannot be changed, then an error occurs only for that PDB.

application_clauses

Use the `APPLICATION` clauses to:

- Install, patch, upgrade, and uninstall applications
- Register application versions and patch numbers
- Sync operations on applications

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information on administering application containers

Specifying Application Names

Most of the *`application_clauses`* require you to specify an application name. The maximum length of an application name is 30 bytes. The name must satisfy the requirements listed in "Database Object Naming Rules ". The application name must be unique within an application container.

Specifying Application Versions

Several of the *application_clauses* require you to specify an application version. The application version can be up to 30 bytes in length and can contain alphanumeric characters, punctuations marks, and spaces. The application version is case-sensitive and must be enclosed in single quotation marks.

Specifying Comments

Several of the *application_clauses* allow you to specify a comment to associate with an application install, patch, or upgrade operation. For *comment*, enter a character string enclosed in single quotation marks.

INSTALL Clauses

Use the INSTALL clauses when installing an application in an application container. The current container must be the application root, not an application PDB.

- Specify the BEGIN INSTALL clause before you start installing the application.
  - Use *app_name* to assign a name to the application.
  - Use *app_version* to assign a version to the application.
  - The optional COMMENT clause allows you to enter a comment to be associated with the application version created by this installation.
- Specify the END INSTALL clause after you have finished installing the application.
  - You must specify the same *app_name* that you specified for the corresponding BEGIN INSTALL clause.
  - You need not specify *app_version*, but if you do, then you must specify the same version that you specified for the corresponding BEGIN INSTALL clause.

PATCH Clauses

Use the PATCH clauses when patching an application in an application container. The current container must be the application root, not an application PDB.

- Specify the BEGIN PATCH clause before you start patching the application.
  - For *app_name*, specify the name of the application you want to patch.
  - For *number*, specify the patch number.
  - The optional MINIMUM VERSION clause allows you to specify the minimum version at which the application must be before the patch can be applied. For *app_version*, specify the minimum application version. If the current application version is lower than the minimum application version, then an error occurs. If you omit this clause, then the minimum version is the current application version.
  - The optional COMMENT clause allows you to enter a comment to be associated with the patch.
- Specify the END PATCH clause after you finish patching the application.
  - You must specify the same *app_name* that you specified for the corresponding BEGIN PATCH clause.
  - You need not specify *number*, but if you do, then you must specify the same value that you specified for the corresponding BEGIN PATCH clause.

UPGRADE Clauses

Use the UPGRADE clauses when upgrading an application in an application container. The current container must be the application root, not an application PDB.

If the application root is using TDE, then you must configure an external store before upgrading the application.

- Specify the `BEGIN UPGRADE` clause before you start upgrading the application.
  - For *app_name*, specify the name of the application you want to upgrade.
  - For *start_app_version*, specify the version from which you are upgrading the application. If this version does not match the current application version, then an error occurs.
  - For *end_app_version*, specify the version to which you are upgrading the application.
  - The optional `COMMENT` clause allows you to enter a comment to be associated with the upgrade.
- Specify the `END UPGRADE` clause after you finish upgrading the application.
  - You must specify the same *app_name* that you specified for the corresponding `BEGIN UPGRADE` clause.
  - You need not specify `TO` *end_app_version*, but if you do, then you must specify the same version that you specified for the corresponding `BEGIN UPGRADE` clause.

UNINSTALL Clauses

Use the `UNINSTALL` clauses when uninstalling an application from an application container. The current container must be the application root, not an application PDB.

- Specify the `BEGIN UNINSTALL` clause before you start uninstalling the application.
  - For *app_name*, specify the name of the application you want to uninstall.
- Specify the `END UNINSTALL` clause after you have finished uninstalling the application.
  - You must specify the same *app_name* that you specified for the corresponding `BEGIN UNINSTALL` clause.

SET PATCH

Use the `SET PATCH` clause to register the patch number of an application that is already installed in an application container. This clause allows you to assign a patch number to an application that was not patched using the `PATCH` clauses. This is useful if the application was migrated from a PDB in an earlier Oracle Database release, when the `PATCH` clauses were not available. The current container can be the application root or an application PDB.

- For *app_name*, specify the name of an existing application.
- Use *number* to assign a patch number to the existing application.

SET VERSION

Use the `SET VERSION` clause to register the version of an application that is already installed in an application container. This clause allows you to assign a name and a version to an application that was not installed using the `INSTALL` clauses. This is useful if the application was migrated from a PDB in an earlier Oracle Database release, when the `INSTALL` clauses were not available. The current container can be the application root or an application PDB.

- Use *app_name* to assign a name to the existing application.
- Use *app_version* to assign a version to the existing application.

SET COMPATIBILITY VERSION

Use the `SET COMPATIBILITY VERSION` clause to set the compatibility version for an application.

The compatibility version of an application is the earliest version of the application possible for the application PDBs that belong to the application container. The current container must be the application root, not an application PDB.

> **Note:**
>
> You cannot plug in an application PDB that uses an application version earlier than the compatibility setting of the application container.

- Use *app_name* to specify the name of the application.
- Use *app_version* to specify the compatibility version for the application.
- If you specify CURRENT, then the compatibility version is set to the version of the application in the application root.

The compatibility version is enforced when the compatibility version is set and when an application PDB is created. If there are application root clones that resulted from application upgrades, then all application root clones that correspond to versions earlier than the compatibility version are implicitly dropped.

SYNC TO

You can synchronize an application to a particular version or a patch number. There are two variations:

1. SYNC TO *version_string*
2. SYNC TO PATCH *patch_number*

Example

Assume that you perform the following operations on application salesapp :

1. Install version 1.0
2. Patch 101
3. Upgrade to version 2.0
4. Patch 102
5. Upgrade to 3.0

ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC TO 2.0 replays all statements up to and including ' Upgrade to version 2.0'.

ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC TO PATCH 102 replays all statements up to and including ' Patch 102'.

Restrictions on SYNC TO

You can use SYNC TO only with an individual application.

You cannot use SYNC TO with the ALL SYNC clause.

You cannot use SYNC TO with the SYNC clause, in the case when you are synchronizing multiple applications in a single statement.

SYNC

Use the `SYNC` clause to synchronize an application in an application PDB to the version and patch level of the same application in the application root. The current container must be an application PDB.

app_name specifies the name of an application that exists in the application root. The application may or may not exist in the application PDB.

Starting with Oracle Database Release 21c you can synchronize mutiple applications in one statement with `SYNC` . This is necessary to preserve functional correctness for applications that depend on one another.

Example

```
ALTER PLUGGABLE DATABASE APPLICATION hrapp payrollapp employeesapp SYNC
```

Restrictions on Synchronizing Multiple Applications Using SYNC

- You cannot use the `SYNC TO` *version_string* clause while synchronizing multiple applications with `SYNC`.
- You cannot use the `SYNC TO PATCH` *patch_number* clause while synchronizing multiple applications with `SYNC`.

ALL SYNC

Use the `ALL SYNC` clause to sync all applications in an application PDB with all applications in the application root. This clause is useful, if you have recently added the application PDB to the CDB and would like to sync its applications with the CDB. The current container must be an application PDB.

With Release 21c you can use `EXCEPT` to exclude applications from `ALL SYNC`.

Example

```
ALTER PLUGGABLE DATABASE APPLICATION ALL EXCEPT hrapp payrollapp SYNC
```

Restrictions on Excluding Multiple Applications Using ALL SYNC

- You cannot use the `SYNC TO` *version_string* clause while excluding multiple applications with `ALL EXCEPT SYNC`.
- You cannot use the `SYNC TO PATCH` *patch_number* clause while excluding multiple applications with `ALL EXCEPT SYNC`.

snapshot_clauses

The snapshot clauses allow you to create and manage snapshots of the PDB for the lifetime of the PDB.

pdb_snapshot_clause

Specify this clause to enable the creation of PDB snapshots. You can also specify this clause in the `CREATE PLUGGABLE DATABASE` statement.

- `NONE` is the default and means that no snapshots of the PDB can be created.
- `MANUAL` means that a snapshot of the PDB can be created only manually.

- If `snapshot_interval` is specified, PDB snapshots will be created automatically at the interval specified. In addition, a user will also be able to create PDB snapshots manually.
- If expressed in minutes, the `snapshot_interval` must be less than 3000.
- If expressed in hours, the `snapshot_interval` must be less than 2000.

materialize_clause

Use this clause to convert a snapshot PDB into a full PDB clone. You can delete and purge a PDB snapshot using the clause in this way.

- This clause can only be specified for PDBs created as a snapshot.
- All blocks in all datafiles belonging to the PDB will be copied.

create_snapshot_clause

Use this clause to manually create a PDB snapshot after connecting to the PDB.

- This statement may be issued even if the PDB was set to have PDB snapshots created automatically.
- If a PDB Snapshot with the specified name already exists, an error will be reported.
- A PDB Snapshot with specified name will be created.

drop_snapshot_clause

Use this clause to manually drop a PDB snapshot after connecting to the PDB.

- If this snapshot is being used by some PDB, an error will be reported.

set_max_pdb_snapshots

Use this clause to increase or decrease the maximum number of snapshots for a given PDB. You must first connect to the PDB.

- If the PDB is not open in read/write mode when issuing the statement, an error is raised.
- You can drop all PDB snapshots by setting the the max number to 0.
- The maximum number of snapshots that you can set per PDB is 8.

prepare_clause

- Use this clause to prepare mirror copies of the database. You must provide a *`mirror_name`* to identify the filegroup that is created. The created filegroup contains all the prepared files.
- Specify the number of copies to be prepared by the `REDUNDANCY` options: `EXTERNAL`, `NORMAL`, or `HIGH`.
- If you do not specify the redundancy of the mirror, the redundancy of the source database is used.
- Use the `FOR DATABASE` clause to specify the new name of the CDB. This name should be unique. It will be used in the *`create_pdb_from_mirror_copy`* clause of the `CREATE PLUGGABLE DATABASE` statement.

Prepare a Pluggable Database By Name: Example

If you specify the name (*`pdb_name`*) of the pluggable database, it checks if *`pdb_name`* matches with the current PDB. If it matches, it runs.

```
ALTER PLUGGABLE DATABASE pdb_name PREPARE MIRROR COPY mirror_name WITH HIGH REDUNDANCY
```

Prepare a Pluggable Database Without a Name: Example

If you do not specify the name (*pdb_name*) of the pluggable database, the statement runs on the current PDB.

```
ALTER PLUGGABLE DATABASE PREPARE MIRROR COPY mirror_name WITH HIGH REDUNDANCY
```

drop_mirror_copy

Use this clause to discard mirror copies of data and metadata created by the prepare statement. You must specify the same mirror name that you used for the prepare operation.

You cannot use this clause to drop a database that has already been split by the `CREATE DATABASE` or `CREATE PLUGGABLE DATABASE` statement.

lost_write_protection

Turn on Lost Write for a Pluggable Database : Example

```
ALTER PLUGGABLE DATABASE
  ENABLE LOST WRITE PROTECTION
```

Turn off Lost Write for a Pluggable Database : Example

```
ALTER PLUGGABLE DATABASE
  DISABLE LOST WRITE PROTECTION
```

Note that disabling lost write for the database does not deallocate the lost write storage. You must use the `DROP TABLESPACE` statement to deallocate lost write storage.

pdb_managed_recovery

Specify this clause to recover a PDB in instances where the PDB is within a physical standby CDB.

ENABLE | DISABLE BACKUP

PDB backup is enabled by default. To exclude the PDB from the backup, you can specify disable.

Examples

Unplugging a PDB from a CDB: Example

The following statement unplugs PDB `pdb1` and stores metadata for the PDB into XML file `/oracle/data/pdb1.xml`:

```
ALTER PLUGGABLE DATABASE pdb1
  UNPLUG INTO '/oracle/data/pdb1.xml';
```

Modifying the Settings of a PDB: Example

The following statement changes the limit for the amount of storage used by all tablespaces in PDB `pdb2` to 500M:

```
ALTER PLUGGABLE DATABASE pdb2
  STORAGE (MAXSIZE 500M);
```

Taking the Data Files of a PDB Offline: Example

The following statement takes the data files associated with PDB `pdb3` offline:

```
ALTER PLUGGABLE DATABASE pdb3
  DATAFILE ALL OFFLINE;
```

Changing the State of a PDB: Examples

Assume that PDB pdb4 is closed—that is, its open mode is MOUNTED. The following statement opens pdb4 with open mode READ ONLY:

```
ALTER PLUGGABLE DATABASE pdb4
  OPEN READ ONLY;
```

The following statement uses the FORCE keyword to change the open mode of pdb4 from READ ONLY to READ WRITE:

```
ALTER PLUGGABLE DATABASE pdb4
  OPEN READ WRITE FORCE;
```

The following statement closes PDB pdb4:

```
ALTER PLUGGABLE DATABASE pdb4
  CLOSE;
```

The following statement opens PDB pdb4 with open mode READ ONLY. Because the RESTRICTED keyword is specified, the PDB is accessible only to users with the RESTRICTED SESSION privilege in the PDB.

```
ALTER PLUGGABLE DATABASE pdb4
  OPEN READ ONLY RESTRICTED;
```

Assume that PDB pdb5 is closed—that is, its open mode is MOUNTED. In an Oracle Real Application Clusters environment, the following statement opens PDB pdb5 with open mode READ WRITE in instances ORCLDB_1 and ORCLDB_2:

```
ALTER PLUGGABLE DATABASE pdb5
  OPEN READ WRITE INSTANCES = ('ORCLDB_1', 'ORCLDB_2');
```

In an Oracle Real Application Clusters environment, the following statement closes PDB pdb6 in the current instance and instructs the database to reopen pdb6 in instance ORCLDB_3:

```
ALTER PLUGGABLE DATABASE pdb6
  CLOSE RELOCATE TO 'ORCLDB_3';
```

Changing the State of All PDBs in a CDB: Example

Assume that the current container is the root. The following statement opens all PDBs in the CDB with open mode READ ONLY:

```
ALTER PLUGGABLE DATABASE ALL
  OPEN READ ONLY;
```

ALTER PMEM FILESTORE

Purpose

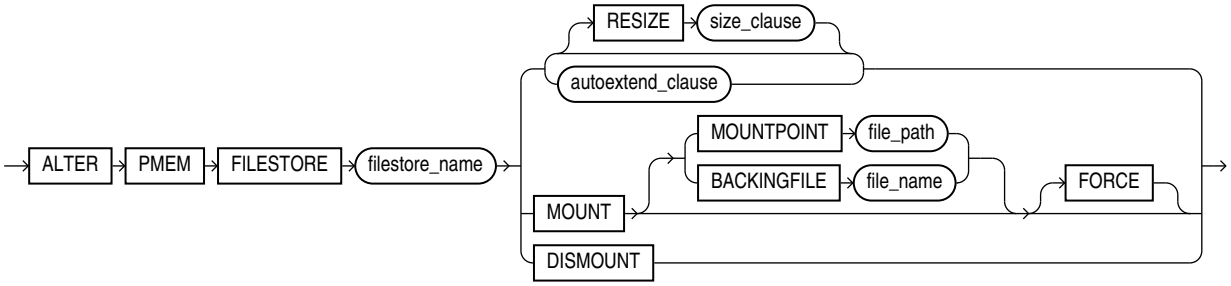
Use this command to change the attributes of a PMEM file store.

Prerequisites

You cannot change the block size of a PMEM file store.

Syntax

alter_pmem_filestore



Semantics

MOUNT

Use this command to mount a PMEM file store. If you have already specified the mount point and backing file paths in the `init.ora` file you can issue the command like this:

```
ALTER PMEM FILESTORE 'filestore_name' MOUNT
```

You can also specify the mount point and backing file paths in the command line. In this case, you must ensure that there is no mismatch between the values in the `init.ora` file and the values you specify in the command line. The command fails when a mismatch occurs, unless you specify `FORCE` to override the values in the `init.ora` file. The paths on the command line become the new paths for the PMEM file store.

If you use a `spfile`, then the parameters are automatically updated with the new paths specified on the command line.

Use the mount PMEM file store command in cases when the PMEM file store was not already automatically mounted during database startup.

Specify the mount point path or the backing file path on the command line when:

- You have not specified either the mount point path or the backing file path in the `init.ora` file
- You want to specify new values for either the mount point path or the backing file path

Before you can change the mount point and the backing file, you must first dismount the file store.

DISMOUNT

Use this command to dismount a PMEM file store. You must ensure that the database instance is in `NOMOUNT` mode.

Examples

Example 1: Resize File Store Named cloud_db_1

```
ALTER PMEM FILESTORE cloud_db_1 RESIZE 5T
```

Example 2: Mount File Store Named cloud_db_1

```
ALTER  PMEM FILESTORE cloud_db_1 MOUNT  MOUNTPOINT '/corp/db/cloud_db_1'
    BACKINGFILE '/var/pmem/foo_1'
```

Example 3: Dismount File Store Named cloud_db_1

```
ALTER PMEM FILESTORE cloud_db_1 DISMOUNT
```

ALTER PROCEDURE

Purpose

Packages are defined using PL/SQL. Therefore, this section provides some general information but refers to *Oracle Database PL/SQL Language Reference* for details of syntax and semantics.

Use the `ALTER PROCEDURE` statement to explicitly recompile a standalone stored procedure. Explicit recompilation eliminates the need for implicit run-time recompilation and prevents associated run-time compilation errors and performance overhead.

To recompile a procedure that is part of a package, recompile the entire package using the `ALTER PACKAGE` statement (see ALTER PACKAGE).

> **Note:**
>
> This statement does not change the declaration or definition of an existing procedure. To redeclare or redefine a procedure, use the `CREATE PROCEDURE` statement with the `OR REPLACE` clause (see CREATE PROCEDURE).

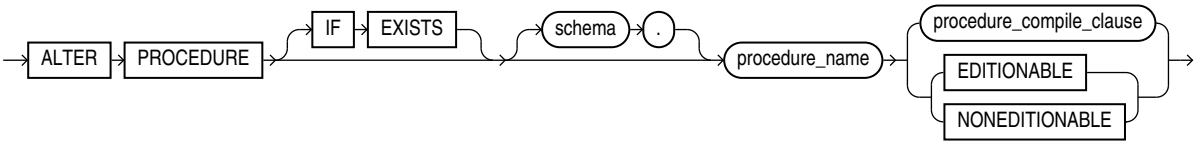
The `ALTER PROCEDURE` statement is quite similar to the `ALTER FUNCTION` statement. Refer to ALTER FUNCTION for more information.

Prerequisites

The procedure must be in your own schema or you must have `ALTER ANY PROCEDURE` system privilege.

Syntax

***alter_procedure*::=**



(*procedure_compile_clause*: See *Oracle Database PL/SQL Language Reference* for the syntax of this clause.)

Semantics

IF EXISTS

Specify `IF EXISTS` to alter an existing table.

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

schema

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

procedure_name

Specify the name of the procedure to be recompiled.

procedure_compile_clause

See *Oracle Database PL/SQL Language Reference* for the syntax and semantics of this clause and for complete information on creating and compiling procedures.

EDITIONABLE | NONEDITIONABLE

Use these clauses to specify whether the procedure becomes an editioned or noneditioned object if editioning is later enabled for the schema object type `PROCEDURE` in *`schema`*. The default is `EDITIONABLE`. For information about altering editioned and noneditioned objects, see *Oracle Database Development Guide*.

ALTER PROFILE

Purpose

Use the `ALTER PROFILE` statement to add, modify, or remove a resource limit or password management parameter in a profile.

Changes made to a profile with an `ALTER PROFILE` statement affect users only in their subsequent sessions, not in their current sessions.

> **See Also:**
>
> CREATE PROFILE for information on creating a profile

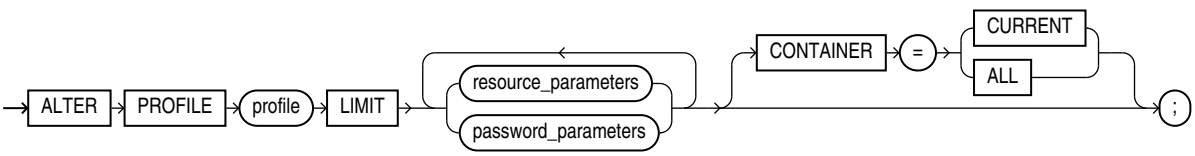
Prerequisites

You must have the `ALTER PROFILE` system privilege.

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. To specify `CONTAINER = CURRENT`, the current container must be a pluggable database (PDB).

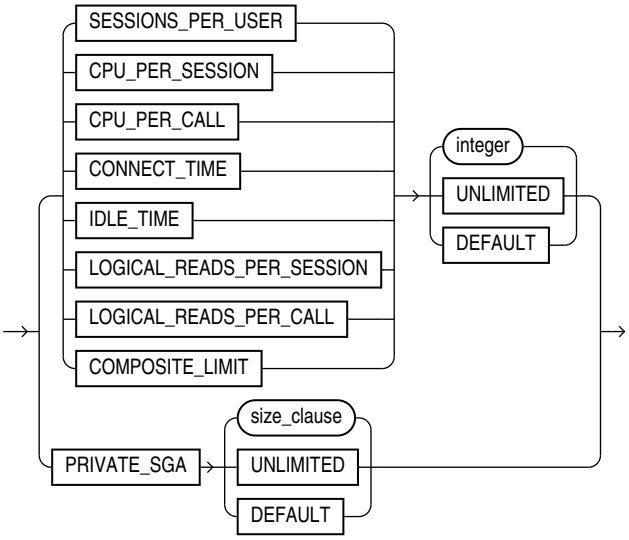
Syntax

***alter_profile*::=**



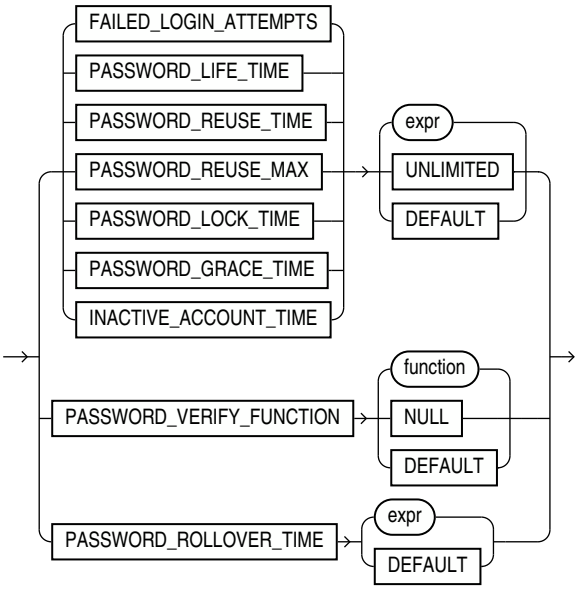
(*resource_parameters*::=, *password_parameters*::=)

***resource_parameters*::=**



(*size_clause*::=)

***password_parameters*::=**



Semantics

The keywords, parameters, and clauses common to `ALTER PROFILE` and `CREATE PROFILE` have the same meaning. For full semantics of these keywords, parameters, and clauses refer to CREATE PROFILE .

Only common users who have been commonly granted the `ALTER PROFILE` system privilege can alter or drop the mandatory profile, and only from the CDB root.

You cannot remove a limit from the `DEFAULT` profile.

Examples

Making a Password Unavailable: Example

The following statement makes the password of the `new_profile` profile (created in "Creating a Profile: Example") unavailable for reuse for 90 days:

```
ALTER PROFILE new_profile
   LIMIT PASSWORD_REUSE_TIME 90
   PASSWORD_REUSE_MAX UNLIMITED;
```

Setting Default Password Values: Example

The following statement defaults the `PASSWORD_REUSE_TIME` value of the `app_user` profile (created in "Setting Profile Resource Limits: Example") to its defined value in the `DEFAULT` profile:

```
ALTER PROFILE app_user
   LIMIT PASSWORD_REUSE_TIME DEFAULT
   PASSWORD_REUSE_MAX UNLIMITED;
```

Limiting Login Attempts and Password Lock Time: Example

The following statement alters profile `app_user` with `FAILED_LOGIN_ATTEMPTS` set to 5 and `PASSWORD_LOCK_TIME` set to 1:

```
ALTER PROFILE app_user LIMIT
   FAILED_LOGIN_ATTEMPTS 5
   PASSWORD_LOCK_TIME 1;
```

This statement causes any user account to which the `app_user` profile is assigned to become locked for one day after five consecutive unsuccessful login attempts.

Changing Password Lifetime and Grace Period: Example

The following statement modifies the profile `app_user2 PASSWORD_LIFE_TIME` to 90 days and `PASSWORD_GRACE_TIME` to 5 days:

```
ALTER PROFILE app_user2 LIMIT
   PASSWORD_LIFE_TIME 90
   PASSWORD_GRACE_TIME 5;
```

Limiting Account Inactivity: Example

The following statement modifies the profile `app_user2 INACTIVE_ACCOUNT_TIME` to 30 consecutive days:

```
ALTER PROFILE app_user2 LIMIT
  INACTIVE_ACCOUNT_TIME 30;
```

If the account has already been inactive for a certain number of days, then those days count toward the new 30 day limit.

Limiting Concurrent Sessions: Example

This statement defines a new limit of 5 concurrent sessions for the `app_user` profile:

```
ALTER PROFILE app_user LIMIT SESSIONS_PER_USER 5;
```

If the `app_user` profile does not currently define a limit for `SESSIONS_PER_USER`, then the preceding statement adds the limit of 5 to the profile. If the profile already defines a limit, then the preceding statement redefines it to 5. Any user assigned the `app_user` profile is subsequently limited to 5 concurrent sessions.

Removing Profile Limits: Example

This statement removes the `IDLE_TIME` limit from the `app_user` profile:

```
ALTER PROFILE app_user LIMIT IDLE_TIME DEFAULT;
```

Any user assigned the `app_user` profile is subject in their subsequent sessions to the `IDLE_TIME` limit defined in the `DEFAULT` profile.

Limiting Profile Idle Time: Example

This statement defines a limit of 2 minutes of idle time for the `DEFAULT` profile:

```
ALTER PROFILE default LIMIT IDLE_TIME  2;
```

This `IDLE_TIME` limit applies to these users:

- Users who are not explicitly assigned any profile
- Users who are explicitly assigned a profile that does not define an `IDLE_TIME` limit

This statement defines unlimited idle time for the `app_user2` profile:

```
ALTER PROFILE app_user2 LIMIT IDLE_TIME UNLIMITED;
```

Any user assigned the `app_user2` profile is subsequently permitted unlimited idle time.

Enable Gradual Password Rollover: Example

This statement sets the password rollover time to 2 days in the profile `usr_prof`:

```
ALTER PROFILE usr_prof LIMIT PASSWORD_ROLLOVER_TIME 2 ;
```

ALTER PROPERTY GRAPH

Purpose

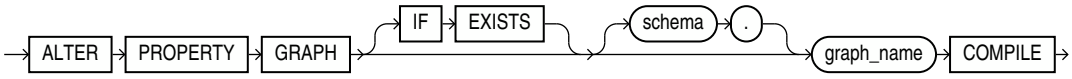
Changes to the underlying objects of the property graph may cause the property graph to be in an invalid state.You can revalidate a graph with `ALTER PROPERTY GRAPH COMPILE`.

Prerequistes

To alter a property graph in any schema except `SYS` and `AUDSYS`, you must have the `ALTER ANY PROPERTY GRAPH` privilege.

Syntax

***alter_property_graph*::=**



Semantics

IF EXISTS

Specify `IF EXISTS` to alter an existing property graph.

If you specify `IF NOT EXISTS` with `ALTER`, the command fails with the error message: `Incorrect IF EXISTS clause for ALTER/DROP statement.`

COMPILE

Use `ALTER PROPERTY GRAPH COMPILE` to revalidate a graph that reports an invalid state even though it is actually valid. This happens because the dependencies of the graph to its underlying objects may be too coarse. In such cases, it may be enough to use `ALTER PROPERTY GRAPH COMPILE` to revalidate the property graph.

Example: How a Valid Graph May Falsely Report an Invalid State

```
SQL> create table tbl1(c1 number primary key, c2 number, c3 number, c4 as (c2/
c3), c5 as (1 / (c2+c3)));

Table created.

SQL> create property graph g vertex tables(tbl1 properties(c2, c3, c5));

Property graph created.

SQL> select object_name, object_type, status from user_objects where
object_type in ('PROPERTY GRAPH', 'TABLE');

OBJECT_NAME OBJECT_TYPE STATUS
------------------- ---------------------- -------
G PROPERTY GRAPH VALID
TBL1 TABLE VALID

SQL> alter table tbl1 drop column c4;

Table altered.

SQL> select object_name, object_type, status from user_objects where
object_type in ('PROPERTY GRAPH', 'TABLE');

OBJECT_NAME OBJECT_TYPE STATUS
------------------- ---------------------- -------
G PROPERTY GRAPH INVALID
TBL1 TABLE VALID
```

```
SQL> alter property graph g compile;

Property graph altered.

SQL> select object_name, object_type, status from user_objects where
object_type in ('PROPERTY GRAPH', 'TABLE');

OBJECT_NAME OBJECT_TYPE STATUS
------------------- ---------------------- -------
G PROPERTY GRAPH VALID
TBL1 TABLE VALID

SQL>
```

If `ALTER PROPERTY GRAPH COMPILE` fails to revalidate the graph, then the graph enters the error state. You must then redefine the graph with `CREATE OR REPLACE PROPERTY GRAPH` .

ALTER RESOURCE COST

Purpose

Use the `ALTER RESOURCE COST` statement to specify or change the formula by which Oracle Database calculates the total resource cost used in a session.

Although Oracle Database monitors the use of other resources, only the four resources shown in the syntax can contribute to the total resource cost for a session.

This statement lets you apply weights to the four resources. Oracle Database then applies the weights to the value of these resources that were specified for a profile to establish a formula for calculating total resource cost. You can limit this cost for a session with the `COMPOSITE_LIMIT` parameter of the `CREATE PROFILE` statement. If the resource cost of a session exceeds the limit, then Oracle Database aborts the session and returns an error. If you use the `ALTER RESOURCE COST` statement to change the weight assigned to each resource, then Oracle Database uses these new weights to calculate the total resource cost for all current and subsequent sessions.

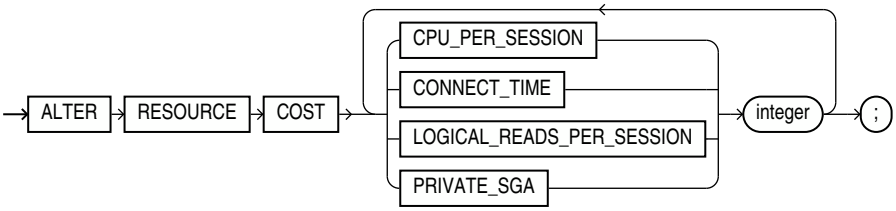
> **See Also:**
>
> CREATE PROFILE for information on all resources and on establishing resource limits

Prerequisites

You must have the `ALTER RESOURCE COST` system privilege.

Syntax

***alter_resource_cost*::=**



Semantics

Oracle Database calculates the total resource cost by first multiplying the amount of each resource used in the session by the weight of the resource, and then summing the products for all four resources. For any session, this cost is limited by the value of the `COMPOSITE_LIMIT` parameter in the user's profile. Both the products and the total cost are expressed in units called **service units**.

CPU_PER_SESSION

Use this keyword to apply a weight to the `CPU_PER_SESSION` resource.

CONNECT_TIME

Use this keyword to apply a weight to the `CONNECT_TIME` resource.

LOGICAL_READS_PER_SESSION

Use this clause to apply a weight to the `LOGICAL_READS_PER_SESSION` resource. Logical reads include blocks read from both memory and disk.

PRIVATE_SGA

Use this clause to apply a weight to the `PRIVATE_SGA` resource. This limit applies only if you are using shared server architecture and allocating private space in the SGA for your session.

integer

Specify the weight of each resource. The weight that you assign to each resource determines how much the use of that resource contributes to the total resource cost. If you do not assign a weight to a resource, then the weight defaults to 0, and use of the resource subsequently does not contribute to the cost. The weights you assign apply to all subsequent sessions in the database.

Examples

Altering Resource Costs: Examples

The following statement assigns weights to the resources `CPU_PER_SESSION` and `CONNECT_TIME`:

```
ALTER RESOURCE COST
   CPU_PER_SESSION 100
   CONNECT_TIME      1;
```

The weights establish this cost formula for a session:

```
cost = (100 * CPU_PER_SESSION) + (1 * CONNECT_TIME)
```

In this example, the values of `CPU_PER_SESSION` and `CONNECT_TIME` are either values in the `DEFAULT` profile or in the profile of the user of the session.

Because the preceding statement assigns no weight to the resources `LOGICAL_READS_PER_SESSION` and `PRIVATE_SGA`, these resources do not appear in the formula.

If a user is assigned a profile with a `COMPOSITE_LIMIT` value of 500, then a session exceeds this limit whenever `cost` exceeds 500. For example, a session using 0.04 seconds of CPU time and 101 minutes of elapsed time exceeds the limit. A session using 0.0301 seconds of CPU time and 200 minutes of elapsed time also exceeds the limit.

You can subsequently change the weights with another `ALTER RESOURCE` statement:

```
ALTER RESOURCE COST
   LOGICAL_READS_PER_SESSION 2
   CONNECT_TIME 0;
```

These new weights establish a new cost formula:

```
cost = (100 * CPU_PER_SESSION) + (2 * LOGICAL_READ_PER_SECOND)
```

where the values of `CPU_PER_SESSION` and `LOGICAL_READS_PER_SECOND` are either the values in the `DEFAULT` profile or in the profile of the user of this session.

This `ALTER RESOURCE COST` statement changes the formula in these ways:

- The statement omits a weight for the `CPU_PER_SESSION` resource. That resource was already assigned a weight, so the resource remains in the formula with its original weight.
- The statement assigns a weight to the `LOGICAL_READS_PER_SESSION` resource, so this resource now appears in the formula.
- The statement assigns a weight of 0 to the `CONNECT_TIME` resource, so this resource no longer appears in the formula.
- The statement omits a weight for the `PRIVATE_SGA` resource. That resource was not already assigned a weight, so the resource still does not appear in the formula.

ALTER ROLE

Purpose

Use the `ALTER ROLE` statement to change the authorization needed to enable a role.

> **See Also:**
>
> - CREATE ROLE for information on creating a role
> - SET ROLE for information on enabling or disabling a role for your session

Prerequisites

You must either have been granted the role with the `ADMIN OPTION` or have `ALTER ANY ROLE` system privilege.

Before you alter a role to `IDENTIFIED GLOBALLY`, you must:

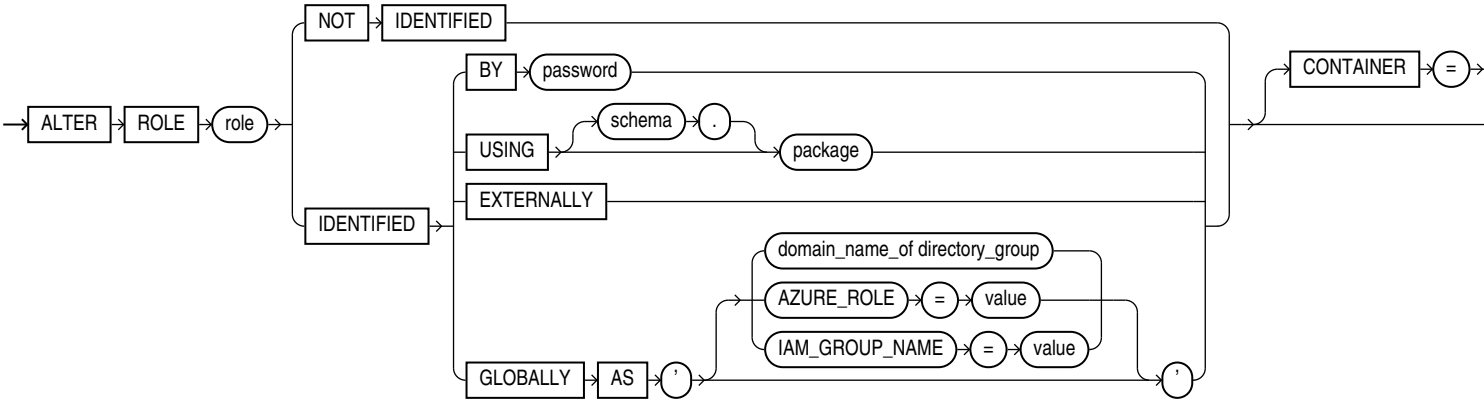
- Revoke all grants of roles identified externally to the role and
- Revoke the grant of the role from all users, roles, and `PUBLIC`.

The one exception to this rule is that you should not revoke the role from the user who is currently altering the role.

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. To specify `CONTAINER = CURRENT`, the current container must be a pluggable database (PDB).

Syntax

***alter_role*::=**



Semantics

The keywords, parameters, and clauses in the `ALTER ROLE` statement all have the same meaning as in the `CREATE ROLE` statement.

Specify `GLOBALLY` with `AS` to map a directory group to a global role when using centrally managed users. The directory group is identified by its domain name.

Restriction on Altering a Role

You cannot alter a `NOT IDENTIFIED` role to any of the `IDENTIFIED` types if it is granted to another role.

Notes on Altering a Role:

- User sessions in which the role is already enabled are not affected.
- If you change a role identified by password to an application role (with the `USING` *`package`* clause), then password information associated with the role is lost. Oracle Database will use the new authentication mechanism the next time the role is to be enabled.
- If you have the `ALTER ANY ROLE` system privilege and you change a role that is `IDENTIFIED GLOBALLY` to `IDENTIFIED BY` *`password`*, `IDENTIFIED EXTERNALLY`, or `NOT IDENTIFIED`, then Oracle Database grants you the altered role with the `ADMIN OPTION`, as it would have if you had created the role identified nonglobally.

For more information, refer to CREATE ROLE and to the examples that follow.

Examples

Changing Role Identification: Example

The following statement changes the role `warehouse_user` (created in "Creating a Role: Example") to `NOT IDENTIFIED`:

```
ALTER ROLE warehouse_user NOT IDENTIFIED;
```

Changing a Role Password: Example

This statement changes the password on the `dw_manager` role (created in "Creating a Role: Example") to `data`:

```
ALTER ROLE dw_manager
   IDENTIFIED BY data;
```

Users granted the `dw_manager` role must subsequently use the new password `data` to enable the role.

Application Roles: Example

The following example changes the `dw_manager` role to an application role using the `hr.admin` package:

```
ALTER ROLE dw_manager IDENTIFIED USING hr.admin;
```

ALTER ROLLBACK SEGMENT

> **Note:**
>
> Oracle strongly recommends that you run your database in automatic undo management mode instead of using rollback segments. Do not use rollback segments unless you must do so for compatibility with earlier versions of Oracle Database. Refer to *Oracle Database Administrator's Guide* for information on automatic undo management.

Purpose

Use the `ALTER ROLLBACK SEGMENT` statement to bring a rollback segment online or offline, change its storage characteristics, or shrink it to an optimal or specified size.

This section assumes that your database is running in rollback undo mode (the `UNDO_MANAGEMENT` initialization parameter is set to `MANUAL` or not set at all). If your database is running in automatic undo mode (the `UNDO_MANAGEMENT` initialization parameter is set to `AUTO`, which is the default), then user-created rollback segments are irrelevant.

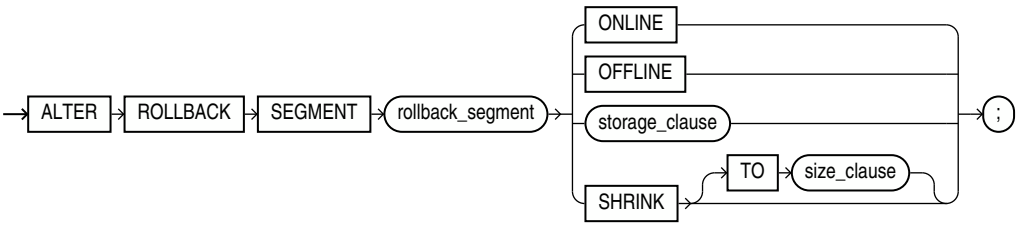
> **See Also:**
>
> - CREATE ROLLBACK SEGMENT for information on creating a rollback segment
> - *Oracle Database Reference* for information on the `UNDO_MANAGEMENT` parameter

Prerequisites

You must have the `ALTER ROLLBACK SEGMENT` system privilege.

Syntax

***alter_rollback_segment*::=**



(*storage_clause* , *size_clause*::=)

Semantics

rollback_segment

Specify the name of an existing rollback segment.

ONLINE

Specify `ONLINE` to bring the rollback segment online. When you create a rollback segment, it is initially offline and not available for transactions. This clause brings the rollback segment online, making it available for transactions by your instance. You can also bring a rollback segment online when you start your instance with the initialization parameter `ROLLBACK_SEGMENTS`.

> **See Also:**
>
> "Bringing a Rollback Segment Online: Example"

OFFLINE

Specify `OFFLINE` to take the rollback segment offline.

- If the rollback segment does not contain any information needed to roll back an active transaction, then Oracle Database takes it offline immediately.
- If the rollback segment does contain information for active transactions, then the database makes the rollback segment unavailable for future transactions and takes it offline after all the active transactions are committed or rolled back.

When the rollback segment is offline, it can be brought online by any instance.

To see whether a rollback segment is online or offline, query `STATUS` column of the data dictionary view `DBA_ROLLBACK_SEGS`. Online rollback segments have a value of `IN_USE`. Offline rollback segments have a value of `AVAILABLE`.

Restriction on Taking Rollback Segments Offline

You cannot take the SYSTEM rollback segment offline.

storage_clause

Use the *storage_clause* to change the storage characteristics of the rollback segment.

Restrictions on Rollback Segment Storage

You cannot change the value of INITIAL parameter. If the rollback segment is in a locally managed tablespace, then the only storage parameter you can change is OPTIMAL. If the rollback segment is in a dictionary-managed tablespace, then the only storage parameters you can change are NEXT, MINEXTENTS, MAXEXTENTS and OPTIMAL.

> **See Also:**
>
> *storage_clause* for syntax and additional information

SHRINK Clause

Specify SHRINK if you want Oracle Database to attempt to shrink the rollback segment to an optimal or specified size. The success and amount of shrinkage depend on the available free space in the rollback segment and how active transactions are holding space in the rollback segment.

If you do not specify TO *size_clause*, then the size defaults to the OPTIMAL value of the *storage_clause* of the CREATE ROLLBACK SEGMENT statement that created the rollback segment. If OPTIMAL was not specified, then the size defaults to the MINEXTENTS value of the *storage_clause* of the CREATE ROLLBACK SEGMENT statement.

Regardless of whether you specify TO *size_clause*:

- The value to which Oracle Database shrinks the rollback segment is valid for the execution of the statement. Thereafter, the size reverts to the OPTIMAL value of the CREATE ROLLBACK SEGMENT statement.
- The rollback segment cannot shrink to less than two extents.

To determine the actual size of a rollback segment after attempting to shrink it, query the BYTES, BLOCKS, and EXTENTS columns of the DBA_SEGMENTS view.

Restriction on Shrinking Rollback Segments

In an Oracle Real Application Clusters environment, you can shrink only rollback segments that are online to your instance.

> **See Also:**
>
> *size_clause* for information on that clause, and "Resizing a Rollback Segment: Example"

Examples

The following examples use the rbs_one rollback segment, which was created in "Creating a Rollback Segment: Example".

Bringing a Rollback Segment Online: Example

This statement brings the rollback segment `rbs_one` online:

```
ALTER ROLLBACK SEGMENT rbs_one ONLINE;
```

Resizing a Rollback Segment: Example

This statement shrinks the rollback segment `rbs_one`:

```
ALTER ROLLBACK SEGMENT rbs_one
   SHRINK TO 100M;
```

ALTER SEQUENCE

Purpose

Use the `ALTER SEQUENCE` statement to change the increment, minimum and maximum values, cached numbers, and behavior of an existing sequence. This statement affects only future sequence numbers.

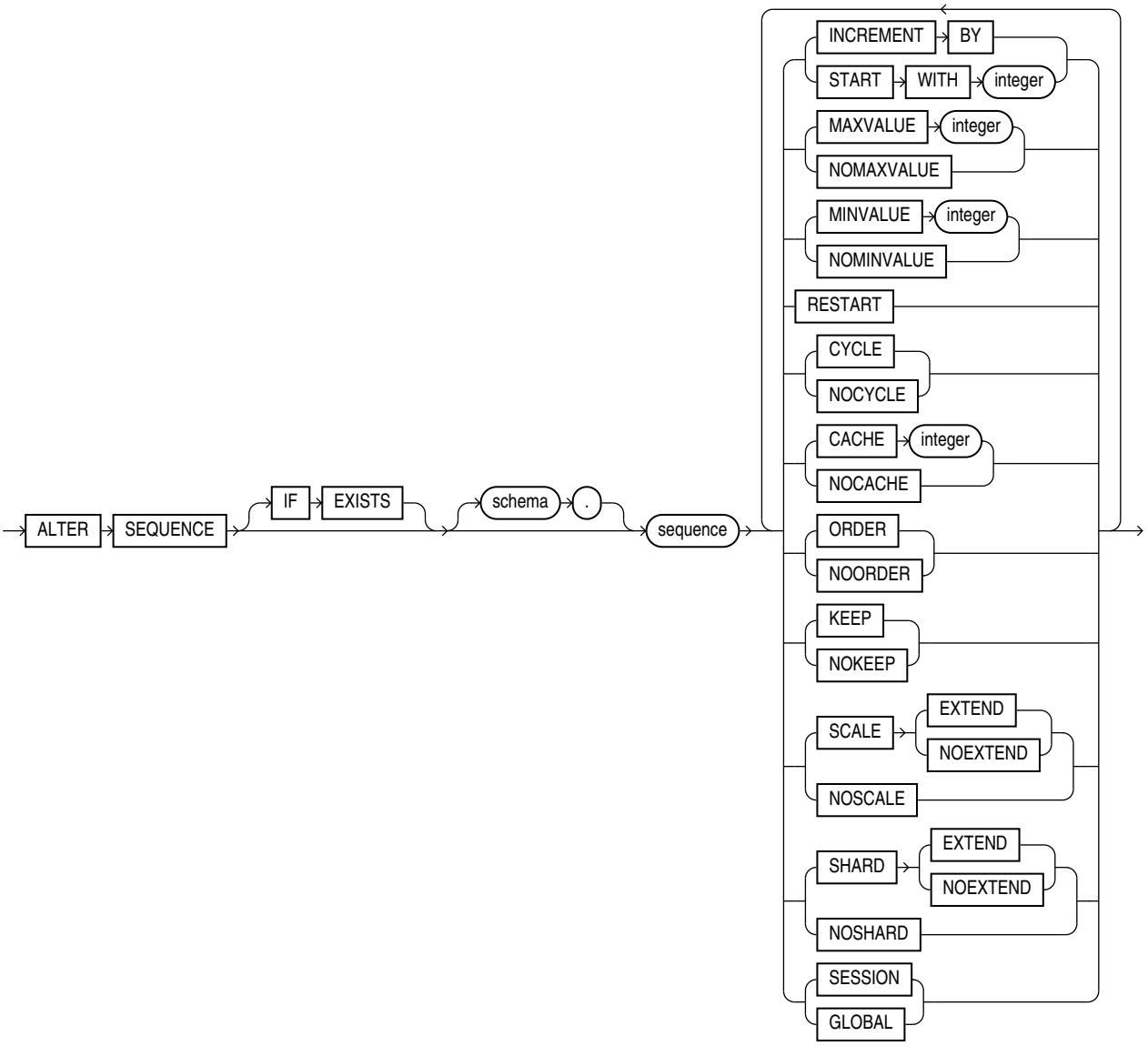
> **See Also:**
>
> CREATE SEQUENCE for additional information on sequences

Prerequisites

The sequence must be in your own schema, or you must have the `ALTER` object privilege on the sequence, or you must have the `ALTER ANY SEQUENCE` system privilege.

Syntax

***alter_sequence*::=**



Semantics

The keywords and parameters in this statement serve the same purposes they serve when you create a sequence.

- If you change the `INCREMENT BY` value before the first invocation of `NEXTVAL`, then some sequence numbers will be skipped. Therefore, if you want to retain the original `START WITH` value, you must drop the sequence and re-create it with the original `START WITH` value and the new `INCREMENT BY` value.
- Specify `RESTART` to reset `NEXTVAL` to `MINVALUE` for an ascending sequence. For a descending sequence `RESTART` resets `NEXTVAL` to `MAXVALUE`.

- To restart the sequence at a different number, specify `RESTART` with the `START WITH` clause to set the value at which the sequence restarts.
- If you alter the sequence by specifying the `KEEP` or `NOKEEP` clause between runtime and failover of a request, then the original value of `NEXTVAL` is not retained during replay for Application Continuity for that request.
- Oracle Database performs some validations. For example, a new `MAXVALUE` cannot be imposed that is less than the current sequence number.

> **See Also:**
>
> CREATE SEQUENCE for information on creating a sequence and DROP SEQUENCE for information on dropping and re-creating a sequence

IF EXISTS

Specify `IF EXISTS` to alter an existing table.

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

SCALE

Use `SCALE` to enable sequence scalability. When `SCALE` is specified, a numeric offset is affixed to the beginning of the sequence which removes all duplicates in generated values.

EXTEND

If you specify `EXTEND` with `SCALE` the generated sequence values are all of length *(x+y)*, where *x* is the length of the scalable offset (default value is 6), and *y* is the maximum number of digits in the sequence *(maxvalue/minvalue)*.

When you use `SCALE` it is highly recommended that you not use `ORDER` simultaneously on the sequence.

NOEXTEND

`NOEXTEND` is the default setting for the `SCALE` clause. With the `NOEXTEND` setting, the generated sequence values are at most as wide as the maximum number of digits in the sequence `(maxvalue/minvalue)`. This setting is useful for integration with existing applications where sequences are used to populate fixed width columns.

SHARD

Use this clause to generate unique sequence numbers across shards.

The sequence object is created as a global, all-shards sharded object that returns unique sequence values across all shards. The sequence object is also created at the catalog database that returns unique sequence values relative to the shard databases.

The `EXTEND` and `NOEXTEND` keywords define the behavior of a sharded sequence.

EXTEND

When you specify `EXTEND` with the `SHARD` clause, the generated sequence values are all of length *(x + y)*, where *x* is the length of an(a) `SHARD` offset of size 3. The size 3 corresponds to

the width of the maximum number of shards i.e. 1000 affixed at the beginning of the sequence values. *y* is the maximum number of digits in the sequence *maxvalue/minvalue*.

NOEXTEND

The default setting for the SHARD clause is NOEXTEND.

When you specify NOEXTEND, the generated sequence values are at most as wide as the maximum number of digits in the sequence *maxvalue/minvalue*. This setting is useful for integration with existing applications where sequences are used to populate fixed width columns.

If you call NEXTVAL on a sequence with SHARD NOEXTEND specified, a user error is thrown, if the generated value requires more digits of representation than the *maxvalue/minvalue* of the sequence.

Sequence with SHARD and SCALE

If you specify the SCALE and the SHARD clauses together, the sequence generates scalable, globally unique values within a shard database for multiple instances and sessions.

If you specify EXTEND with the SCALE and SHARD clauses, the generated sequence values are all of length *(x+y+z)*, where *x* is the length of a SHARD offset with a default value of size 4, *y* is the length of the scalable offset with a default value of 6(5), and *z* is the maximum number of digits in the sequence *maxvalue/minvalue* .

If you specify EXTEND or NOEXTEND with the SHARD and SCALE clauses, it applies to both SHARD and SCALE. You do not need to specify EXTEND or NOEXTEND separately. If you specify the EXTEND or NOEXTEND option separately for both the SHARD and SCALE clauses, with the same or different value, a parsing error results, with a message of a duplicate or conflicting EXTEND clause.

When you use SHARD it is highly recommended that you not use ORDER simultaneously on the sequence.

You can use SHARD with CACHE and NOCACHE modes of operation.

> **Note:**
>
> - Starting with Oracle Database Release 23 a sharded sequence without scale will have the leading "1" of the offset removed.
> - Starting with Oracle Database Release 23 a sharded sequence with scale will have the leading "1" of the offset removed.
>
> Prior to Release 23, a a sharded sequence with scale had one leading "1" for the combined offset. This leading "1" is removed from Release 23 onwards.

Examples

Modifying a Sequence: Examples

This statement sets a new maximum value for the customers_seq sequence, which was created in "Creating a Sequence: Example":

```
ALTER SEQUENCE customers_seq
   MAXVALUE 1500;
```

This statement turns on `CYCLE` and `CACHE` for the `customers_seq` sequence:

```
ALTER SEQUENCE customers_seq
   CYCLE
   CACHE 5;
```

ALTER SESSION

Purpose

Use the `ALTER SESSION` statement to set or modify any of the conditions or parameters that affect your connection to the database. The statement stays in effect until you disconnect from the database.

Prerequisites

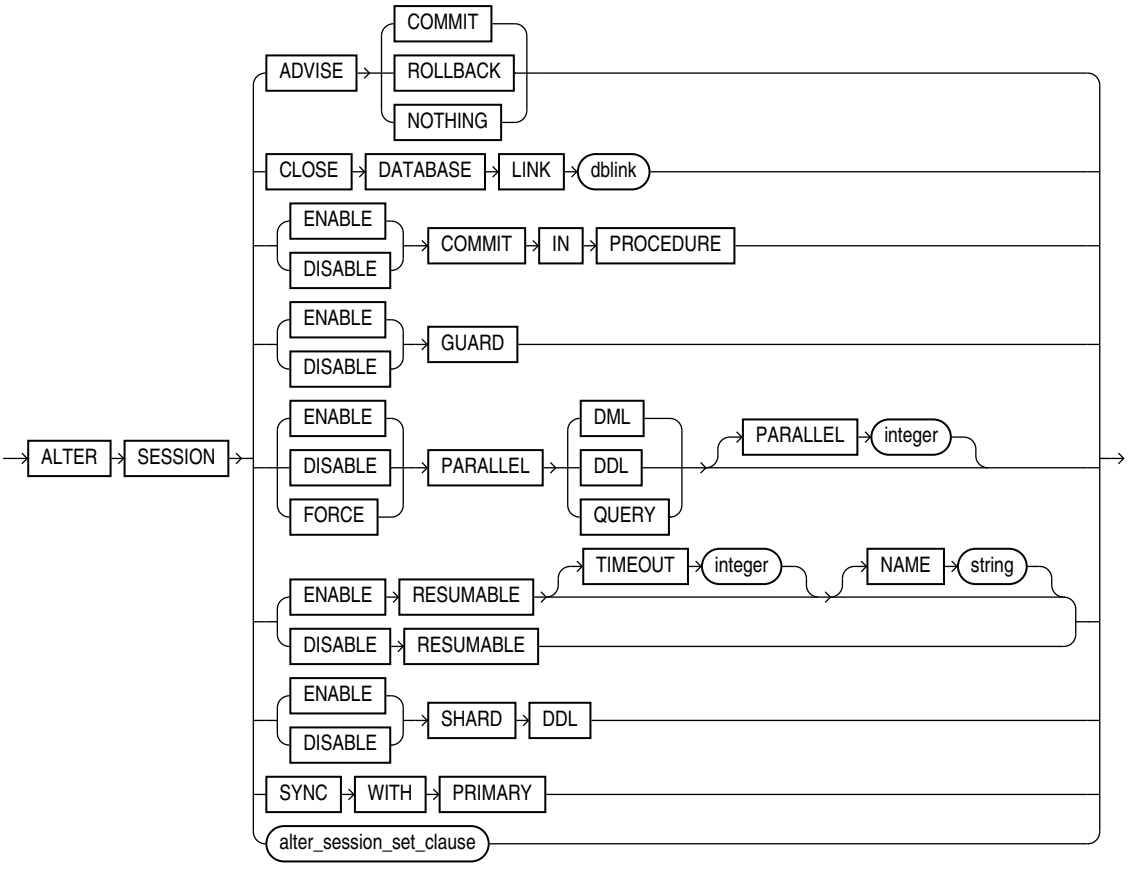
To enable and disable the SQL trace facility, you must have `ALTER SESSION` system privilege.

To enable or disable resumable space allocation, you must have the `RESUMABLE` system privilege.

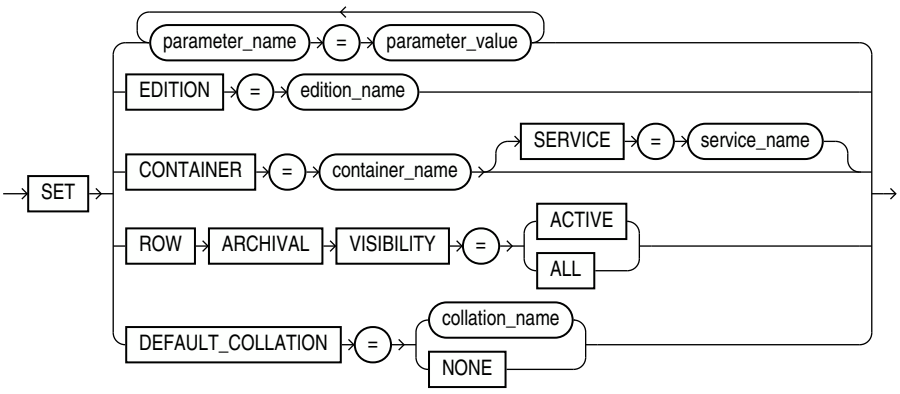
You do not need any privileges to perform the other operations of this statement unless otherwise indicated.

Syntax

***alter_session*::=**



***alter_session_set_clause*::=**



Semantics

ADVISE Clause

The `ADVISE` clause sends advice to a remote database to force a distributed transaction. The advice appears in the `ADVICE` column of the `DBA_2PC_PENDING` view on the remote database (the values are '`C`' for `COMMIT`, '`R`' for `ROLLBACK`, and ' ' for `NOTHING`). If the transaction becomes in doubt, then the administrator of that database can use this advice to decide whether to commit or roll back the transaction.

You can send different advice to different remote databases by issuing multiple `ALTER SESSION` statements with the `ADVISE` clause in a single transaction. Each such statement sends advice to the databases referenced in the following statements in the transaction until another such statement is issued.

> **See Also:**
>
> "Forcing a Distributed Transaction: Example"

CLOSE DATABASE LINK Clause

Specify `CLOSE DATABASE LINK` to close the database link *`dblink`*. When you issue a statement that uses a database link, Oracle Database creates a session for you on the remote database using that link. The connection remains open until you end your local session or until the number of database links for your session exceeds the value of the initialization parameter `OPEN_LINKS`. If you want to reduce the network overhead associated with keeping the link open, then use this clause to close the link explicitly if you do not plan to use it again in your session.

> **See Also:**
>
> Closing a Database Link: Example

ENABLE | DISABLE COMMIT IN PROCEDURE

Procedures and stored functions written in PL/SQL can issue `COMMIT` and `ROLLBACK` statements. If your application would be disrupted by a `COMMIT` or `ROLLBACK` statement not issued directly by the application itself, then specify `DISABLE COMMIT IN PROCEDURE` clause to prevent procedures and stored functions called during your session from issuing these statements.

You can subsequently allow procedures and stored functions to issue `COMMIT` and `ROLLBACK` statements in your session by issuing the `ENABLE COMMIT IN PROCEDURE`.

Some applications automatically prohibit `COMMIT` and `ROLLBACK` statements in procedures and stored functions. Refer to your application documentation for more information.

ENABLE | DISABLE GUARD

The *`security_clause`* of `ALTER DATABASE` lets you prevent anyone other than the `SYS` user from making any changes to data or database objects on the primary or standby database. This clause lets you override that setting for the current session.

> **See Also:**
>
> *security_clause* for more information on the `GUARD` setting

PARALLEL DML | DDL | QUERY

The `PARALLEL` parameter determines whether all subsequent DML, DDL, or query statements in the session will be considered for parallel execution. This clause enables you to override the degree of parallelism of tables during the current session without changing the tables themselves. Uncommitted transactions must either be committed or rolled back prior to executing this clause for DML.

> **See Also:**
>
> "Enabling Parallel DML: Example"

ENABLE Clause

Specify `ENABLE` to execute subsequent statements in the session in parallel. This is the default for DDL and query statements.

- `DML`: DML statements are executed in parallel mode if a parallel hint or a parallel clause is specified.
- `DDL`: DDL statements are executed in parallel mode if a parallel clause is specified.
- `QUERY`: Queries are executed in parallel mode if a parallel hint or a parallel clause is specified.

Restriction on the ENABLE clause

You cannot specify the optional `PARALLEL` *`integer`* with `ENABLE`.

DISABLE Clause

Specify `DISABLE` to execute subsequent statements in the session serially. This is the default for DML statements.

- `DML`: DML statements are executed serially.
- `DDL`: DDL statements are executed serially.
- `QUERY`: Queries are executed serially.

Restriction on the DISABLE clause

You cannot specify the optional `PARALLEL` *`integer`* with `DISABLE`.

FORCE Clause

`FORCE` forces parallel execution of subsequent statements in the session. If no parallel clause or hint is specified, then a default degree of parallelism is used. This clause overrides any *`parallel_clause`* specified in subsequent statements in the session but is overridden by a parallel hint.

- `DML`: Provided no parallel DML restrictions are violated, subsequent DML statements in the session are executed with the default degree of parallelism, unless a degree is specified in this clause.
- `DDL`: Subsequent DDL statements in the session are executed with the default degree of parallelism, unless a degree is specified in this clause. Resulting database objects will have associated with them the prevailing degree of parallelism.

  Specifying `FORCE DDL` automatically causes all tables created in this session to be created with a default level of parallelism. The effect is the same as if you had specified the *`parallel_clause`* (with the default degree) in the `CREATE TABLE` statement.
- `QUERY`: Subsequent queries are executed with the default degree of parallelism, unless a degree is specified in this clause.

PARALLEL *integer*

Specify an integer to explicitly specify a degree of parallelism:

- For `FORCE DDL`, the degree overrides any parallel clause in subsequent DDL statements.
- For `FORCE DML` and `QUERY`, the degree overrides the degree currently stored for the table in the data dictionary.
- A degree specified in a statement through a hint will override the degree being forced.

The following types of DML operations are not parallelized regardless of this clause:

- Operations on cluster tables
- Operations with embedded functions that either write or read database or package states
- Operations on tables with triggers that could fire
- Operations on tables or schema objects containing object types, or `LONG` or LOB data types

RESUMABLE Clauses

These clauses let you enable and disable resumable space allocation. This feature allows an operation to be suspended in the event of an out-of-space error condition and to resume automatically from the point of interruption when the error condition is fixed.

> **Note:**
>
> Resumable space allocation is fully supported for operations on locally managed tablespaces. Some restrictions apply if you are using dictionary-managed tablespaces. For information on these restrictions, refer to *Oracle Database Administrator's Guide*.

ENABLE RESUMABLE

This clause enables resumable space allocation for the session.

TIMEOUT

`TIMEOUT` lets you specify (in seconds) the time during which an operation can remain suspended while waiting for the error condition to be fixed. If the error condition is not fixed within the `TIMEOUT` period, then Oracle Database aborts the suspended operation.

NAME

`NAME` lets you specify a user-defined text string to help users identify the statements issued during the session while the session is in resumable mode. Oracle Database inserts the text string into the `USER_RESUMABLE` and `DBA_RESUMABLE` data dictionary views. If you do not specify `NAME`, then Oracle Database inserts the default string '`User` *`username`*`(`*`userid`*`), Session` *`sessionid`*`, Instance` *`instanceid`*'.

> **See Also:**
>
> *Oracle Database Reference* for information on the data dictionary views

DISABLE RESUMABLE

This clause disables resumable space allocation for the session.

SHARD DDL Clauses

These clauses are valid only if you are connected to a sharded database. They let you control whether DDLs issued in the session are issued against the shard catalog database and all shards, or against only the shard catalog database.

- If you specify `ENABLE SHARD DDL`, then DDLs issued in the session are issued against the shard catalog database and all shards. This mode is the default for the SDB user—a user that exists in the shard catalog database and in all shards.
- If you specify `DISABLE SHARD DDL`, then DDLs issued in the session are issued against only the shard catalog database. This mode is the default for a local user—a user that exists only in the shard catalog database.

> **See Also:**
>
> *Using Oracle Sharding*

SYNC WITH PRIMARY

Use this clause to synchronize redo apply on a physical standby database with the primary database. An `ALTER SESSION` statement with this clause blocks until redo apply has applied all redo data received by the standby at the time the statement is issued. This clause returns an error, and synchronization does not occur, if the redo transport state for the standby database is not `SYNCHRONIZED` or if redo apply is not active.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for more information on this session parameter

alter_session_set_clause

Use the *alter_session_set_clause* to set initialization parameter values or to set an edition for the current session.

Initialization Parameters

You can set two types of parameters using this clause:

- Initialization parameters that are dynamic in the scope of the `ALTER SESSION` statement (listed in "Initialization Parameters and ALTER SESSION")
- Session parameters (listed in "Session Parameters and ALTER SESSION ")

You can set values for multiple parameters in the same *alter_session_set_clause*.

EDITION

Specify `EDITION` = *edition* to set the specified edition as the edition in the database session. You must have the `USE` object privilege on *edition*, *edition* must already have been created, and it must be `USABLE`.

When this statement is successful, the database discards PL/SQL package state corresponding to editionable packages but retains package state corresponding to packages that are not editionable.

You can also set the edition for the current session at startup with the `EDITION` parameter of the SQL*Plus `CONNECT` command. However, you cannot specify an `ALTER SESSION SET EDITION` statement in a recursive SQL or PL/SQL block.

You can determine the edition in use by the current session with the following query:

```
SELECT SYS_CONTEXT('USERENV', 'CURRENT_EDITION_NAME') FROM DUAL;
```

> **See Also:**
>
> CREATE EDITION for more information on editions and *Oracle Database PL/SQL Language Reference* for information on how editions are designated as `USABLE`

CONTAINER

Use this clause in a multitenant container database (CDB) to switch to the container specified by *container_name*.

To use this clause, you must be a common user with the SET CONTAINER privilege, either granted commonly or granted locally in *container_name*.

For *container_name*, specify one of the following:

- CDB$ROOT to switch to the root
- PDB$SEED to switch to the seed
- A pluggable database (PDB) name to switch to that PDB. You can view the names of the PDBs in a CDB by querying the DBA_PDBS view.

You can determine the container to which the current session is connected by using the SQL*Plus SHOW CON_NAME command or with the following SQL query:

```
SELECT SYS_CONTEXT('USERENV', 'CON_NAME') FROM DUAL;
```

SERVICE

By default, when you switch to a container, the session uses the default service for the container. Specify the SERVICE clause to use a different service for the container. For *service_name*, specify the name of the service you want to use.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information on switching to a container

ROW ARCHIVAL VISIBILITY

Use this clause to configure row archival visibility for the session. This clause lets you implement In-Database Archiving, which allows you to designate table rows as active or archived. You can then perform queries on only the active rows within the table.

- If you specify ACTIVE, then the database will consider only active rows when performing queries on tables that are enabled for row archival. This is the default.
- If you specify ALL, then the database will consider all rows when performing queries on tables that are enabled for row archival.

This clause has no effect on queries on tables that are not enabled for row archival.

> **See Also:**
>
> - The `CREATE TABLE` ROW ARCHIVAL clause to learn how to enable a new table for row archival
> - The `ALTER TABLE` [NO] ROW ARCHIVAL clause to learn how to enable or disable an existing table for row archival
> - *Oracle Database VLDB and Partitioning Guide* for more information on In-Database Archiving

DEFAULT_COLLATION

Use this clause to set the default collation for the session.

- Use *`collation_name`* to specify the default collation for the session. You can specify the name of any valid named collation or pseudo-collation. This collation becomes the *effective schema default collation*. This collation is assigned to tables, views, and materialized views that are subsequently created in any schema for the duration of the session. The default collation for the session does not get propagated to any remote sessions connected to the current session using DB links.
- If you specify `NONE`, then there is no default collation for the session. In this case, the default collation for a particular schema becomes the *effective schema default collation* for that schema. That default collation is assigned to tables, views, and materialized views that are subsequently created in the schema for the duration of the session.

In either of the preceding cases, you can override the effective schema default collation and assign a default collation to a particular table, materialized view, or view by specifying the `DEFAULT COLLATION` clause of the `CREATE` or `ALTER` statement for the table, materialized view, or view.

The effective schema default collation also affects the DDL statements `CREATE FUNCTION`, `CREATE PACKAGE`, `CREATE PROCEDURE`, `CREATE TRIGGER`, and `CREATE TYPE`. Refer to *Oracle Database PL/SQL Language Reference* for more details on these statements.

You can query the default collation for a session with the following statement:

```
SELECT SYS_CONTEXT('USERENV', 'SESSION_DEFAULT_COLLATION') FROM DUAL;
```

You can specify the `SET DEFAULT_COLLATION` clause only if the `COMPATIBLE` initialization parameter is set to `12.2` or greater, and the `MAX_STRING_SIZE` initialization parameter is set to `EXTENDED`.

> **See Also:**
>
> The DEFAULT COLLATION Clause clause of `CREATE USER` for more information on the default collation of a schema

> **Note:**
>
> The effective schema default collation for a session should not be confused with the session parameter NLS_SORT. The effective schema default collation is used by DDL statements to decide the default data-bound collation of tables, views, and materialized views when they are created. The session parameter NLS_SORT points to a named collation that is used when Oracle executes a query, a DML statement, or PL/SQL code containing a SQL operation whose determined collation is a pseudo-collation, such as USING_NLS_COMP or USING_NLS_SORT. Refer to *Oracle Database Globalization Support Guide* for more information.

Initialization Parameters and ALTER SESSION

Some initialization parameter are dynamic in the scope of ALTER SESSION. When you set these parameters using ALTER SESSION, the value you set persists only for the duration of the current session.To determine whether a parameter can be altered using an ALTER SESSION statement, query the ISSES_MODIFIABLE column of the V$PARAMETER dynamic performance view.

> **Note:**
>
> Before changing the values of initialization parameters, refer to their full description in *Oracle Database Reference*.

A number of parameters that can be set using ALTER SESSION are not initialization parameters. You can set them only with ALTER SESSION, not in an initialization parameter file. Those session parameters are described in "Session Parameters and ALTER SESSION ".

Session Parameters and ALTER SESSION

The following parameters are session parameters only, not initialization parameters:

CONSTRAINT[S]

Syntax:

```
CONSTRAINT[S] = { IMMEDIATE | DEFERRED | DEFAULT }
```

The CONSTRAINT[S] parameter determines when conditions specified by a deferrable constraint are enforced.

- IMMEDIATE indicates that the conditions specified by the deferrable constraint are checked immediately after each DML statement. This setting is equivalent to issuing the SET CONSTRAINTS ALL IMMEDIATE statement at the beginning of each transaction in your session.
- DEFERRED indicates that the conditions specified by the deferrable constraint are checked when the transaction is committed. This setting is equivalent to issuing the SET CONSTRAINTS ALL DEFERRED statement at the beginning of each transaction in your session.
- DEFAULT restores all constraints at the beginning of each transaction to their initial state of DEFERRED or IMMEDIATE.

CURRENT_SCHEMA

Syntax:

```
CURRENT_SCHEMA = schema
```

The CURRENT_SCHEMA parameter changes the current schema of the session to the specified schema. Subsequent unqualified references to schema objects during the session will resolve to objects in the specified schema. The setting persists for the duration of the session or until you issue another ALTER SESSION SET CURRENT_SCHEMA statement.

This setting offers a convenient way to perform operations on objects in a schema other than that of the current user without having to qualify the objects with the schema name. This setting changes the current schema, but it does not change the session user or the current user, nor does it give the session user any additional system or object privileges for the session.

ERROR_ON_OVERLAP_TIME

Syntax:

```
ERROR_ON_OVERLAP_TIME = {TRUE | FALSE}
```

The ERROR_ON_OVERLAP_TIME parameter determines how Oracle Database should handle an ambiguous boundary datetime value—a case in which it is not clear whether the datetime is in standard or daylight saving time.

- Specify TRUE to return an error for the ambiguous overlap timestamp.
- Specify FALSE to default the ambiguous overlap timestamp to the standard time. This is the default.

Refer to "Support for Daylight Saving Times " for more information on boundary datetime values.

FLAGGER

Syntax:

```
FLAGGER = { ENTRY | OFF }
```

The FLAGGER parameter specifies FIPS flagging (as specified in Federal Information Processing Standard 127-2), which causes an error message to be generated when a SQL statement issued is an extension of the Entry Level of SQL-92 (officially, ANSI X3.135-1992, a standard that is now superseded by SQL:2016). FLAGGER is a session parameter only, not an initialization parameter.

After flagging is set in a session, a subsequent ALTER SESSION SET FLAGGER statement will work, but generates the message, ORA-00097. This allows FIPS flagging to be altered without disconnecting the session. OFF turns off flagging.

> **See Also:**
>
> Oracle and Standard SQL, for more information about Oracle compliance with current ANSI SQL standards
>
> Starting with Oracle Database 23ai, several parameters associated with `FIPS_140` are deprecated.
>
> `FIPS_140` in `FIPS.ORA` can be used to enable `FIPS` for all features starting with Oracle Database 23ai. The following FIPS parameters are deprecated:
>
> - SQLNET.ORA: `FIPS_140` to enable FIPS for native network encryption
> - FIPS.ORA: `SSLFIPS_140` to enable FIPS for TLS
> - Initialization parameter: `DBFIPS_140` to enable FIPS for TDE and `DBMS_CRYPTO`

INSTANCE

Syntax:

```
INSTANCE = integer
```

Setting the `INSTANCE` parameter lets you access another instance as if you were connected to your own instance. `INSTANCE` is a session parameter only, not an initialization parameter. In an Oracle Real Application Clusters (Oracle RAC) environment, each Oracle RAC instance retains static or dynamic ownership of disk space for optimal DML performance based on the setting of this parameter.

ISOLATION_LEVEL

Syntax:

```
ISOLATION_LEVEL = {SERIALIZABLE | READ COMMITTED}
```

The `ISOLATION_LEVEL` parameter specifies how transactions containing database modifications are handled. `ISOLATION_LEVEL` is a session parameter only, not an initialization parameter.

- `SERIALIZABLE` indicates that transactions in the session use the serializable transaction isolation mode as specified in the SQL standard. If a serializable transaction attempts to execute a DML statement that updates rows currently being updated by another uncommitted transaction at the start of the serializable transaction, then the DML statement fails. A serializable transaction can see its own updates.
- `READ COMMITTED` indicates that transactions in the session will use the default Oracle Database transaction behavior. If the transaction contains DML that requires row locks held by another transaction, then the DML statement will wait until the row locks are released.

> **Note:**
>
> Serializable transactions do not work with deferred segment creation or interval partitioning. Trying to insert data into an empty table with no segment created, or into a partition of an interval partitioned table that does not yet have a segment, causes an error.

STANDBY_MAX_DATA_DELAY

Syntax:

```
STANDBY_MAX_DATA_DELAY =  { integer | NONE }
```

In an Active Data Guard environment, this session parameter can be used to specify a session-specific apply lag tolerance, measured in seconds, for queries issued by non-administrative users to a physical standby database that is in real-time query mode. This capability allows queries to be safely offloaded from the primary database to a physical standby database, because it is possible to detect if the standby database has become unacceptably stale.

If `STANDBY_MAX_DATA_DELAY` is set to the default value of `NONE`, queries issued to a physical standby database will be executed regardless of the apply lag on that database.

If `STANDBY_MAX_DATA_DELAY` is set to a nonzero value, a query issued to a physical standby database will be executed only if the apply lag is less than or equal to `STANDBY_MAX_DATA_DELAY`. Otherwise, an `ORA-3172` error is returned to alert the client that the apply lag is too large.

If `STANDBY_MAX_DATA_DELAY` is set to `0`, a query issued to a physical standby database is guaranteed to return the exact same result as if the query were issued on the primary database, unless the standby database is lagging behind the primary database, in which case an `ORA-3172` error is returned.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for more information on Active Data Guard and using this session parameter

TIME_ZONE

Syntax:

```
TIME_ZONE =  '[+ | -] hh:mi'
             | LOCAL
             | DBTIMEZONE
             | 'time_zone_region'
```

The `TIME_ZONE` parameter specifies the default local time zone offset or region name for the current SQL session. `TIME_ZONE` is a session parameter only, not an initialization parameter. To determine the time zone of the current session, query the built-in function `SESSIONTIMEZONE` (see SESSIONTIMEZONE).

- Specify a format mask (`'[+|-]hh:mi'`) indicating the hours and minutes before or after UTC (Coordinated Universal Time—formerly Greenwich Mean Time). The valid range for *hh:mi* is -12:00 to +14:00.
- Specify `LOCAL` to set the default local time zone offset of the current SQL session to the original default local time zone offset that was established when the current SQL session was started.
- Specify `DBTIMEZONE` to set the current session time zone to match the value set for the database time zone. If you specify this setting, then the `DBTIMEZONE` function will return the database time zone as a UTC offset or a time zone region, depending on how the database time zone has been set.
- Specify a valid *time_zone_region*. To see a listing of valid time zone region names, query the `TZNAME` column of the `V$TIMEZONE_NAMES` dynamic performance view. If you specify this setting, then the `SESSIONTIMEZONE` function will return the region name.

> **Note:**
>
> Time zone region names are needed by the daylight saving feature. These names are stored in two types of time zone files: one large and one small. One of these files is the default file, depending on your environment and the release of Oracle Database you are using. For more information regarding time zone files and names, see *Oracle Database Globalization Support Guide*.

> **See Also:**
>
> *Oracle Database Globalization Support Guide* for a complete listing of the time zone region names in both files

> **Note:**
>
> You can also set the default client session time zone using the `ORA_SDTZ` environment variable. Refer to *Oracle Database Globalization Support Guide* for more information on this variable.

USE_PRIVATE_OUTLINES

Syntax:

```
USE_PRIVATE_OUTLINES = { TRUE | FALSE | category_name }
```

The `USE_PRIVATE_OUTLINES` parameter lets you control the use of private outlines. When this parameter is enabled and an outlined SQL statement is issued, the optimizer retrieves the outline from the session private area rather than the public area used when `USE_STORED_OUTLINES` is enabled. If no outline exists in the session private area, then the optimizer will not use an outline to compile the statement. `USE_PRIVATE_OUTLINES` is not an initialization parameter.

- TRUE causes the optimizer to use private outlines stored in the DEFAULT category when compiling requests.
- FALSE specifies that the optimizer should not use stored private outlines. This is the default. If USE_STORED_OUTLINES is enabled, then the optimizer will use stored public outlines.
- *category_name* causes the optimizer to use outlines stored in the *category_name* category when compiling requests.

Restriction on USE_PRIVATE_OUTLINES

You cannot enable this parameter if USE_STORED_OUTLINES is enabled.

USE_STORED_OUTLINES

> **Note:**
>
> Stored outlines are deprecated. They are still supported for backward compatibility. However, Oracle recommends that you use SQL plan management instead. Refer to *Oracle Database SQL Tuning Guide* for more information about SQL plan management.

Syntax:

```
USE_STORED_OUTLINES = { TRUE | FALSE | category_name }
```

The USE_STORED_OUTLINES parameter determines whether the optimizer will use stored public outlines to generate execution plans. USE_STORED_OUTLINES is not an initialization parameter.

- TRUE causes the optimizer to use outlines stored in the DEFAULT category when compiling requests.
- FALSE specifies that the optimizer should not use stored outlines. This is the default.
- *category_name* causes the optimizer to use outlines stored in the *category_name* category when compiling requests.

Restriction on USED_STORED_OUTLINES

You cannot enable this parameter if USE_PRIVATE_OUTLINES is enabled.

Examples

Enabling Parallel DML: Example

Issue the following statement to enable parallel DML mode for the current session:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Forcing a Distributed Transaction: Example

The following transaction inserts an employee record into the employees table on the database identified by the database link remote and deletes an employee record from the employees table on the database identified by local:

```
ALTER SESSION
   ADVISE COMMIT;
```

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

ALTER SESSION
   ADVISE ROLLBACK;

DELETE FROM employees@local
   WHERE employee_id = 8002;

COMMIT;
```

This transaction has two `ALTER SESSION` statements with the `ADVISE` clause. If the transaction becomes in doubt, then `remote` is sent the advice '`COMMIT`' by virtue of the first `ALTER SESSION` statement and `local` is sent the advice '`ROLLBACK`' by virtue of the second statement.

Closing a Database Link: Example

This statement updates the `jobs` table on the `local` database using a database link, commits the transaction, and explicitly closes the database link:

```
UPDATE jobs@local SET min_salary = 3000
   WHERE job_id = 'SH_CLERK';

COMMIT;

ALTER SESSION
   CLOSE DATABASE LINK local;
```

Changing the Date Format Dynamically: Example

The following statement dynamically changes the default date format for your session to `'YYYY MM DD-HH24:MI:SS'`:

```
ALTER SESSION
   SET NLS_DATE_FORMAT = 'YYYY MM DD HH24:MI:SS';
```

Oracle Database uses the new default date format:

```
SELECT TO_CHAR(SYSDATE) Today
   FROM DUAL;

TODAY
-------------------
2001 04 12 12:30:38
```

Changing the Date Language Dynamically: Example

The following statement changes the language for date format elements to French:

```
ALTER SESSION
   SET NLS_DATE_LANGUAGE = French;

SELECT TO_CHAR(SYSDATE, 'Day DD Month YYYY') Today
   FROM DUAL;

TODAY
---------------------------
Jeudi    12 Avril     2001
```

Changing the ISO Currency: Example

The following statement dynamically changes the ISO currency symbol to the ISO currency symbol for the territory America:

```
ALTER SESSION
   SET NLS_ISO_CURRENCY = America;

SELECT TO_CHAR( SUM(salary), 'C999G999D99') Total
   FROM employees;

TOTAL
------------------
     USD694,900.00
```

Changing the Decimal Character and Group Separator: Example

The following statement dynamically changes the decimal character to comma (,) and the group separator to period (.):

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ',.' ;
```

Oracle Database returns these new characters when you use their number format elements:

```
ALTER SESSION SET NLS_CURRENCY = 'FF';

SELECT TO_CHAR( SUM(salary), 'L999G999D99') Total FROM employees;

TOTAL
---------------------
         FF694.900,00
```

Changing the NLS Currency: Example

The following statement dynamically changes the local currency symbol to 'DM':

```
ALTER SESSION
   SET NLS_CURRENCY = 'DM';

SELECT TO_CHAR( SUM(salary), 'L999G999D99') Total
   FROM employees;

TOTAL
---------------------
         DM694.900,00
```

Changing the NLS Language: Example

The following statement dynamically changes to French the language in which error messages are displayed:

```
ALTER SESSION
   SET NLS_LANGUAGE = FRENCH;

Session modifiee.

SELECT * FROM DMP;

ORA-00942: Table ou vue inexistante
```

Changing the Linguistic Sort Sequence: Example

The following statement dynamically changes the linguistic sort sequence to Spanish:

```
ALTER SESSION
   SET NLS_SORT = XSpanish;
```

Oracle Database sorts character values based on their position in the Spanish linguistic sort sequence.

Enabling Query Rewrite: Example

This statement enables query rewrite in the current session for all materialized views that have not been explicitly disabled:

```
ALTER SESSION
  SET QUERY_REWRITE_ENABLED = TRUE;
```