10 SQL Statements: ADMINISTER KEY MANAGEMENT to ALTER JSON RELATIONAL DUALITY VIEW

This chapter lists the various types of SQL statements and then describes the first set (in alphabetical order) of SQL statements. The remaining SQL statements appear in alphabetical order in the subsequent chapters.

This chapter contains the following sections:

- Types of SQL Statements
- How the SQL Statement Chapters are Organized
- ADMINISTER KEY MANAGEMENT
- ALTER ANALYTIC VIEW
- ALTER ATTRIBUTE DIMENSION
- ALTER AUDIT POLICY (Unified Auditing)
- ALTER CLUSTER
- ALTER DATABASE
- ALTER DATABASE DICTIONARY
- ALTER DATABASE LINK
- ALTER DIMENSION
- ALTER DISKGROUP
- ALTER DOMAIN
- ALTER FLASHBACK ARCHIVE
- ALTER FUNCTION
- ALTER HIERARCHY
- ALTER INDEX
- ALTER INDEXTYPE
- ALTER INMEMORY JOIN GROUP
- ALTER JAVA
- ALTER JSON RELATIONAL DUALITY VIEW

Types of SQL Statements

The lists in the following sections provide a functional summary of SQL statements and are divided into these categories:

- Data Definition Language (DDL) Statements

- Data Manipulation Language (DML) Statements
- Transaction Control Statements
- Session Control Statements
- System Control Statements
- Embedded SQL Statements

Data Definition Language (DDL) Statements

Data definition language (DDL) statements let you to perform these tasks:

- Create, alter, and drop schema objects
- Grant and revoke privileges and roles
- Analyze information on a table, index, or cluster
- Establish auditing options
- Add comments to the data dictionary

The `CREATE`, `ALTER`, and `DROP` commands require exclusive access to the specified object. For example, an `ALTER TABLE` statement fails if another user has an open transaction on the specified table.

The `GRANT`, `REVOKE`, `ANALYZE`, `AUDIT`, and `COMMENT` commands do not require exclusive access to the specified object. For example, you can analyze a table while other users are updating the table.

Oracle Database implicitly commits the current transaction before and after every DDL statement.

A DDL statement is either blocking or nonblocking, and both types of DDL statements require exclusive locks on internal structures.

> **See Also:**
>
> *Oracle Database Development Guide* to learn about the difference between blocking and nonblocking DDL

Many DDL statements may cause Oracle Database to recompile or reauthorize schema objects. For information on how Oracle Database recompiles and reauthorizes schema objects and the circumstances under which a DDL statement would cause this, see *Oracle Database Concepts*.

DDL statements are supported by PL/SQL with the use of the `DBMS_SQL` package.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for more information about this package

The DDL statements are:

ALTER ... (All statements beginning with ALTER, except ALTER SESSION and ALTER SYSTEM—see "Session Control Statements " and "System Control Statements")

ANALYZE

ASSOCIATE STATISTICS

AUDIT

COMMENT

CREATE ... (All statements beginning with CREATE)

DISASSOCIATE STATISTICS

DROP ... (All statements beginning with DROP)

FLASHBACK ... (All statements beginning with FLASHBACK)

GRANT

NOAUDIT

PURGE

RENAME

REVOKE

TRUNCATE

Data Manipulation Language (DML) Statements

Data manipulation language (DML) statements access and manipulate data in existing schema objects. These statements do not implicitly commit the current transaction. The data manipulation language statements are:

CALL

DELETE

EXPLAIN PLAN

INSERT

LOCK TABLE

MERGE

SELECT

UPDATE

The SELECT statement is a limited form of DML statement in that it can only access data in the database. It cannot manipulate data stored in the database, although it can manipulate the accessed data before returning the results of the query.

The SELECT statement is supported in PL/SQL only when executed dynamically. However, you can use the similar PL/SQL statement SELECT INTO in PL/SQL code, and you do not have to execute it dynamically. The CALL and EXPLAIN PLAN statements are supported in PL/SQL only when executed dynamically. All other DML statements are fully supported in PL/SQL.

Transaction Control Statements

Transaction control statements manage changes made by DML statements. The transaction control statements are:

COMMIT

ROLLBACK

SAVEPOINT

SET TRANSACTION

SET CONSTRAINT

All transaction control statements, except certain forms of the `COMMIT` and `ROLLBACK` commands, are supported in PL/SQL. For information on the restrictions, see COMMIT and ROLLBACK .

Session Control Statements

Session control statements dynamically manage the properties of a user session. These statements do not implicitly commit the current transaction.

PL/SQL does not support session control statements. The session control statements are:

```
ALTER SESSION
SET ROLE
```

System Control Statements

- Use `ADMINISTER KEY MANAGEMENT` to manage software and hardware keystores, encryption keys, and secrets.
- Use `ALTER SYSTEM` to dynamically manage the properties of an Oracle Database instance.

  This statement does not implicitly commit the current transaction and is not supported in PL/SQL.

Embedded SQL Statements

Embedded SQL statements place DDL, DML, and transaction control statements within a procedural language program. Embedded SQL is supported by the Oracle precompilers and is documented in the following books:

- *Pro*COBOL Programmer's Guide*
- *Pro*C/C++ Programmer's Guide*

How the SQL Statement Chapters are Organized

All SQL statements in this book are organized into the following sections:

Syntax

The syntax diagrams show the keywords and parameters that make up the statement.

> **Note:**
>
> Not all keywords and parameters are valid in all circumstances. Be sure to refer to the "Semantics" section of each statement and clause to learn about any restrictions on the syntax.

Purpose

The "Purpose" section describes the basic uses of the statement.

Prerequisites

The "Prerequisites" section lists privileges you must have and steps that you must take before using the statement. In addition to the prerequisites listed, most statements also require that the database be opened by your instance, unless otherwise noted.

Semantics

The "Semantics" section describes the purpose of the keywords, parameters, and clauses that make up the syntax, as well as restrictions and other usage notes that may apply to them. (The conventions for keywords and parameters used in this chapter are explained in the "Preface" of this reference.)

Examples

The "Examples" section shows how to use the various clauses and parameters of the statement.

ADMINISTER KEY MANAGEMENT

Purpose

The `ADMINISTER KEY MANAGEMENT` statement provides a unified key management interface for Transparent Data Encryption. Use this statement to:

- Manage software and hardware keystores
- Manage encryption keys
- Manage secrets

For an application PDB, the key management operation can only be performed outside an application action (install, uninstall, upgrade, or patch).

Starting with Oracle Database 23ai, the Transparent Data Encryption (TDE) decryption libraries for the GOST and SEED algorithms are deprecated, and encryption to GOST and SEED are desupported.

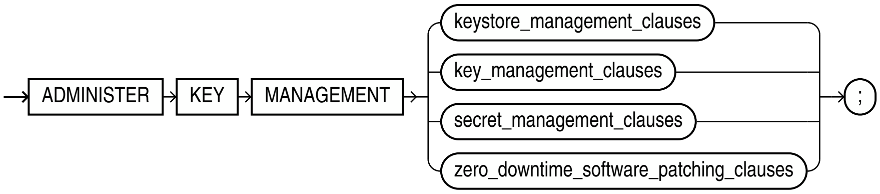
Prerequisites

You must have the `ADMINISTER KEY MANAGEMENT` or `SYSKM` 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 and you must have the commonly granted `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

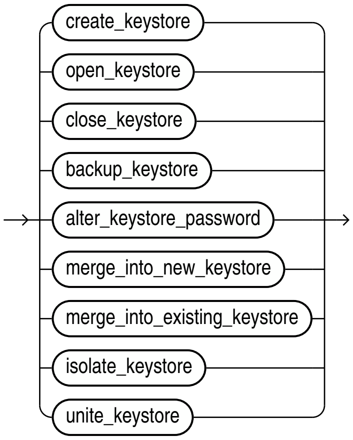
Syntax

***administer_key_management*::=**


ADMINISTER
KEY
MANAGEMENT
keystore_management_clauses
key_management_clauses
secret_management_clauses
zero_downtime_software_patching_clauses
;

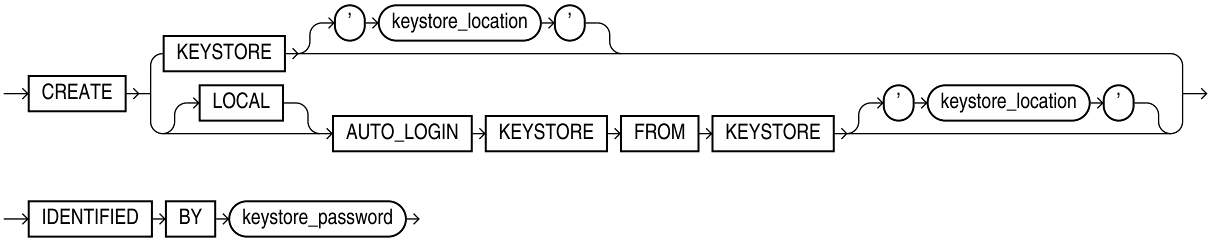
(*keystore_management_clauses*::=, *key_management_clauses*::=, *secret_management_clauses*::=)

***keystore_management_clauses*::=**

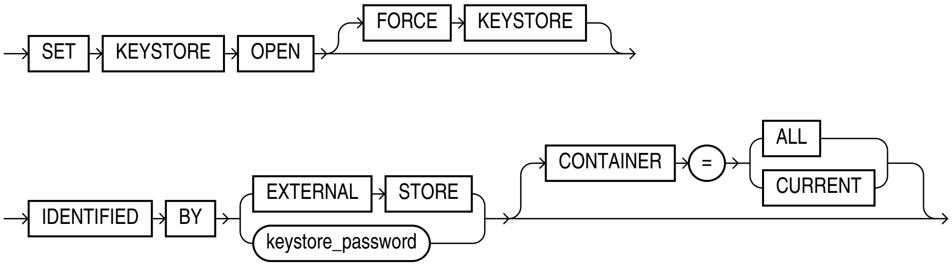


(*create_keystore*::=, *open_keystore*::=, *close_keystore*::=, *backup_keystore*::=, *alter_keystore_password*::=, *merge_into_new_keystore*::=, *merge_into_existing_keystore*::=)

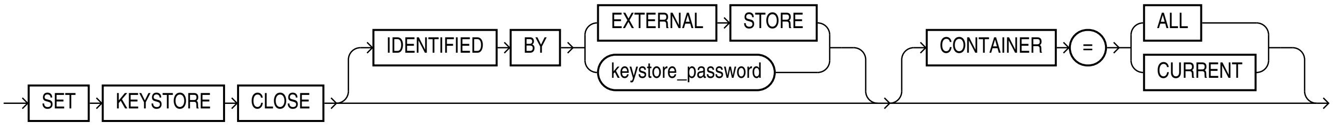
***create_keystore*::=**



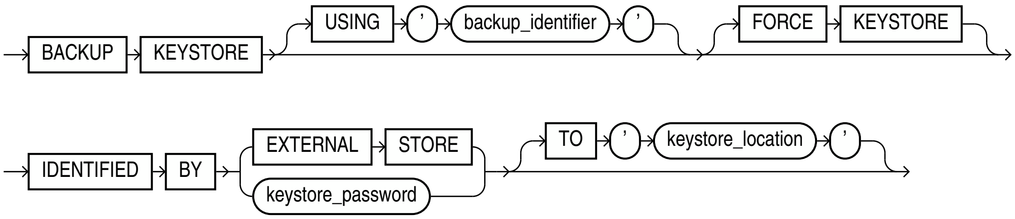
***open_keystore*::=**



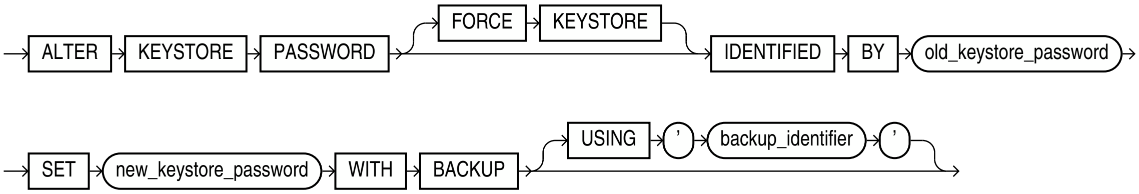
***close_keystore*::=**



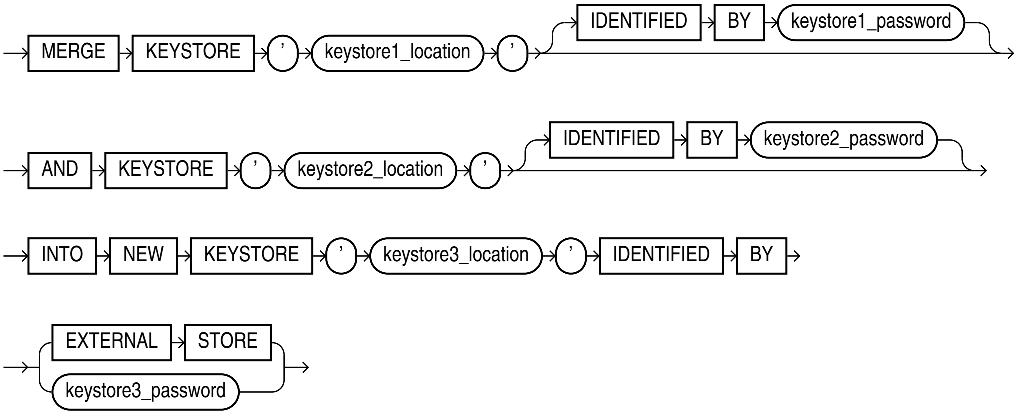
backup_keystore::=



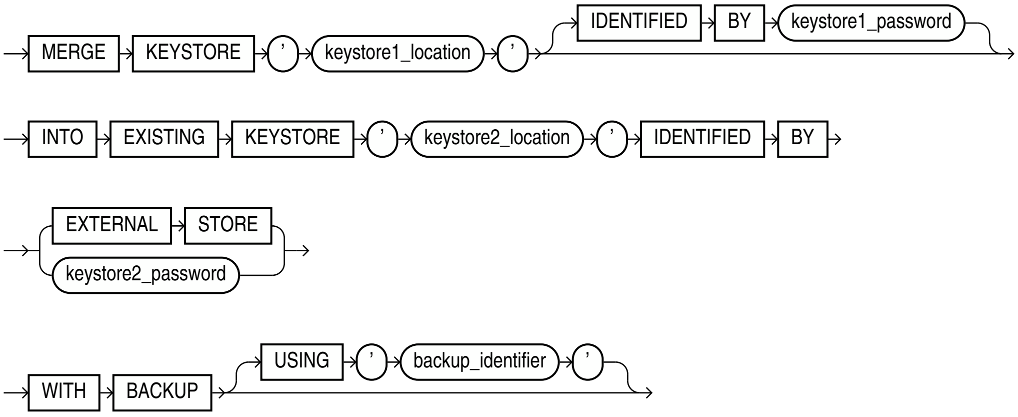
alter_keystore_password::=



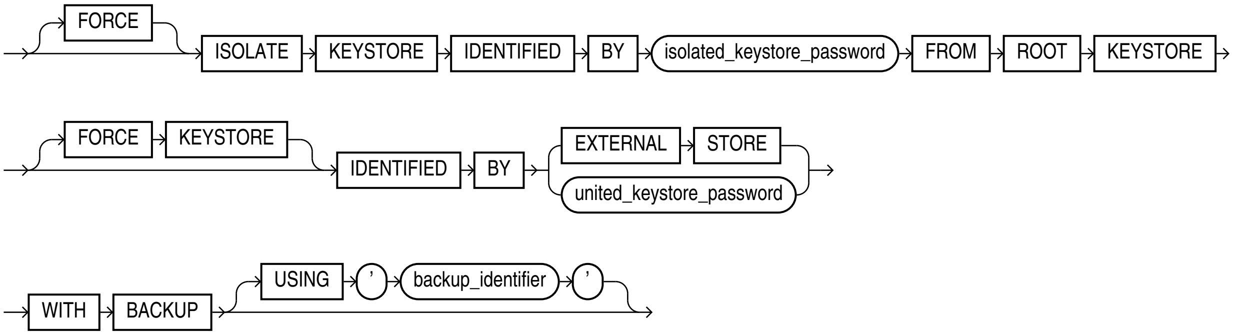
merge_into_new_keystore::=



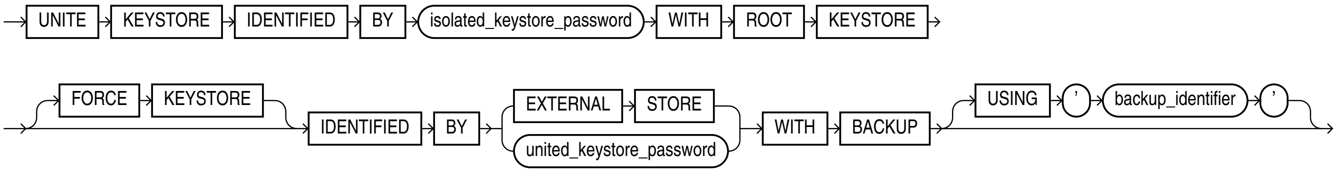
merge_into_existing_keystore::=



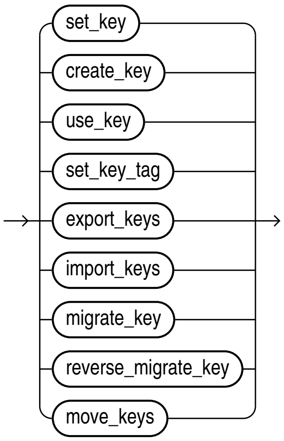
***isolate_keystore*::=**



***unite_keystore* ::=**

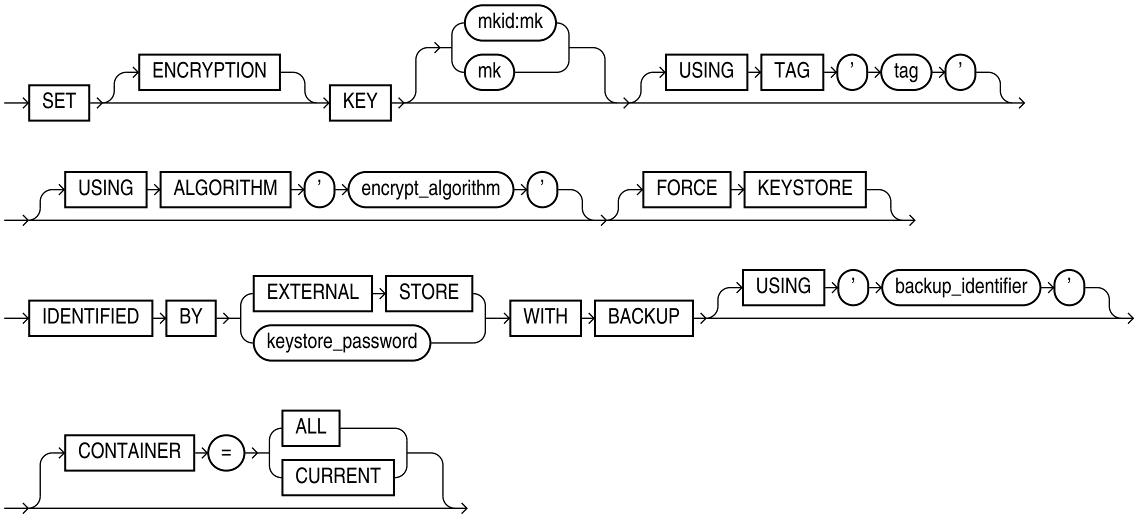


***key_management_clauses*::=**

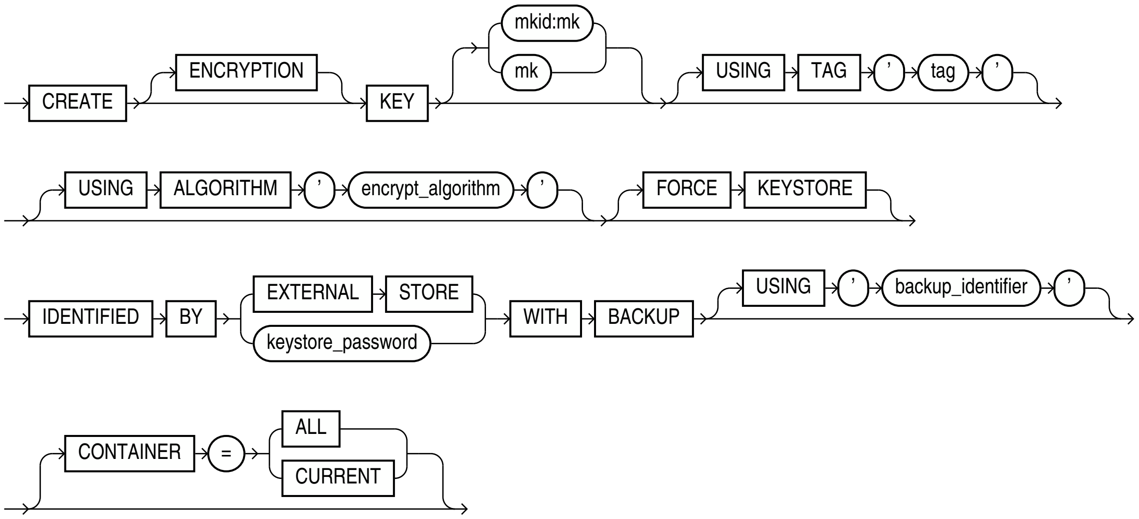


(*set_key*::=, *create_key*::=, *use_key*::=, *set_key_tag*::=, *export_keys*::=, *import_keys*::=, *migrate_key*::=, *reverse_migrate_key*::=)

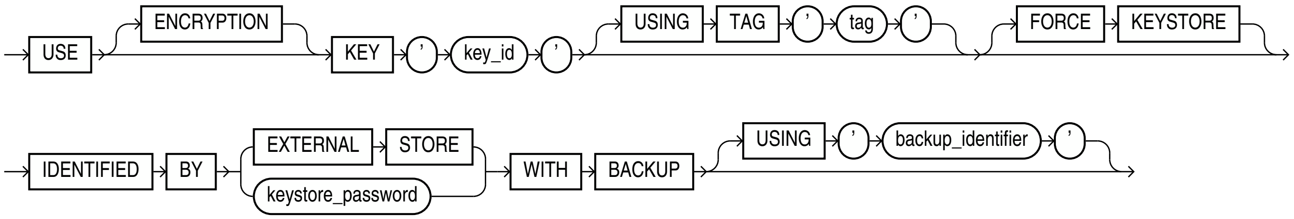
***set_key*::=**



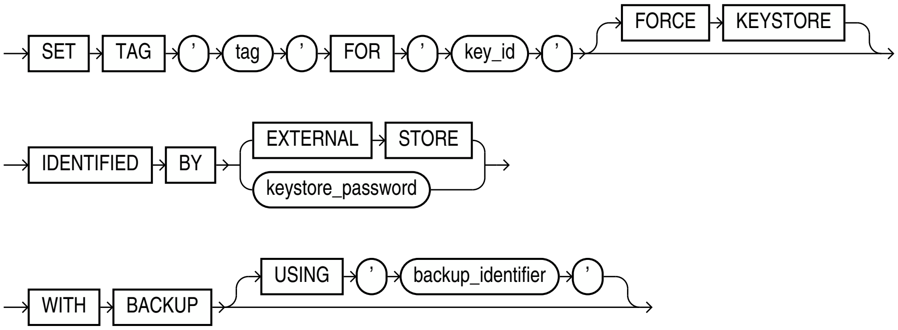
***create_key*::=**



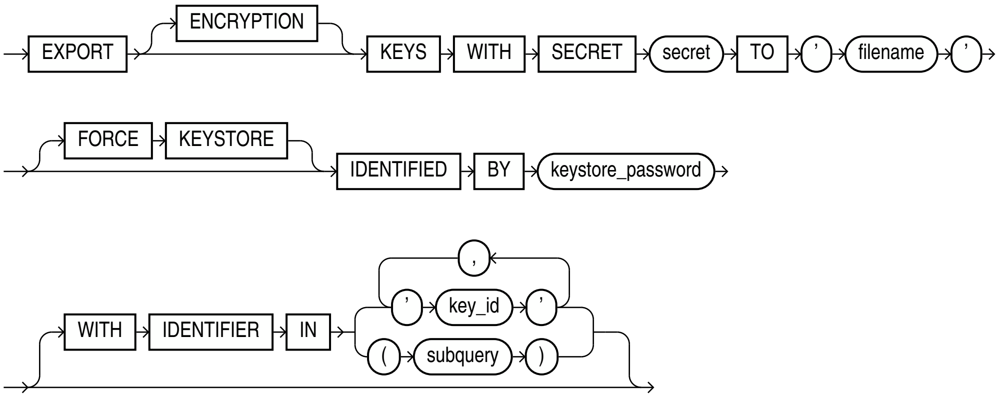
***use_key*::=**



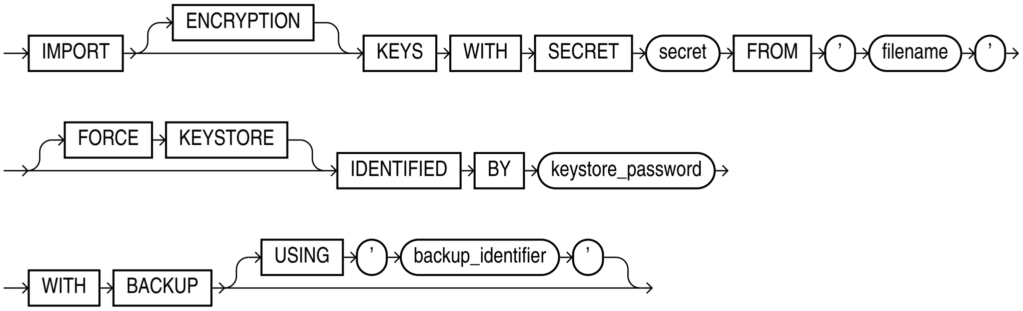
***set_key_tag*::=**



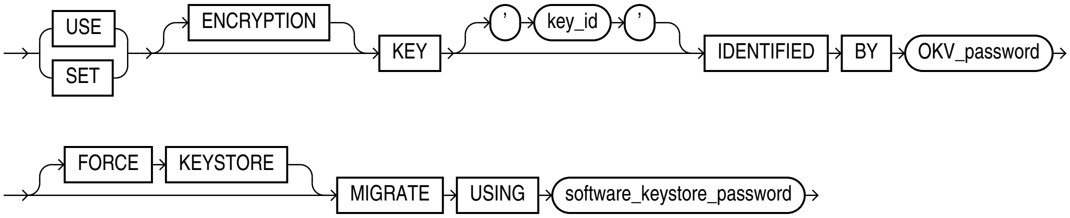
***export_keys*::=**



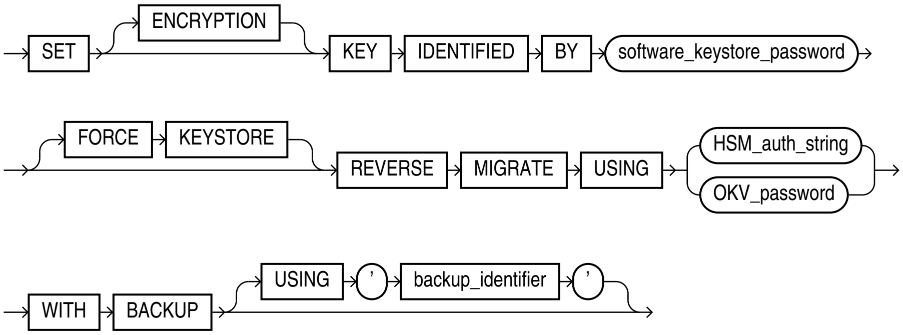
***import_keys*::=**



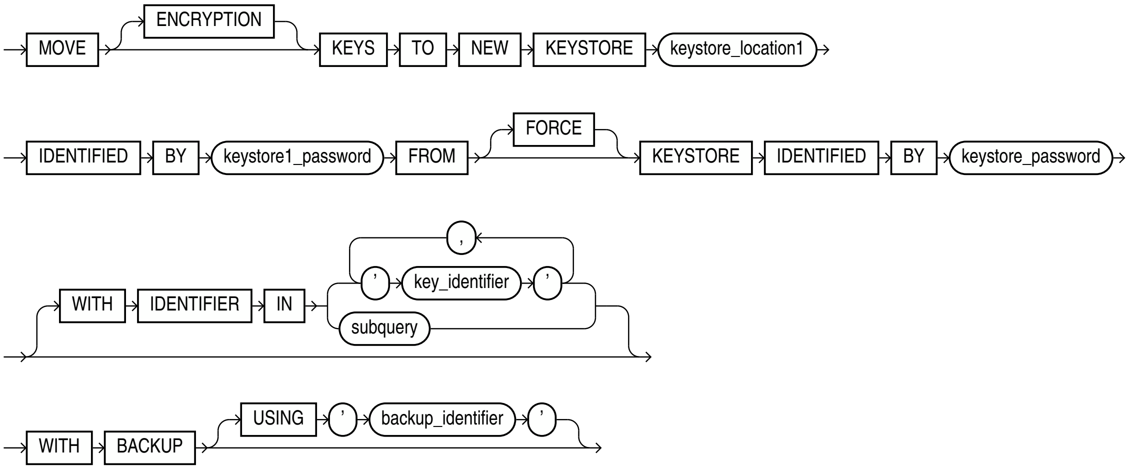
***migrate_key*::=**



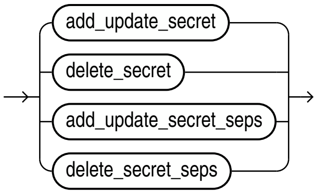
***reverse_migrate_key*::=**



***move_keys* ::=**

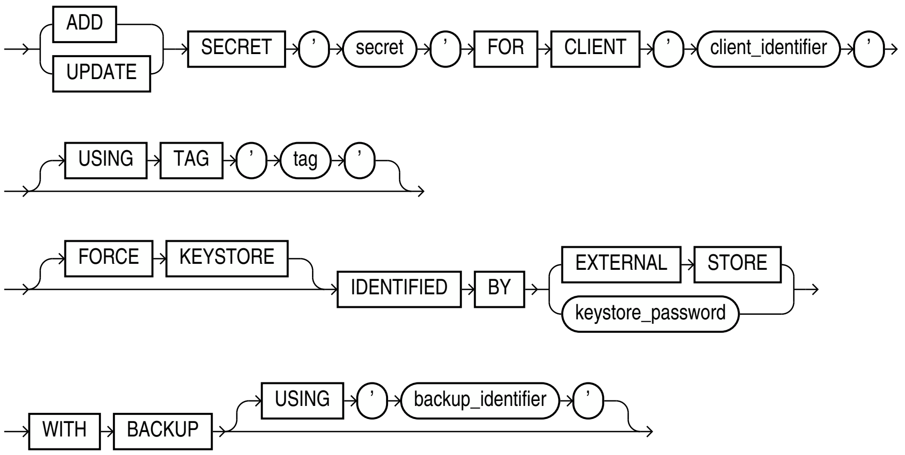


***secret_management_clauses*::=**

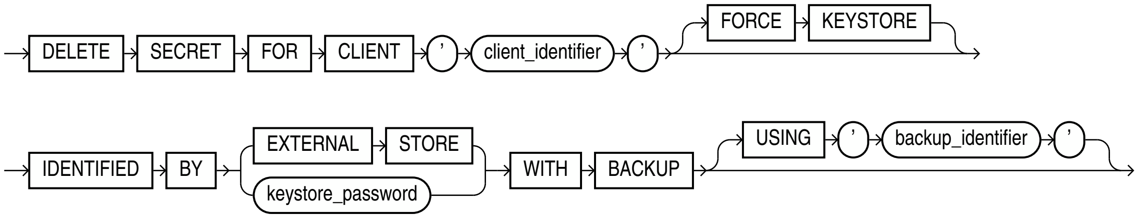


(*add_update_secret*::=, *delete_secret*::=)

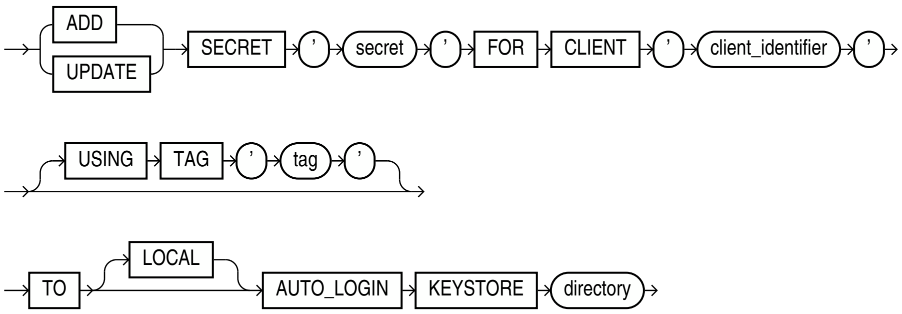
***add_update_secret*::=**



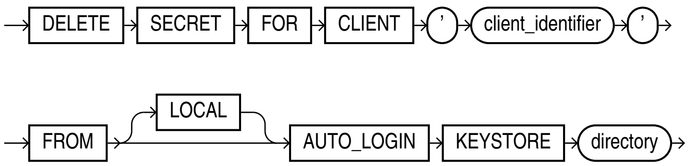
***delete_secret*::=**



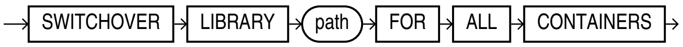
***add_update_secret_seps*::=**



***delete_secret_seps*::=**



***zero_downtime_software_patching_clauses*::=**



Semantics

keystore_management_clauses

Use these clauses to perform the following keystore management operations:

- Create a software keystore
- Open and close a software keystore or a hardware keystore
- Back up a password-protected software keystore
- Change the password of a password-protected software keystore
- Merge two existing software keystores into a new password-protected software keystore
- Merge one existing software keystore into an existing password-protected software keystore
- Isolate the keystore of a Pluggable Database (PDB) from the Container Database (CDB) so that the PDB can manage its own keystore.
- Unite the keystore of a PDB with the CDB.

create_keystore

This clause lets you create the following types of software keystores: password-protected software keystores and auto-login software keystores. To issue this clause in a multitenant environment, you must be connected to the root.

CREATE KEYSTORE

Specify this clause to create a password-protected software keystore.

- For *keystore_location*, specify the full path name of the software keystore directory. The keystore will be created in this directory in a file named `ewallet.p12`. This clause is optional if the `WALLET_ROOT` parameter has been set. Refer to *Transparent Data Encryption* to learn how to determine the software keystore directory for your system.
- Use the `IDENTIFIED BY` clause to set the password for the keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.

CREATE [LOCAL] AUTO_LOGIN KEYSTORE

Specify this clause to create an auto-login software keystore. An auto-login software keystore is created from an existing password-protected software keystore. The auto-login keystore has a system-generated password. It is stored in a PKCS#12-based file named `cwallet.sso` in the same directory as the password-protected software keystore.

- By default, Oracle creates an auto-login keystore, which can be opened from computers other than the computer on which the keystore resides. If you specify the `LOCAL` keyword, then Oracle Database creates a local auto-login keystore, which can be opened only from the computer on which the keystore resides.

- For *keystore_location*, specify the full path name of the directory in which the existing password-protected software keystore resides. The password-protected software keystore can be open or closed.
- Use the IDENTIFIED BY clause to specify the password for the existing password-protected software keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.

Restriction on Creating Keystores

You can create at most one password-protected software keystore and one auto-login software keystore, either local or not, in any single directory.

> **See Also:**
>
> *Transparent Data Encryption* for more information on creating software keystores

open_keystore

This clause lets you open a password-protected software keystore or Oracle Key Vault.

> **Note:**
>
> You do not need to use this clause to open auto-login and local auto-login software keystores because they are opened automatically when they are required—that is, when the master encryption key is accessed.

- The FORCE KEYSTORE clause is useful when opening a keystore in a PDB. It ensures that the CDB root keystore is open before opening the PDB keystore. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the IDENTIFIED BY clause to specify the password for the keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.
- The CONTAINER clause applies when you are connected to a CDB.

  If the current container is a pluggable database (PDB), then specify CONTAINER = CURRENT to open the keystore in the PDB. The keystore must be open in the root before you open it in the PDB.

  If the current container is the root, then specify CONTAINER = CURRENT to open the keystore in the root, or specify CONTAINER = ALL to open the keystore in the root and in all PDBs.

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

See Also:

- *Transparent Data Encryption Managing Keystores and TDE Master Encryption Keys in United Mode*
- *Transparent Data Encryption Managing Keystores and TDE Master Encryption Keys in Isolated Mode*
- *Transparent Data Encryption* for more information on opening password-based software keystores and hardware keystores

close_keystore

This clause lets you close a password-protected software keystore, an auto-login software keystore, or a hardware keystore. Closing a keystore disables all encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

- To close a password-protected software keystore or a hardware keystore, specify the `IDENTIFIED BY` clause. Refer to "Notes on Specifying Keystore Passwords" for more information.
- To close an auto-login keystore, do not specify the `IDENTIFIED BY` clause. Before you close an auto-login keystore, check the `WALLET_TYPE` column of the `V$ENCRYPTION_WALLET` view. If it returns `AUTOLOGIN`, then you can close the keystore. Otherwise, if you attempt to close the keystore, then an error occurs.
- The `CONTAINER` clause applies when you are connected to a CDB.

  If the current container is a PDB, then specify `CONTAINER = CURRENT` to close the keystore in the PDB.

  If the current container is the root, then the `CONTAINER = CURRENT` and `CONTAINER = ALL` clauses have the same effect; both clauses close the keystore in the root and in all PDBs.

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

See Also:

Transparent Data Encryption for more information on closing keystores

backup_keystore

This clause lets you back up a password-protected software keystore. The keystore must be open.

- By default, Oracle Database creates a backup file with a name of the form `ewallet_`*`timestamp`*`.p12`, where *`timestamp`* is the file creation timestamp in UTC format. The optional `USING '`*`backup_identifier`*`'` clause lets you specify a backup identifier which is added to the backup file name. For example, if you specify a backup identifier of `'Backup1'`, then Oracle Database creates a backup file with a name of the form `ewallet_`*`timestamp`*`_Backup1.p12`.
- The `FORCE KEYSTORE` clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.

- Use the `IDENTIFIED BY` clause to specify the password for the keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.
- The optional `TO` '*keystore_location*' clause lets you specify the directory in which the backup file is created. If you omit this clause, then the backup is created in the same directory as the keystore that you are backing up.

> **See Also:**
>
> *Transparent Data Encryption* for more information on backing up password-based software keystores

alter_keystore_password

This clause lets you change the password for a password-protected software keystore. The keystore must be open.

- The `FORCE KEYSTORE` clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- For *old_keystore_password*, specify the old password for the keystore. For *new_keystore_password*, specify the new password for the keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.
- The optional `WITH BACKUP` clause instructs the database to create a backup of the keystore before changing the password. Refer to "Notes on the WITH BACKUP Clause" for more information.

> **See Also:**
>
> *Transparent Data Encryption* for more information on changing a password-based software keystore password

merge_into_new_keystore

This clause lets you merge two software keystores into a new keystore. The keys and attributes in the two constituent keystores are added to the new keystore. The constituent keystores can be password-based or auto-login (including local auto-login) software keystores; they can be open or closed. The new keystore is a password-protected software keystore. It is in a closed state when the merge completes. Any or none of the keystores specified in this clause can be the keystore configured for use by the database.

- For *keystore1_location*, specify the full path name of the directory in which the first keystore resides.
- Specify `IDENTIFIED BY` *keystore1_password* only if the first keystore is a password-based software keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.
- For *keystore2_location*, specify the full path name of the directory in which the second keystore resides.
- Specify `IDENTIFIED BY` *keystore2_password* only if the second keystore is a password-based software keystore.

- For *`keystore3_location`*, specify the full path name of the directory in which the new keystore is created.
- For *`keystore3_password`*, specify the password for the new keystore.

> **See Also:**
>
> *Transparent Data Encryption* for more information on merging software keystores

merge_into_existing_keystore

This clause lets you merge a software keystore into another existing software keystore. The keys and attributes in the keystore from which you merge are added to the keystore into which you merge. The keystore from which you merge can be a password-protected or auto-login (including local auto-login) software keystore; it can be open or closed. The keystore into which you merge must be a password-based software keystore. It can be open or closed when the merge begins. However, it will be in a closed state when the merge completes. Either or neither of the keystores specified in this clause can be the keystore configured for use by the database.

- For *`keystore1_location`*, specify the full path name of the directory in which the keystore from which you merge resides.
- Specify `IDENTIFIED BY` *`keystore1_password`* only if the keystore from which you merge is a password-based software keystore.
- For *`keystore2_location`*, specify the full path name of the directory in which the keystore into which you merge resides.
- For *`keystore2_password`*, specify the password for the keystore into which you merge.
- The optional `WITH BACKUP` clause instructs the database to create a backup of the keystore into which you merge before performing the merge. Refer to "Notes on the WITH BACKUP Clause" for more information.

> **See Also:**
>
> *Transparent Data Encryption* for more information on merging software keystores

isolate_keystore

Pluggable Databases (PDB) within a Container Database (CDB) can create and manage their own keystore. The `isolate_keystore` clause allows a tenant to:

- Manage its Transparent Data Encryption keys independently from those of the CDB.
- Create a password for its independent keystore.

Within the CDB environment you can choose how the keys of a given PDB are protected. PDBs can either protect their keys with an independent password, or use the united password of the CDB.

- Use the `IDENTIFIED BY` clause to specify the password for the keystore.
- The *`isolated_keystore_password`* refers to the independent password of the PDB keystore.

- The FORCE KEYSTORE clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- The *united_keystore_password* refers to the password of the CDB keystore.
- The optional WITH BACKUP clause instructs the database to create a backup of the keystore before changing the password. Refer to "Notes on the WITH BACKUP Clause" for more information.

FORCE Clause with *isolate_keystore*

The FORCE clause of the ADMINISTER KEY MANAGEMENT FORCE ISOLATE KEYSTORE command is used when a clone of the PDB is using the master key being isolated. This command copies the keys from the CDB keystore into the isolated PDB keystore. For example:

```
ADMINISTER KEY MANAGEMENT
FORCE ISOLATE KEYSTORE
IDENTIFIED BY <isolated_keystore_password>
FROM ROOT KEYSTORE
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | <united_keystore_password>]
[WITH BACKUP [USING <backup_identifier>]
```

unite_keystore

The unite_keystore clause allows a PDB that was independently managing its keystore to change its keystore management mode to united. In united mode CDB$ROOT keystore password is used to manage PDBs within the CDB.

- Use the IDENTIFIED BY clause to specify the password for the keystore.
- The *isolated_keystore_password* refers to the independent password of the PDB keystore.
- The FORCE KEYSTORE clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- The *united_keystore_password* refers to the password of the CDB keystore.
- The optional WITH BACKUP clause instructs the database to create a backup of the keystore before changing the password. Refer to "Notes on the WITH BACKUP Clause" for more information.

For example:

```
ADMINISTER KEY MANAGEMENT
UNITE KEYSTORE
IDENTIFIED BY <isolated_keystore_password>
WITH ROOT KEYSTORE
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | <united_keystore_password>]
[WITH BACKUP [USING <backup_identifier>]
```

key_management_clauses

Use these clauses to perform the following key management operations:

- Create and activate a master encryption key
- Set the tag for an encryption key

- Export encryption keys from a keystore into a file
- Import encryption keys from a file into a keystore
- Migrate from a password-protected software keystore to a hardware keystore
- Migrate from a hardware keystore to a password-protected software keystore

set_key

This clause creates a new master encryption key and activates it. You can use this clause to create the first master encryption key in a keystore or to rotate (change) the master encryption key. If a master encryption key is active when you use this clause, then it is deactivated before the new master encryption key is activated. The keystore that contains the key can be a password-protected software keystore or a hardware keystore. The keystore must be open.

If you specify `CONTAINER = ALL`, you must ensure that all the PDBs are open. Otherwise the command fails.

Specify the desired value for your TDE Master Key ID (`MKID`) and desired value of the TDE Master Encryption Key (`MK`) to create your own TDE Master Encryption Key.

If you do not specify `MKID` or `MK`, the default keys used are the system generated `MKID` and `MK`.

- In TDE encrypted databases, the TDE Master Key ID(`MKID`) is used to keep track of which TDE Master Encryption Key is in use. The `MKID:MK` option allows both the `MKID` and the `MK` to be specified.
- If only the `MK` is specified, the database generates a `MKID` for you, so that you can keep track of the TDE Master Encryption Key having the `MK` value that you specified.
- If the `MKID` is invalid, for example if it is the wrong length, or if it is a string of zeroes, you will see the following error: `ORA-46685: invalid master key identifier or master key value.`
- If the `MKID` you specified is the same as the `MKID` of an existing TDE Master Encryption Key in the keystore, you will see the following error: `ORA-46684: master key identifier exists in the keystore.`
- If either the `MKID` or the `MK` is invalid, you will see the following error: `ORA-46685: invalid master key identifier or master key value.`
- You must specify both `MKID:MK` for the `set_key` clause and `create_key` clause.
- For the `use_key` clause, you need to only specify `MKID`.
- The `ENCRYPTION` keyword is optional and is provided for semantic clarity.
- Specify the optional `USING TAG` clause to associate a tag to the new master encryption key. Refer to "Notes on the USING TAG Clause" for more information.
- If you specify the `USING ALGORITHM` clause, then the database creates a master encryption key that conforms to the specified encryption algorithm. For *`encrypt_algorithm`*, you can specify `AES256`, `ARIA256`, `GOST256`, or `SEED128`. To specify this clause, the `COMPATIBLE` initialization parameter must be set to `12.2` or higher. If you omit this clause, then the default is `AES256`.

  The ARIA, SEED, and GOST algorithms are country-specific national and government standards for encryption and hashing. See *Oracle Database Security Guide* for more information.
- The `FORCE KEYSTORE` clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.

- Use the IDENTIFIED BY clause to specify the password for the keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.
- Specify the WITH BACKUP clause, and optionally the USING '*backup_identifier*' clause, to create a backup of the keystore before the new master encryption key is created. Refer to "Notes on the WITH BACKUP Clause" for more information.
- The CONTAINER clause applies when you are connected to a CDB.

  If the current container is a PDB, then specify CONTAINER = CURRENT to create and activate a new master encryption key in the PDB. A master encryption key must exist in the root before you create a master encryption key in the PDB.

  If the current container is the root, then specify CONTAINER = CURRENT to create and activate a new master encryption key in the root, or specify CONTAINER = ALL to create and activate new master encryption keys in the root and in all PDBs.

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

> **See Also:**
>
> - *Transparent Data Encryption Managing Keystores and TDE Master Encryption Keys in United Mode*
> - *Transparent Data Encryption* Managing Keystores and TDE Master Encryption Keys in Isolated Mode
> - *Transparent Data Encryption* for more information on creating and activating a master encryption key

create_key

For details on specifying the MKID:MK option, see the semantics for the set_key clause.

This clause lets you create a master encryption key for later use. You can subsequently activate the key by using the *use_key* clause. The keystore that contains the key can be a password-protected software keystore or a hardware keystore. The keystore must be open.

- The ENCRYPTION keyword is optional and is provided for semantic clarity.
- Specify the optional USING TAG clause to associate a tag to the encryption key. Refer to "Notes on the USING TAG Clause" for more information.
- If you specify the USING ALGORITHM clause, then the database creates a master encryption key that conforms to the specified encryption algorithm. For *encrypt_algorithm*, you can specify AES256, ARIA256, GOST256, or SEED128. To specify this clause, the COMPATIBLE initialization parameter must be set to 12.2 or higher. If you omit this clause, then the default is AES256.

  The ARIA, SEED, and GOST algorithms are country-specific national and government standards for encryption and hashing. See *Oracle Database Security Guide* for more information.
- The FORCE KEYSTORE clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the IDENTIFIED BY clause to specify the password for the keystore in which the key will be created. Refer to "Notes on Specifying Keystore Passwords" for more information.

- Specify the `WITH BACKUP` clause, and optionally the `USING '`*`backup_identifier`*`'` clause, to create a backup of the keystore before the key is created. Refer to "Notes on the WITH BACKUP Clause" for more information.
- The `CONTAINER` clause applies when you are connected to a CDB.

  If the current container is a PDB, then specify `CONTAINER = CURRENT` to create a master encryption key in the PDB. A master encryption key must exist in the root before you create a master encryption key in the PDB

  If the current container is the root, then specify `CONTAINER = CURRENT` to create a master encryption key in the root, or specify `CONTAINER = ALL` to create master encryption keys in the root and in all PDBs.

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

> **See Also:**
>
> *Transparent Data Encryption* for more information on creating a master encryption key for later use

use_key

This clause lets you activate a master encryption key that has already been created. If a master encryption key is active when you use this clause, then it is deactivated before the new master encryption key is activated. The keystore that contains the key can be a password-based software keystore or a hardware keystore. The keystore must be open.

- The `ENCRYPTION` keyword is optional and is provided for semantic clarity.
- For *`key_id`*, specify the identifier of the key that you want to activate. You can find the key identifier by querying the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view.
- Specify the optional `USING TAG` clause to associate a tag to the encryption key. Refer to "Notes on the USING TAG Clause" for more information.
- The `FORCE KEYSTORE` clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the `IDENTIFIED BY` clause to specify the password for the keystore that contains the key. Refer to "Notes on Specifying Keystore Passwords" for more information.
- Specify the `WITH BACKUP` clause, and optionally the `USING '`*`backup_identifier`*`'` clause, to create a backup of the keystore before the key is activated. Refer to "Notes on the WITH BACKUP Clause" for more information.

> **See Also:**
>
> *Transparent Data Encryption* for more information on activating a master encryption key

set_key_tag

This clause lets you set the tag for the specified encryption key. The tag is an optional, user-defined descriptor for the key. If the key has no tag, then use this clause to create a tag. If the key already has a tag, then use this clause to replace the tag. You can view encryption key tags by querying the TAG column of the V$ENCRYPTION_KEYS view. The keystore must be open.

- For *tag*, specify an alphanumeric string. Enclose *tag* in single quotation marks.
- For *key_id*, specify the identifier of the encryption key. You can find the key identifier by querying the KEY_ID column of the V$ENCRYPTION_KEYS view.
- The FORCE KEYSTORE clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the IDENTIFIED BY clause to specify the password for the keystore that contains the key. Refer to "Notes on Specifying Keystore Passwords" for more information.
- Specify the WITH BACKUP clause, and optionally the USING '*backup_identifier*' clause, to create a backup of the keystore before you set the key tag. Refer to "Notes on the WITH BACKUP Clause" for more information.

> **See Also:**
>
> *Transparent Data Encryption* for more information on setting a key tag

export_keys

Use this clause to export one or more encryption keys from a password-protected software keystore into a file. The keystore must be open. Each encryption key is exported together with its key identifier and key attributes. The exported keys are protected in the file with a password (secret). You can subsequently import one or more of the keys into a password-protected software keystore by using the *import_keys* clause.

- The ENCRYPTION keyword is optional and is provided for semantic clarity.
- Specify *secret* to set the password (secret) that protects the keys in the file. The secret is an alphanumeric string. You can optionally enclose the secret in double quotation marks. Quoted and nonquoted secrets are case sensitive.
- For *filename*, specify the full path name of the file to which the keys are to be exported. Enclose *filename* in single quotation marks.
- The FORCE KEYSTORE clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the IDENTIFIED BY clause to specify the password for the keystore that contains the keys you want to export. Refer to "Notes on the WITH BACKUP Clause" for more information.
- Use the WITH IDENTIFIER IN clause to specify one or more encryption keys that you would like to export using one of the following methods:
  - Use *key_id* to specify the identifier of the encryption key you would like to export. You can specify more than one *key_id* in a comma-separated list. You can find key identifiers by querying the KEY_ID column of the V$ENCRYPTION_KEYS view.
  - Use *subquery* to specify a query that returns a list of key identifiers for the encryption keys you would like to export. For example, the following *subquery* returns the key identifiers for all encryption keys in the database whose tags begin with the string mytag:

    ```
    SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE TAG LIKE 'mytag%'
    ```

Be aware that Oracle Database executes *subquery* within the current user's rights and not with definer's rights.

- If you omit the WITH IDENTIFIER IN clause, then all encryption keys in the database are exported.

Restriction on the WITH IDENTIFIER IN Clause

In a multitenant environment, you cannot specify WITH IDENTIFIER IN when exporting keys from a PDB. This ensures that all of the keys in the PDB are exported, along with metadata about the active encryption key. If you subsequently clone the PDB, or unplug and plug in the PDB, then you can use the export file to import the keys into the cloned or newly plugged-in PDB and preserve information about the active encryption key.

Note, that the keystores on Automatic Storage Management (ASM) disk groups or regular file systems can be merged with MERGE statements. The export files used in the EXPORT and the IMPORT statements can only be a regular operating system file and cannot be located on an ASM disk group.

ADMINISTER KEY MANAGEMENT *export_keys* and *import_keys* do not support wallet files in ASM.

> **See Also:**
>
> *Transparent Data Encryption* for more information on exporting encryption keys

import_keys

Use this clause to import one or more encryption keys from a file into a password-based software keystore. The keystore must be open. Each encryption key is imported together with its key identifier and key attributes. The keys must have been previously exported to the file by using the *export_keys* clause. You cannot re-import keys that have already been imported into the keystore.

- The ENCRYPTION keyword is optional and is provided for semantic clarity.
- For *secret*, specify the password (secret) that protects the keys in the file. The secret is an alphanumeric string. You can optionally enclose the secret in double quotation marks. Quoted and nonquoted secrets are case sensitive.
- For *filename*, specify the full path name of the file from which the keys are to be imported. Enclose *filename* in single quotation marks.
- The FORCE KEYSTORE clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the IDENTIFIED BY clause to specify the password for the keystore into which you want to import the keys. Refer to "Notes on the WITH BACKUP Clause" for more information.
- Specify the WITH BACKUP clause, and optionally the USING '*backup_identifier*' clause, to create a backup of the keystore before the keys are imported. Refer to "Notes on the WITH BACKUP Clause" for more information.

Note, that the keystores on Automatic Storage Management (ASM) disk groups or regular file systems can be merged with MERGE statements. The export files used in the EXPORT and the IMPORT statements can only be a regular operating system file and cannot be located on an ASM disk group.

ADMINISTER KEY MANAGEMENT *export_keys* and *import_keys* do not support wallet files in ASM.

> **See Also:**
>
> *Transparent Data Encryption* for more information on importing encryption keys

migrate_key

Use this clause to migrate from a password-protected software keystore to a hardware keystore. This clause decrypts existing table encryption keys and tablespace encryption keys with the master encryption key in the software keystore and then re-encrypts them with the newly created master encryption key in the hardware keystore.

You can use use_key with migrate_key to migrate an exisiting key to a hardware keystore.

You must specify the key_id with use_key as follows:

```
ADMINISTER KEY MANAGEMENT
  USE ENCRYPTION KEY '0673C1262AA1D04F14BF26D720480C55B2'
  IDENTIFIED BY "external_keystore_password"
  MIGRATE USING software_keystore_password;
```

> **Note:**
>
> The use of this clause is only one step in a series of steps for migrating from a password-protected software keystore to a hardware keystore. Refer to *Transparent Data Encryption* for the complete set of steps before you use this clause.

- The ENCRYPTION keyword is optional and is provided for semantic clarity.
- For *HSM_auth_string*, specify the hardware keystore password. Refer to "Notes on Specifying Keystore Passwords" for more information.
- The FORCE KEYSTORE clause enables this operation even if the keystores are closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- For *software_keystore_password*., specify the password-based software keystore password. Refer to "Notes on Specifying Keystore Passwords" for more information.
- Specify the WITH BACKUP clause, and optionally the USING '*backup_identifier*' clause, to create a backup of the keystore before the migration occurs. Refer to "Notes on the WITH BACKUP Clause" for more information.

reverse_migrate_key

Use this clause to migrate from a hardware keystore to a password-protected software keystore. This clause decrypts existing table encryption keys and tablespace encryption keys with the master encryption key in the hardware keystore and then re-encrypts them with the newly created master encryption key in the password-protected software keystore.

> **Note:**
>
> The use of this clause is only one step in a series of steps for migrating from a hardware keystore to a password-protected software keystore. Refer to *Transparent Data Encryption* for the complete set of steps before you use this clause.

- The `ENCRYPTION` keyword is optional and is provided for semantic clarity.
- For *`software_keystore_password`*., specify the password-based software keystore password. Refer to "Notes on Specifying Keystore Passwords" for more information.
- The `FORCE KEYSTORE` clause enables this operation even if the keystores are closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- For *`HSM_auth_string`*, specify the hardware keystore password. Refer to "Notes on Specifying Keystore Passwords" for more information.

move_keys

Use the `move_keys` clause to move an encryption key into a new keystore. You must be a user with the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privileges to log into the database. You must query the `KEY_ID`column of the `V$ENCRYPTION_KEYS` view to find the key identifier of the keystore that you want to move the keys to.

`keystore_location1` is the path to the wallet directory that will store the new keystore `.p12` file. By default, this directory is in `$ORACLE_BASE/admin/db_unique_name/wallet`.

`keystore1_password` is the password for the new keystore.

`keystore_password` is the password for the keystore from which the key is moving.

`key_identifier` is the key identifier that you find from querying the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view. Enclose this setting in single quotation marks (' ').

`subquery` can be used to find the exact key identifier that you want.

`backup_identifier` is an optional description of the backup. Enclose `backup_identifier` in single quotation marks (' ').

For example:

```
ADMINISTER KEY MANAGEMENT MOVE KEYS
TO NEW KEYSTORE $ORACLE_BASE/admin/orcl/wallet
IDENTIFIED BY keystore_password
FROM FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH IDENTIFIER IN
(SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM < 2);
```

secret_management_clauses

Use these clauses to add, update, and delete secrets in password-protected software keystores or hardware keystores.

> **See Also:**
>
> *Transparent Data Encryption* for more information on adding, updating, and deleting secrets

add_update_secret

This clause lets you add a secret to a keystore or update an existing secret in a keystore. The keystore must be open.

- Specify `ADD` to add a secret to a keystore.
- Specify `UPDATE` to update an existing secret in a keystore.
- For *`secret`*, specify the secret to be added or updated. The secret is an alphanumeric string. Enclose the secret in single quotation marks.
- For *`client_identifier`*, specify an alphanumeric string used to identify the secret. Enclose *`client_identifier`* in single quotation marks. This value is case-sensitive. You can enter any of the following fixed values:
  - `TDE_WALLET` if the keystore was configured as FILE
  - `OKV_PASSWORD` if the keystore was configured as Oracle Key Vault.
  - `HSM_PASSWORD` if the keystore was configured for a third-party HSM
- Specify the optional `USING TAG` clause to associate a tag to *`secret`*. The *`tag`* is an optional, user-defined descriptor for the secret. Enclose the tag in single quotation marks. You can view secret tags by querying the `SECRET_TAG` column of the `V$CLIENT_SECRETS` view.
- The `FORCE KEYSTORE` clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the `IDENTIFIED BY` clause to specify the password for the keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.
- Specify the `WITH BACKUP` clause, and optionally the `USING '`*`backup_identifier`*`'` clause, to create a backup of the keystore before adding or updating the secret in a password-based software keystore. Refer to "Notes on the WITH BACKUP Clause" for more information.

delete_secret

This clause lets you delete a secret from a keystore. The keystore must be open.

- For *`client_identifier`*, specify an alphanumeric string used to identify the secret. Enclose *`client_identifier`* in single quotation marks. You can view client identifiers by querying the `CLIENT` column of the `V$CLIENT_SECRETS` view.
- The `FORCE KEYSTORE` clause enables this operation even if the keystore is closed. Refer to "Notes on the FORCE KEYSTORE Clause" for more information.
- Use the `IDENTIFIED BY` clause to specify the password for the keystore. Refer to "Notes on Specifying Keystore Passwords" for more information.
- Specify the `WITH BACKUP` clause, and optionally the `USING '`*`backup_identifier`*`'` clause, to create a backup of the keystore before deleting the secret from a password-based software keystore. Refer to "Notes on the WITH BACKUP Clause" for more information.

Notes on the USING TAG Clause

Many ADMINISTER KEY MANAGEMENT operations include the USING TAG clause, which lets you associate a tag to an encryption key. The *tag* is an optional, user-defined descriptor for the key. It is a character string enclosed in single quotation marks.

You can view encryption key tags by querying the TAG column of the V$ENCRYPTION_KEYS view.

Notes on the FORCE KEYSTORE Clause

When a auto-login wallet exists, the FORCE KEYSTORE clause enables a keystore operation even if the keystore is closed.. The behavior of this clause depends on whether you are connected to a non-CDB, a CDB root, or a PDB.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

- When you are connected to a non-CDB:
  - If the password-protected software or hardware keystore is closed, then the database opens the password-protected software or hardware keystore while the operation is performed and leaves it open, and then updates the auto-login keystore, if one exists, with the new information.
  - If the auto-login keystore is open, then the database opens the password-protected software or hardware keystore temporarily while the operation is performed and updates the auto-login keystore with the new information, without switching out the auto-login keystore.
  - If the password-protected software or hardware keystore is open, then the FORCE KEYSTORE clause is not necessary and has no effect.
- When you are connected to the CDB root:
  - To perform an operation on the CDB root keystore (CONTAINER=CURRENT), the CDB root keystore must be open. Therefore, the behavior described for a non-CDB applies to the CDB root.
  - To perform an operation on the CDB root keystore and all PDB keystores (CONTAINER=ALL), the CDB root keystore and all PDB keystores must be open. Therefore, the behavior described for a non-CDB applies to the CDB root and each PDB.
- When you are connected to a PDB:
  - To perform an operation on a PDB keystore, the CDB root keystore and the keystore for that PDB must be open. Therefore, the behavior described for a non-CDB applies to the CDB root and that PDB.

Notes on Specifying Keystore Passwords

Specify keystore passwords as follows:

- For a password-protected software keystore, specify the password as a character string. You can optionally enclose the password in double quotation marks. Quoted and nonquoted passwords are case sensitive. Keystore passwords adhere to the same rules as database user passwords. Refer to the BY *password* clause of CREATE USER for the complete details.

- For a hardware keystore, specify the password as a string of the form "*user_id*:*password*" where:
  - *user_id* is the user ID created for the database using the HSM management interface
  - *password* is the password created for the user ID using the HSM management interface

  Enclose the *user_id*:*password* string in double quotation marks (" ") and separate *user_id* and *password* with a colon (:).
- If you specify EXTERNAL STORE, then the database uses the keystore password stored in the external store to perform the operation. This feature enables you to store the password in a separate location where it can be centrally managed and accessed. To use this functionality, you must create a directory WALLET_ROOT/tde_seps for the database to auto-discover this wallet.Refer to *Database Transparent Data Encryption* for more information on configuring an external store for a keystore password.

Notes on the WITH BACKUP Clause

Many ADMINISTER KEY MANAGEMENT operations include the WITH BACKUP clause. This clause applies only to password-protected software keystores. It indicates that the keystore must be backed up before the operation is performed.

You must either specify WITH BACKUP when performing the operation, or issue ADMINISTER KEY MANAGEMENT with WITH BACKUP immediately *before* performing the operation.

You can also back up the auto-login wallet using WITH BACKUP.

When you specify the WITH BACKUP clause, Oracle Database creates a backup file with a name of the form ewallet_*timestamp*.p12, where *timestamp* is the file creation timestamp in UTC format. The backup file is created in the same directory as the keystore you are backing up.

The optional USING '*backup_identifier*' clause lets you specify a backup identifier, which is added to the backup file name. For example, if you specify a backup identifier of 'Backup1', then Oracle Database creates a backup file with a name of the form ewallet_*timestamp*_Backup1.p12.

The WITH BACKUP clause is mandatory for password-protected software keystores, but optional for hardware keystores.

add_update_secret_seps

Specify this clause to manage keys in a secure external password store (SEPS) also known as a SEPS wallet. The semantics of this clause is the same as the *add_update_secret* clause.

delete_secret_seps

Specify this clause to delete keys in a secure external password store (SEPS) also known as a SEPS wallet. The semantics of this clause is the same as the *delete_secret* clause.

zero_downtime_software_patching_clauses

Specify this clause to switch over to a new PKCS#11 endpoint library. Afterward, you can switch over to the updated PKCS#11 endpoint shared library by executing the following statement:

```
ADMINISTER KEY MANAGEMENT SWITCHOVER TO LIBRARY
'updated_fully_qualified_file_name_of_library' FOR ALL CONTAINERS
```

> **See Also:**
>
> Managing Updates to the PKCS#11 Library

Examples

Creating a Keystore: Examples

The following statement creates a password-protected software keystore in directory `/etc/ORACLE/WALLETS/orcl`:

```
ADMINISTER KEY MANAGEMENT
  CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl'
  IDENTIFIED BY password;
```

The following statement creates an auto-login software keystore from the keystore created in the previous statement:

```
ADMINISTER KEY MANAGEMENT
  CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE '/etc/ORACLE/WALLETS/orcl'
  IDENTIFIED BY password;
```

Opening a Keystore: Examples

The following statement opens a password-protected software keystore:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE OPEN
  IDENTIFIED BY password;
```

If you are connected to a CDB, then the following statement opens a password-protected software keystore in the current container:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE OPEN
  IDENTIFIED BY password
  CONTAINER = CURRENT;
```

The following statement opens a hardware keystore:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE OPEN
  IDENTIFIED BY "user_id:password";
```

The following statement opens a keystore whose password is stored in the external store:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE OPEN
  IDENTIFIED BY EXTERNAL STORE;
```

Closing a Keystore: Examples

The following statement closes a password-protected software keystore:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE CLOSE
  IDENTIFIED BY password;
```

The following statement closes an auto-login software keystore:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE CLOSE;
```

The following statement closes a hardware keystore:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE CLOSE
  IDENTIFIED BY "user_id:password";
```

The following statement closes a keystore whose password is stored in the external store:

```
ADMINISTER KEY MANAGEMENT
  SET KEYSTORE CLOSE
  IDENTIFIED BY EXTERNAL STORE;
```

Backing Up a Keystore: Example

The following statement creates a backup of a password-protected software keystore. The backup is stored in directory `/etc/ORACLE/KEYSTORE/DB1` and the backup file name contains the tag `hr.emp_keystore`.

```
ADMINISTER KEY MANAGEMENT
  BACKUP KEYSTORE USING 'hr.emp_keystore'
  IDENTIFIED BY password
  TO '/etc/ORACLE/KEYSTORE/DB1/';
```

Changing a Keystore Password: Example

The following statement changes the password for a password-protected software keystore. It also creates a backup of the keystore, with the tag `pwd_change`, before changing the password.

```
ADMINISTER KEY MANAGEMENT
  ALTER KEYSTORE PASSWORD IDENTIFIED BY old_password
  SET new_password WITH BACKUP USING 'pwd_change';
```

Merging Two Keystores Into a New Keystore: Example

The following statement merges an auto-login software keystore with a password-protected software keystore to create a new password-protected software keystore at a new location:

```
ADMINISTER KEY MANAGEMENT
  MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
  AND KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
    IDENTIFIED BY existing_keystore_password
  INTO NEW KEYSTORE '/etc/ORACLE/KEYSTORE/DB3'
    IDENTIFIED BY new_keystore_password;
```

Merging a Keystore Into an Existing Keystore: Example

The following statement merges an auto-login software keystore into a password-protected software keystore. It also creates a backup of the password-protected software keystore before performing the merge.

```
ADMINISTER KEY MANAGEMENT
  MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
  INTO EXISTING KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
    IDENTIFIED BY existing_keystore_password
  WITH BACKUP;
```

Creating and Activating a Master Encryption Key: Examples

The following statement creates and activates a master encryption key in a password-protected software keystore. It encrypts the key using the SEED128 algorithm. It also creates a backup of the keystore before creating the new master encryption key.

```
ADMINISTER KEY MANAGEMENT
  SET KEY USING ALGORITHM 'SEED128'
  IDENTIFIED BY password
  WITH BACKUP;
```

The following statement creates a master encryption key in a password-protected software keystore, but does not activate the key. It also creates a backup of the keystore before creating the new master encryption key.

```
ADMINISTER KEY MANAGEMENT
  CREATE KEY USING TAG 'mykey1'
  IDENTIFIED BY password
  WITH BACKUP;
```

The following query displays the key identifier for the master encryption key that was created in the previous statement:

```
SELECT TAG, KEY_ID
  FROM V$ENCRYPTION_KEYS
  WHERE TAG = 'mykey1';

TAG     KEY_ID
---     ----------------------------------------------------
mykey1  ARgEtzPxpE/Nv8WdPu8LJJUAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

The following statement activates the master encryption key that was queried in the previous statement. It also creates a backup of the keystore before activating the new master encryption key.

```
ADMINISTER KEY MANAGEMENT
  USE KEY 'ARgEtzPxpE/Nv8WdPu8LJJUAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
  IDENTIFIED BY password
  WITH BACKUP;
```

Setting a Key Tag: Example

This example assumes that the keystore is closed. The following statement temporarily opens the keystore and changes the tag to mykey2 for the master encryption key that was activated in the previous example. It also creates a backup of the keystore before changing the tag.

```
ADMINISTER KEY MANAGEMENT
  SET TAG 'mykey2' FOR 'ARgEtzPxpE/Nv8WdPu8LJJUAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
  FORCE KEYSTORE
  IDENTIFIED BY password
  WITH BACKUP;
```

Exporting Keys: Examples

The following statement exports two master encryption keys from a password-protected software keystore to file /etc/TDE/export.exp. The statement encrypts the master encryption keys in the file using the secret my_secret. The identifiers of the master encryption keys to be exported are provided as a comma-separated list.

```
ADMINISTER KEY MANAGEMENT
  EXPORT KEYS WITH SECRET "my_secret"
  TO '/etc/TDE/export.exp'
  IDENTIFIED BY password
  WITH IDENTIFIER IN 'AdoxnJ0uH08cv7xkz83ovwsAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',
                     'AW5z3CoyKE/yv3cNT5CWCXUAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
```

The following statement exports master encryption keys from a password-protected software keystore to file `/etc/TDE/export.exp`. Only the keys whose tags are `mytag1` or `mytag2` are exported. The master encryption keys in the file are encrypted using the secret `my_secret`. The key identifiers are found by querying the `V$ENCRYPTION_KEYS` view.

```
ADMINISTER KEY MANAGEMENT
  EXPORT KEYS WITH SECRET "my_secret"
  TO '/etc/TDE/export.exp'
  IDENTIFIED BY password
  WITH IDENTIFIER IN
    (SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE TAG IN ('mytag1', 'mytag2'));
```

The following statement exports all master encryption keys of the database to file `/etc/TDE/export.exp`. The master encryption keys in the file are encrypted using the secret `my_secret`.

```
ADMINISTER KEY MANAGEMENT
  EXPORT KEYS WITH SECRET "my_secret"
  TO '/etc/TDE/export.exp'
  IDENTIFIED BY password;
```

In a multitenant environment, the following statements exports all master encryption keys of the PDB `salespdb`, along with metadata, to file `/etc/TDE/salespdb.exp`. The master encryption keys in the file are encrypted using the secret `my_secret`. If the PDB is subsequently cloned, or unplugged and plugged back in, then the export file created by this statement can be used to import the keys into the cloned or newly plugged-in PDB.

```
ALTER SESSION SET CONTAINER = salespdb;
ADMINISTER KEY MANAGEMENT
  EXPORT KEYS WITH SECRET "my_secret"
  TO '/etc/TDE/salespdb.exp'
  IDENTIFIED BY password;
```

Importing Keys: Example

The following statement imports the master encryption keys, encrypted with secret `my_secret`, from file `/etc/TDE/export.exp` to a password-protected software keystore. It also creates a backup of the password-protected software keystore before importing the keys.

```
ADMINISTER KEY MANAGEMENT
  IMPORT KEYS WITH SECRET "my_secret"
  FROM '/etc/TDE/export.exp'
  IDENTIFIED BY password
  WITH BACKUP;
```

Migrating a Keystore: Example

The following statement migrates from a password-protected software keystore to a hardware keystore. It also creates a backup of the password-protected software keystore before performing the migration.

```
ADMINISTER KEY MANAGEMENT
  SET ENCRYPTION KEY IDENTIFIED BY "user_id:password"
  MIGRATE USING software_keystore_password
  WITH BACKUP;
```

Reverse Migrating a Keystore: Example

The following statement reverse migrates from a hardware keystore to a password-protected software keystore:

```
ADMINISTER KEY MANAGEMENT
  SET ENCRYPTION KEY IDENTIFIED BY software_keystore_password
  REVERSE MIGRATE USING "user_id:password";
```

Adding a Secret to a Keystore: Examples

The following statement adds secret `secret1`, with the tag `My first secret`, for client `client1` to a password-protected software keystore. It also creates a backup of the password-protected software keystore before adding the secret.

```
ADMINISTER KEY MANAGEMENT
  ADD SECRET 'secret1' FOR CLIENT 'client1'
  USING TAG 'My first secret'
  IDENTIFIED BY password
  WITH BACKUP;
```

The following statement adds a similar secret to a hardware keystore:

```
ADMINISTER KEY MANAGEMENT
  ADD SECRET 'secret2' FOR CLIENT 'client2'
  USING TAG 'My second secret'
  IDENTIFIED BY "user_id:password";
```

Updating a Secret in a Keystore: Examples

The following statement updates the secret that was created in the previous example in a password-based software keystore. It also creates a backup of the password-protected software keystore before updating the secret.

```
ADMINISTER KEY MANAGEMENT
  UPDATE SECRET 'secret1' FOR CLIENT 'client1'
  USING TAG 'New Tag 1'
  IDENTIFIED BY password
  WITH BACKUP;
```

The following statement updates the secret that was created in the previous example in a hardware keystore:

```
ADMINISTER KEY MANAGEMENT
  UPDATE SECRET 'secret2' FOR CLIENT 'client2'
  USING TAG 'New Tag 2'
  IDENTIFIED BY "user_id:password";
```

Deleting a Secret from a Keystore: Examples

The following statement deletes the secret that was updated in the previous example from a password-protected software keystore. It also creates a backup of the password-protected software keystore before deleting the secret.

```
ADMINISTER KEY MANAGEMENT
  DELETE SECRET FOR CLIENT 'client1'
  IDENTIFIED BY password
  WITH BACKUP;
```

The following statement deletes the secret that was updated in the previous example from a hardware keystore:

```
ADMINISTER KEY MANAGEMENT
  DELETE SECRET FOR CLIENT 'client2'
  IDENTIFIED BY "user_id:password";
```

ALTER ANALYTIC VIEW

Purpose

Use the ALTER ANALYTIC VIEW statement to rename or compile an analytic view. Additionally, you can modify grouping level caches by adding or dropping a new level grouping cache to a specifed analytic view.

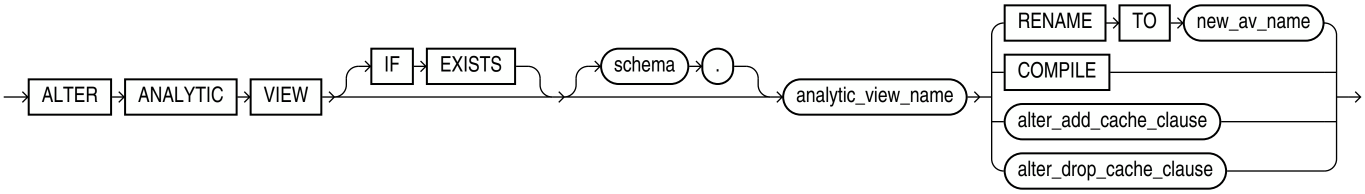
For other alterations, use CREATE OR REPLACE ANALYTIC VIEW.

Prerequisites

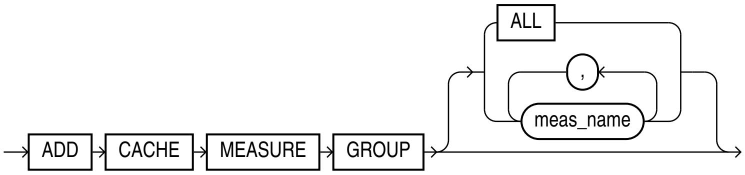
To alter an analytic view in your own schema, you must have the ALTER ANALYTIC VIEW system privilege. To alter an analytic view in another user's schema, you must have the ALTER ANY ANALYTIC VIEW system privilege or ALTER ANY TABLE granted on the analytic view.

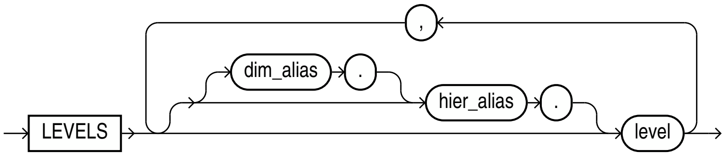
Syntax

***alter_analytic_view*::=**

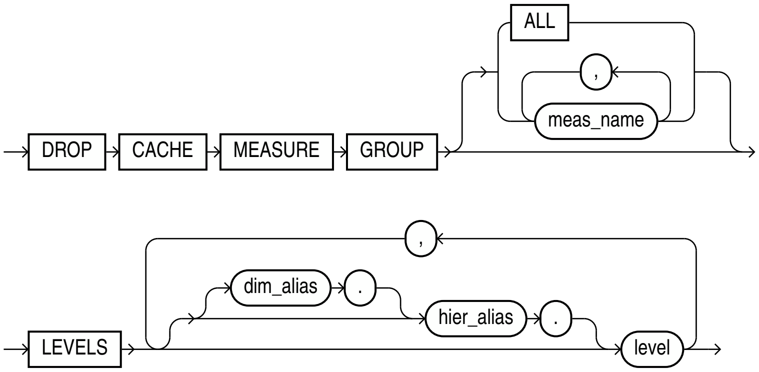


***alter_add_cache_clause*::=**





***alter_drop_cache_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 in which the analytic view exists. If you do not specify a schema, then Oracle Database looks for the analytic view in your own schema.

analytic_view_name

Specify the name of the analytic view.

RENAME TO

Specify `RENAME TO` to change the name of the analytic view. For *`new_av_name`*, specify a new name for the analytic view.

COMPILE

Specify `COMPILE` to compile the analytic view.

alter_add_cache_clause

Use this clause to add a new level grouping cache to a specified analytic view like the measure group, level clause and the cache type. Before you add a new level grouping cache, you must ensure that it does not match a previously defined cache with the same measures and levels.

alter_drop_cache_clause

Use this clause to drop an existent level grouping cache from an analytic view. You must specify the attributes of the level grouping you are about to drop, like the measure group and the level clause.

Example: Change the Name of an Analytic View

```
ALTER ANALYTIC VIEW sales_av RENAME TO mysales_av;
```

Example: Add a New Level Grouping Cache to an Analytic View

```
ALTER ANALYTIC VIEW TKHCSGL308_UNITS_AVIEW_CACHE ADD CACHE
    MEASURE GROUP (sales, units, cost)
    LEVELS (TIME.FISCAL.FISCAL_QUARTER, WAREHOUSE);
```

ALTER ATTRIBUTE DIMENSION

Purpose

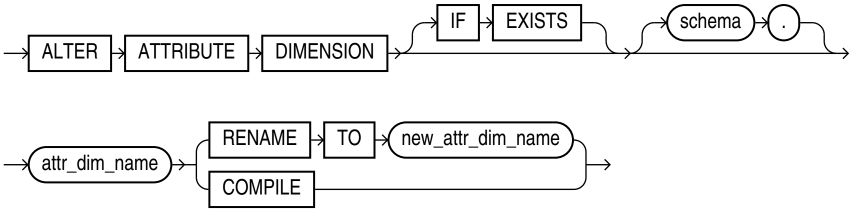
Use the `ALTER ATTRIBUTE DIMENSION` statement to rename or compile an attribute dimension. For other alterations, use `CREATE OR REPLACE ATTRIBUTE DIMENSION`.

Prerequisites

To alter an attribute dimension in your own schema, you must have the `ALTER ATTRIBUTE DIMENSION` system privilege. To alter an attribute dimension in another user's schema, you must have the `ALTER ANY ATTRIBUTE DIMENSION` system privilege or have been granted `ALTER` on the attribute dimension directly.

Syntax

***alter_attribute_dimension*::=**



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 in which the attribute dimension exists. If you do not specify a schema, then Oracle Database looks for the attribute dimension in your own schema.

attr_dim_name

Specify the name of the attribute dimension.

RENAME TO

Specify `RENAME TO` to change the name of the attribute dimension. For *`new_attr_dim_name`*, specify a new name for the attribute dimension.

COMPILE

Specify COMPILE to compile the attribute dimension.

Example

The following statement changes the name of an attribute dimension:

```
ALTER ATTRIBUTE DIMENSION product_attr_dim RENAME TO my_product_attr_dim;
```

ALTER AUDIT POLICY (Unified Auditing)

This section describes the ALTER AUDIT POLICY statement for **unified auditing**. This type of auditing is new beginning with Oracle Database 12*c* and provides a full set of enhanced auditing features. Refer to *Oracle Database Security Guide* for more information on unified auditing.

Purpose

Use the ALTER AUDIT POLICY statement to modify a unified audit policy.

> **See Also:**
>
> - CREATE AUDIT POLICY (Unified Auditing)
> - DROP AUDIT POLICY (Unified Auditing)
> - AUDIT (Unified Auditing)
> - NOAUDIT (Unified Auditing)

Prerequisites

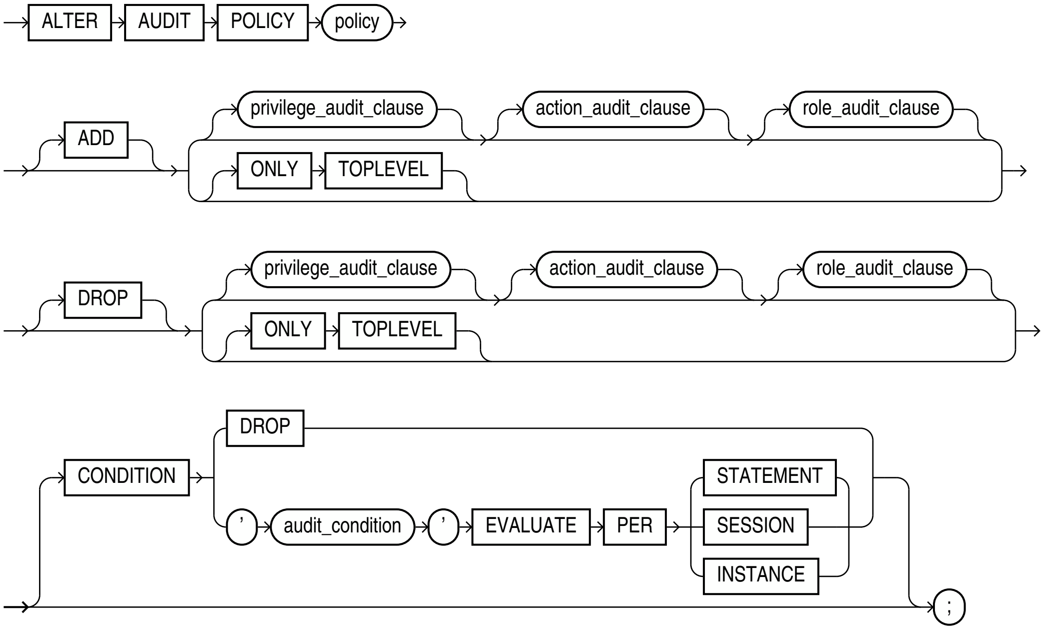
You must have the AUDIT SYSTEM system privilege or the AUDIT_ADMIN role.

If you are connected to a multitenant container database (CDB), then to modify a common unified audit policy, the current container must be the root and you must have the commonly granted AUDIT SYSTEM privilege or the AUDIT_ADMIN common role. To modify a local unified audit policy, the current container must be the container in which the audit policy was created and you must have the commonly granted AUDIT SYSTEM privilege or the AUDIT_ADMIN common role, or you must have the locally granted AUDIT SYSTEM privilege or the AUDIT_ADMIN local role in the container.

After you alter an unified audit policy with object audit options, the new audit settings take place immediately, for both the active and subsequent user sessions. If you alter an unified audit policy with system audit options, or audit conditions, then they become effective only for new user sessions, but not for the current user session.

Syntax

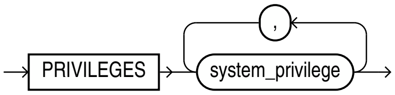
***alter_audit_policy*::=**



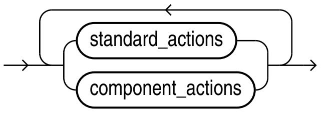
> **Note:**
>
> If you specify the `ADD` or `DROP` clause, then you must specify at least one of the clauses *`privilege_audit_clause`*, *`action_audit_clause`*, or *`role_audit_clause`*.

(*privilege_audit_clause*::=, *action_audit_clause*::=, *role_audit_clause*::=)

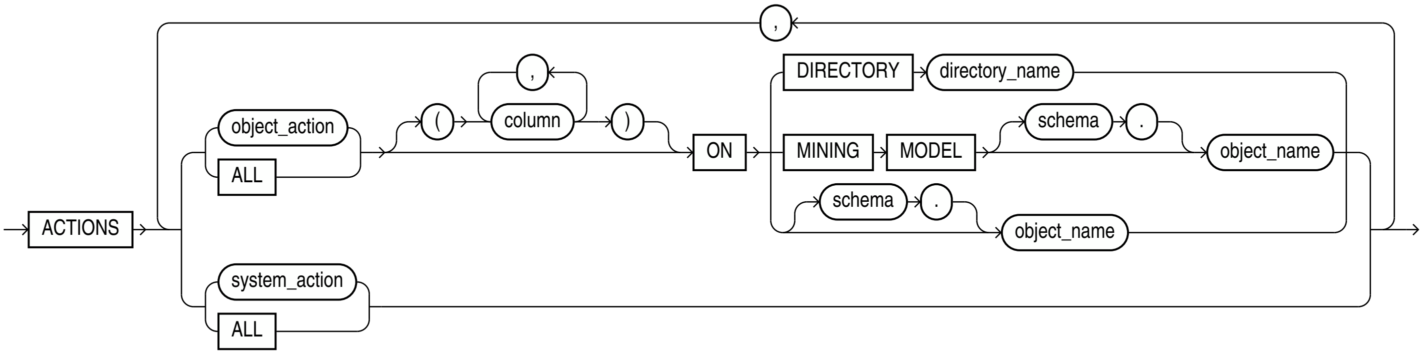
***privilege_audit_clause*::=**



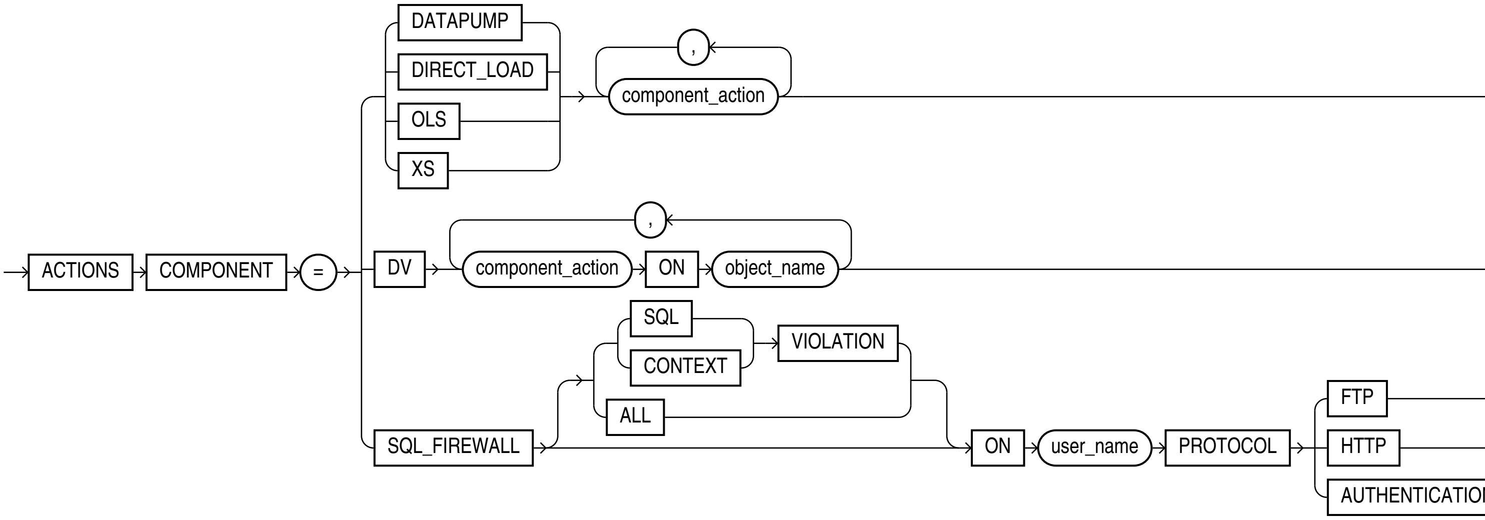
***action_audit_clause*::=**



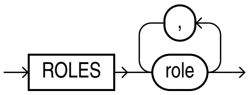
***standard_actions*::=**



***component_actions*::=**



***role_audit_clause*::=**



Semantics

policy

Specify the name of the unified audit policy to be modified. The policy must have been created using the `CREATE AUDIT POLICY` statement. You can find descriptions of all unified audit policies by querying the `AUDIT_UNIFIED_POLICIES` view.

> **See Also:**
>
> - CREATE AUDIT POLICY (Unified Auditing)
> - *Oracle Database Reference* for more information on the `AUDIT_UNIFIED_POLICIES` view

ADD | DROP

Use the `ADD` clause to add privileges to be audited to *`policy`*.

Use the `DROP` clause to remove privileges to be audited from *`policy`*.

Refer to *privilege_audit_clause*, *action_audit_clause*, and *role_audit_clause* of `CREATE AUDIT POLICY` for the full semantics of these clauses.

CONDITION

Use this clause to drop, add, or replace the audit condition for *`policy`*.

Specify `DROP` to drop the audit condition from *`policy`*.

Specify `'`*`audit_condition`*`'` ... to add or replace the audit condition for *`policy`*.

Refer to audit_condition, EVALUATE PER STATEMENT, EVALUATE PER SESSION, and EVALUATE PER INSTANCE of `CREATE AUDIT POLICY` for the full semantics of these clauses.

ONLY TOPLEVEL

Specify this clause to change the existing unified audit policy to audit only the top level SQL statements issued by the user.

Example: Add Top Level Auditing

The example changes the HR audit policy *`hr_audit_policy`* to capture only top level statements.

```
ALTER AUDIT POLICY hr_audit_policy ADD ONLY TOPLEVEL
```

You can drop top level auditing from an existing audit policy auditing the top level SQL statements.

Example: Drop Top Level Auditing

```
ALTER AUDIT POLICY hr_audit_policy DROP ONLY TOPLEVEL
```

See *Database Security Guide* for more information.

Examples

The following examples modify unified audit policies that were created in the `CREATE AUDIT POLICY` "Examples".

Adding Privileges, Actions, and Roles to a Unified Audit Policy: Examples

The following statement adds the system privileges `CREATE ANY TABLE` and `DROP ANY TABLE` to unified audit policy `dml_pol`:

```
ALTER AUDIT POLICY dml_pol
  ADD PRIVILEGES CREATE ANY TABLE, DROP ANY TABLE;
```

The following statement adds the system actions `CREATE JAVA`, `ALTER JAVA`, and `DROP JAVA` to unified audit policy `java_pol`:

```
ALTER AUDIT POLICY java_pol
  ADD ACTIONS CREATE JAVA, ALTER JAVA, DROP JAVA;
```

The following statement adds the role `dba` to unified audit policy `table_pol`:

```
ALTER AUDIT POLICY table_pol
  ADD ROLES dba;
```

The following statement adds multiple system privileges, actions, and roles to unified audit policy `security_pol`:

```
ALTER AUDIT POLICY security_pol
  ADD PRIVILEGES CREATE ANY LIBRARY, DROP ANY LIBRARY
      ACTIONS DELETE on hr.employees,
              INSERT on hr.employees,
              UPDATE on hr.employees,
              ALL on hr.departments
      ROLES dba, connect;
```

Dropping Privileges, Actions, and Roles from a Unified Audit Policy: Examples

The following statement drops the system privilege `CREATE ANY TABLE` from unified audit policy `table_pol`:

```
ALTER AUDIT POLICY table_pol
  DROP PRIVILEGES CREATE ANY TABLE;
```

The following statement drops the `INSERT` and `UPDATE` actions on `hr.employees` from unified audit policy `dml_pol`:

```
ALTER AUDIT POLICY dml_pol
  DROP ACTIONS INSERT on hr.employees,
               UPDATE on hr.employees;
```

The following statement drops the role `java_deploy` from unified audit policy `java_pol`:

```
ALTER AUDIT POLICY java_pol
  DROP ROLES java_deploy;
```

The following statement drops a system privilege, an action, and a role from unified audit policy `hr_admin_pol`:

```
ALTER AUDIT POLICY hr_admin_pol
  DROP PRIVILEGES CREATE ANY TABLE
       ACTIONS LOCK TABLE
       ROLES audit_viewer;
```

Adding and Dropping Actions for a Unified Audit Policy: Example

The following statement adds `EXPORT` actions for Oracle Data Pump to unified audit policy `dp_actions_pol` and drops `IMPORT` actions for Oracle Data Pump:

```
ALTER AUDIT POLICY dp_actions_pol
  ADD ACTIONS COMPONENT = datapump EXPORT
  DROP ACTIONS COMPONENT = datapump IMPORT;
```

Dropping the Audit Condition from a Unified Audit Policy: Example

The following statement drops the audit condition from unified audit policy `order_updates_pol`:

```
ALTER AUDIT POLICY order_updates_pol
  CONDITION DROP;
```

Modifying the Audit Condition for a Unified Audit Policy: Example

The following statement modifies the audit condition for unified audit policy `emp_updates_pol` so that the policy is enforced only when the auditable statement is issued by a user whose UID is 102:

```
ALTER AUDIT POLICY emp_updates_pol
  CONDITION 'UID = 102'
  EVALUATE PER STATEMENT;
```

Altering an Audit Policy at the Column Level: Example

The audit policy `employee_audit_policy` generates audit records only when the select operation is performed on the `sal` column in the `emp` table.

```
CREATE AUDIT POLICY employee_audit_policy ACTIONS SELECT(sal) on scott.emp;
```

The example alters the `employee_audit_policy` so that audit records are generated also when insert operations are done in the `dname` column of the `dept` table.

```
ALTER AUDIT POLICY employee_audit_policy ACTIONS ADD INSERT(dname) on scott.dept;
```

ALTER CLUSTER

Purpose

Use the `ALTER CLUSTER` statement to redefine storage and parallelism characteristics of a cluster.

> **Note:**
>
> You cannot use this statement to change the number or the name of columns in the cluster key, and you cannot change the tablespace in which the cluster is stored.

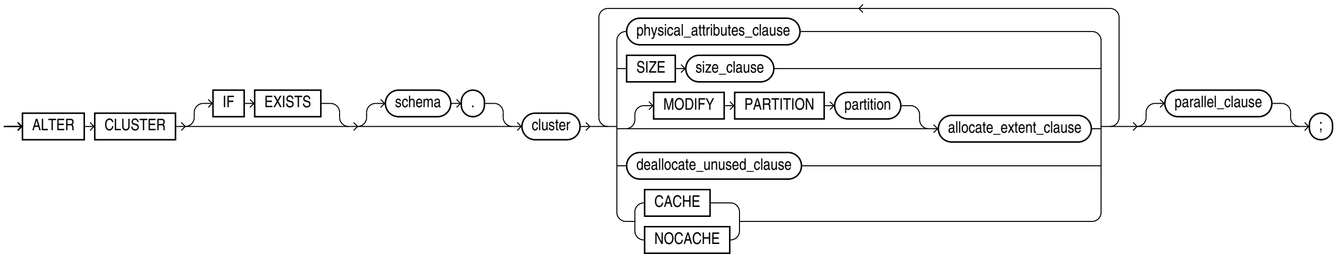
> **See Also:**
>
> CREATE CLUSTER for information on creating a cluster, DROP CLUSTER and DROP TABLE for information on removing tables from a cluster, and CREATE TABLE ... *physical_properties* for information on adding a table to a cluster

Prerequisites

The cluster must be in your own schema or you must have the `ALTER ANY CLUSTER` system privilege.

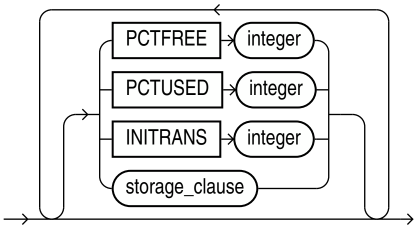
Syntax

***alter_cluster*::=**



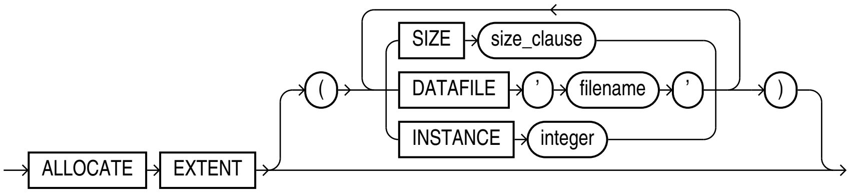
(*physical_attributes_clause*::, *size_clause*::=, MODIFY PARTITION, *allocate_extent_clause*::=, *deallocate_unused_clause*::=, *parallel_clause*::=)

***physical_attributes_clause*::**



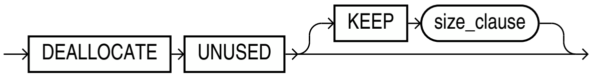
(*storage_clause*::=)

***allocate_extent_clause*::=**



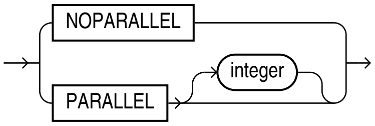
(*size_clause*::=)

***deallocate_unused_clause*::=**



(*size_clause::=*)

***parallel_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 cluster. If you omit *`schema`*, then Oracle Database assumes the cluster is in your own schema.

cluster

Specify the name of the cluster to be altered.

physical_attributes_clause

Use this clause to change the values of the `PCTUSED`, `PCTFREE`, and `INITRANS` parameters of the cluster.

Use the `STORAGE` clause to change the storage characteristics of the cluster.

> **See Also:**
>
> - *physical_attributes_clause* for information on the parameters
> - storage_clause for a full description of that clause

Restriction on Physical Attributes

You cannot change the values of the storage parameters `INITIAL` and `MINEXTENTS` for a cluster.

SIZE

integer

Use the `SIZE` clause to specify the number of cluster keys that will be stored in data blocks allocated to the cluster.

Restriction on SIZE

You can change the `SIZE` parameter only for an indexed cluster, not for a hash cluster.

> **See Also:**
>
> CREATE CLUSTER for a description of the `SIZE` parameter and "Modifying a Cluster: Example"

MODIFY PARTITION

Specify `MODIFY PARTITION` *`partition allocate_extent_clause`* to explicitly allocate a new extent for a cluster partition. This operation is valid only for range-partitioned hash clusters. For *`partition`*, specify the cluster partition name.

allocate_extent_clause

Specify *`allocate_extent_clause`* to explicitly allocate a new extent for a cluster. This operation is valid only for indexed clusters and nonpartitioned hash clusters.

When you explicitly allocate an extent with the *`allocate_extent_clause`*, Oracle Database does not evaluate the storage parameters of the cluster and determine a new size for the next extent to be allocated (as it does when you create a table). Therefore, specify `SIZE` if you do not want Oracle Database to use a default value.

> **See Also:**
>
> *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 cluster and make the freed space available for other segments.

> **See Also:**
>
> *deallocate_unused_clause* for a full description of this clause and "Deallocating Unused Space: Example"

parallel_clause

Specify the *`parallel_clause`* to change the default degree of parallelism for queries on the cluster.

> **See Also:**
>
> *parallel_clause* in the documentation on `CREATE TABLE` for complete information on this clause

Examples

The following examples modify the clusters that were created in the CREATE CLUSTER "Examples".

Modifying a Cluster: Example

The next statement alters the personnel cluster:

```
ALTER CLUSTER personnel
   SIZE 1024 CACHE;
```

Oracle Database allocates 1024 bytes for each cluster key value and enables the cache attribute. Assuming a data block size of 2 kilobytes, future data blocks within this cluster contain 2 cluster keys in each data block, or 2 kilobytes divided by 1024 bytes.

Deallocating Unused Space: Example

The following statement deallocates unused space from the language cluster, keeping 30 kilobytes of unused space for future use:

```
ALTER CLUSTER language
   DEALLOCATE UNUSED KEEP 30 K;
```

Altering Clusters: Example

The following statement creates a cluster with the default key size (600):

```
CREATE CLUSTER EMP_DEPT (DEPTNO NUMBER(3))
   SIZE 600
   TABLESPACE USERS
   STORAGE (INITIAL 200K
      NEXT 300K
      MINEXTENTS 2
      PCTINCREASE 33);
```

The following statement queries USER_CLUSTERS to display the cluster metadata:

```
SELECT CLUSTER_NAME, TABLESPACE_NAME, KEY_SIZE, CLUSTER_TYPE,
AVG_BLOCKS_PER_KEY, MIN_EXTENTS, MAX_EXTENTS FROM USER_CLUSTERS;

CLUSTER_NAME    TABLESPACE_NAME                  KEY_SIZE CLUST
AVG_BLOCKS_PER_KEY MIN_EXTENTS MAX_EXTENTS
--------------- ------------------------------ ---------- -----
------------------ ----------- -----------
EMP_DEPT        USERS                                 600
INDEX                              1  2147483645
```

The following statement modifies the cluster key size:

```
ALTER CLUSTER EMP_DEPT SIZE 1024;
```

The following statement displays the metadata of the modified cluster:

```
SELECT CLUSTER_NAME, TABLESPACE_NAME, KEY_SIZE, CLUSTER_TYPE,
AVG_BLOCKS_PER_KEY, MIN_EXTENTS, MAX_EXTENTS FROM USER_CLUSTERS;
```

```
CLUSTER_NAME    TABLESPACE_NAME                  KEY_SIZE CLUST
AVG_BLOCKS_PER_KEY MIN_EXTENTS MAX_EXTENTS
--------------- ------------------------------ ---------- -----
------------------ ----------- -----------
EMP_DEPT        USERS                                1024
INDEX                              1   2147483645
```

The following statement deallocates unused space from the `EMP_DEPT` cluster, keeping 30 kilobytes of unused space for future use:

```
ALTER CLUSTER EMP_DEPT DEALLOCATE UNUSED KEEP 30 K;
```

The following statement displays the metadata of the modified cluster:

```
SELECT CLUSTER_NAME, TABLESPACE_NAME, KEY_SIZE, CLUSTER_TYPE,
AVG_BLOCKS_PER_KEY, MIN_EXTENTS, MAX_EXTENTS FROM USER_CLUSTERS;

CLUSTER_NAME    TABLESPACE_NAME                  KEY_SIZE CLUST
AVG_BLOCKS_PER_KEY MIN_EXTENTS MAX_EXTENTS
--------------- ------------------------------ ---------- -----
------------------ ----------- -----------
EMP_DEPT        USERS                                1024
INDEX                              1   2147483645
```

Live SQL:

View and run a related example on Oracle Live SQL at *Creating and Altering Clusters*

ALTER DATABASE

Purpose

Use the `ALTER DATABASE` statement to modify, maintain, or recover an existing database.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for examples of performing media recovery
- *Oracle Data Guard Concepts and Administration* for additional information on using the `ALTER DATABASE` statement to maintain standby databases
- CREATE DATABASE for information on creating a database

Prerequisites

You must have the `ALTER DATABASE` system privilege.

To specify the *startup_clauses*, you must also be connected AS SYSDBA, AS SYSOPER, AS SYSBACKUP, or AS SYSDG.

To specify the *general_recovery* clause, you must also have the SYSDBA or SYSBACKUP system privilege.

To specify the DEFAULT EDITION clause, you must also have the USE object privilege WITH GRANT OPTION on the specified edition.

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

- To modify the entire CDB, the current container must be the root and you must have the commonly granted ALTER DATABASE privilege.
- To modify a container, it must be the current container and you must have the ALTER DATABASE privilege, either granted commonly or granted locally in the container.

Notes on Using ALTER DATABASE in a CDB

When you issue the ALTER DATABASE statement while connected to a CDB, the behavior of the statement depends on the current container and the clause(s) you specify.

If the current container is the root, then ALTER DATABASE statements with the following clauses modify the entire CDB. In order to specify these clauses, you must have the commonly granted ALTER DATABASE privilege:

- *startup_clauses*
- *recovery_clauses*

  Note: A subset of the *recovery_clauses* are supported to back up and recover an individual pluggable database (PDB). In order to specify these clauses, you must have the ALTER DATABASE privilege, either granted commonly or granted locally in the PDB. Refer to "Notes on Using the recovery_clauses in a CDB" for more information.
- *logfile_clauses*
- *controlfile_clauses*
- *standby_database_clauses*
- *instance_clauses*
- *security_clause*
- RENAME GLOBAL_NAME TO
- ENABLE BLOCK CHANGE TRACKING
- DISABLE BLOCK CHANGE TRACKING
- *undo_mode_clause*

If the current container is the root, then ALTER DATABASE statements with the following clauses modify only the root. In order to specify these clauses, you must have the ALTER DATABASE privilege, either granted commonly or granted locally in the root:

- *database_file_clauses*
- DEFAULT EDITION
- DEFAULT TABLESPACE

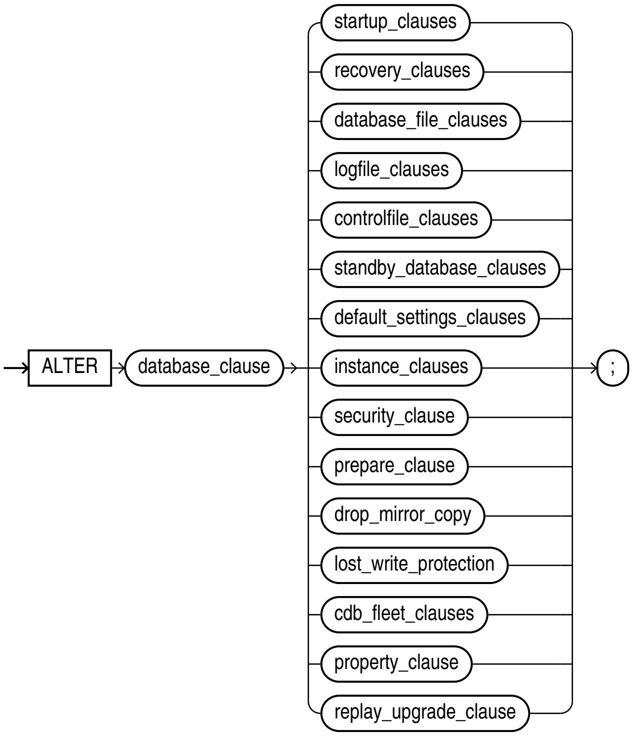
If the current container is the root, then ALTER DATABASE statements with the following clauses modify the root and set default values for the PDBs. In order to specify these clauses, you must have the commonly granted ALTER DATABASE privilege:

- DEFAULT [LOCAL] TEMPORARY TABLESPACE
- *flashback_mode_clause*
- SET DEFAULT { BIGFILE | SMALLFILE } TABLESPACE
- *set_time_zone_clause*

If the current container is a PDB, then ALTER DATABASE statements modify that PDB. In this case, you can issue only ALTER DATABASE clauses that are also supported by the ALTER PLUGGABLE 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. Refer to the documentation on ALTER PLUGGABLE DATABASE for complete information on these clauses.

Syntax

***alter_database*::=**

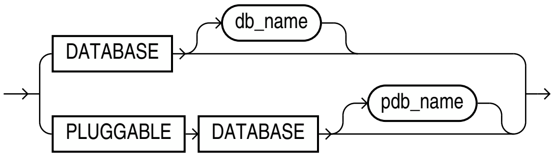


Groups of ALTER DATABASE syntax:

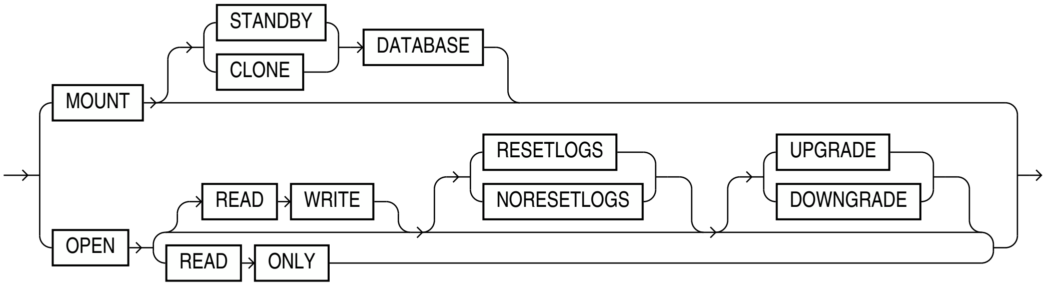
- ***startup_clauses*::=**
- ***recovery_clauses*::=**
- ***database_file_clauses*::=**
- ***logfile_clauses*::=**

- ***controlfile_clauses*::=**
- ***standby_database_clauses*::=**
- ***default_settings_clauses*::=**
- ***instance_clauses*::=**
- ***security_clause*::=**

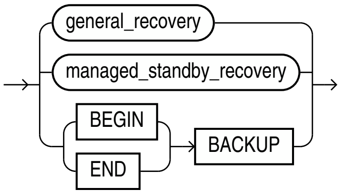
***database_clause*::=**



***startup_clauses*::=**

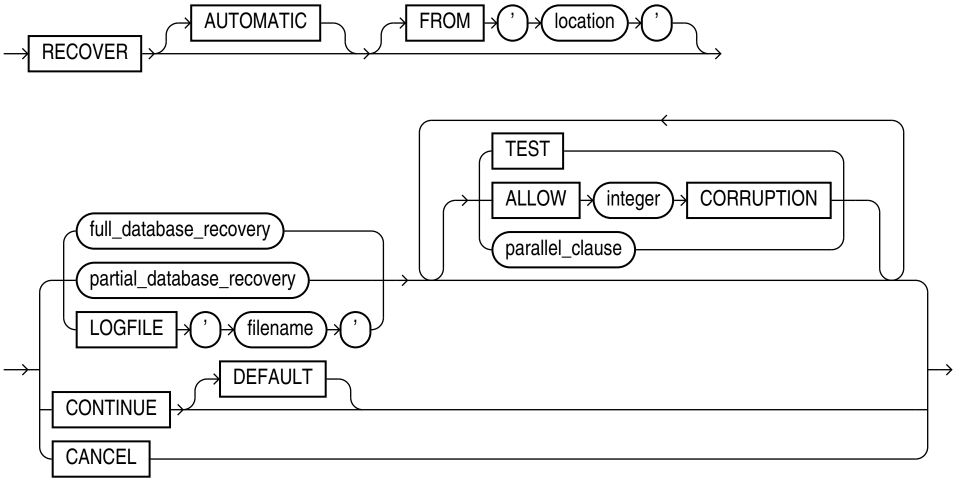


***recovery_clauses*::=**



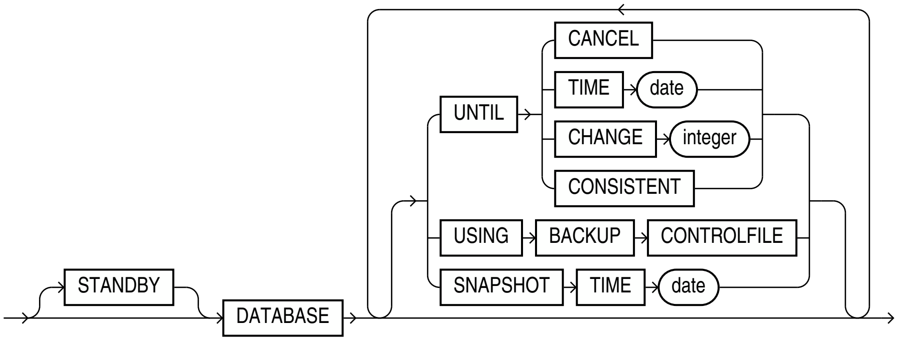
(*general_recovery*::=, *managed_standby_recovery*::=)

***general_recovery*::=**

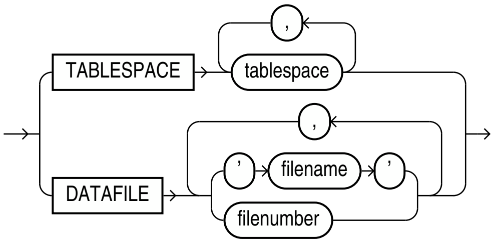


(*full_database_recovery::=*, *partial_database_recovery::=*, *parallel_clause::=*)

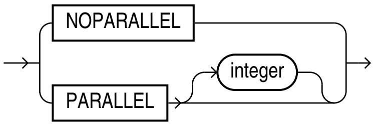
***full_database_recovery*::=**



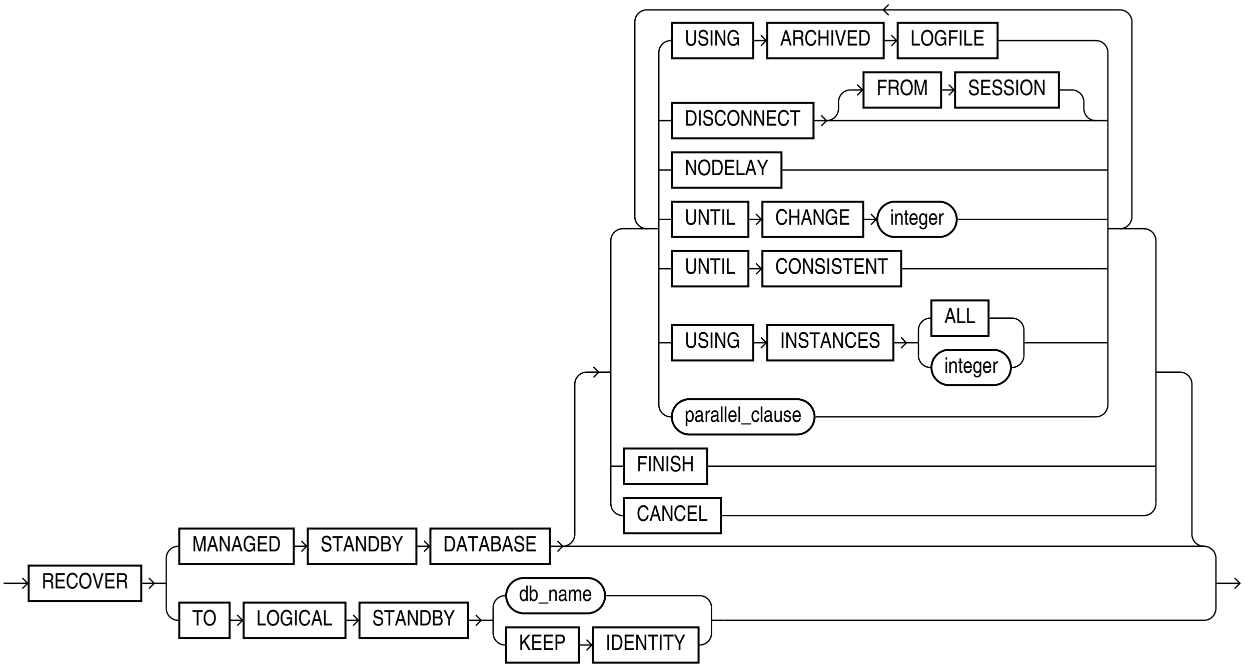
***partial_database_recovery*::=**



***parallel_clause*::=**



***managed_standby_recovery*::=**

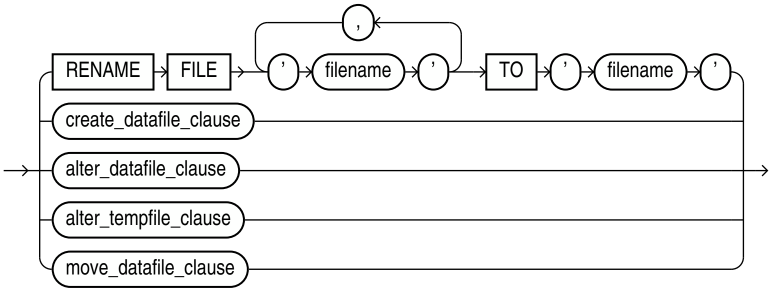


(*parallel_clause*::=)

Note:

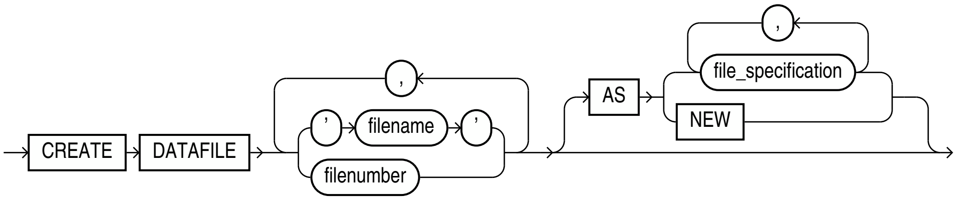
Several subclauses of `managed_standby_recovery` are no longer needed and have been deprecated. These clauses no longer appear in the syntax diagrams. Refer to the semantics of *managed_standby_recovery*.

***database_file_clauses*::=**



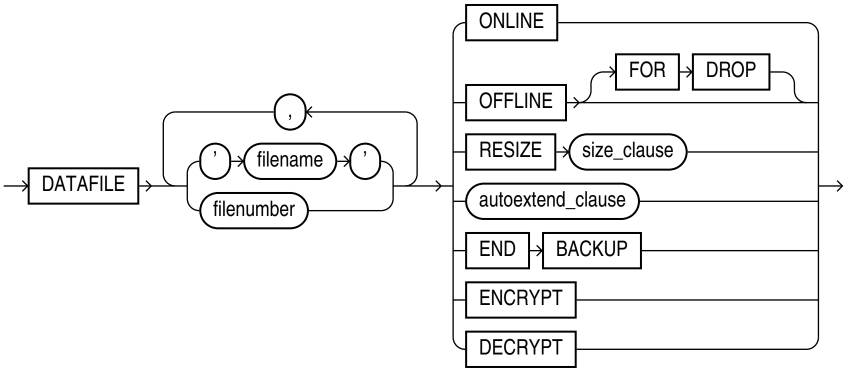
(*create_datafile_clause*::=, *alter_datafile_clause*::=, *alter_tempfile_clause*::=, *move_datafile_clause*::=)

***create_datafile_clause*::=**



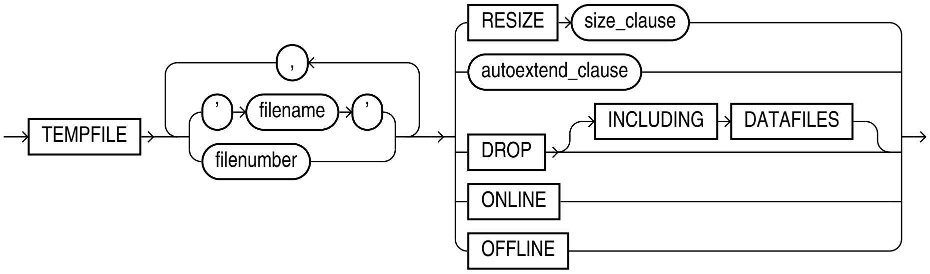
(*file_specification*::=)

***alter_datafile_clause*::=**



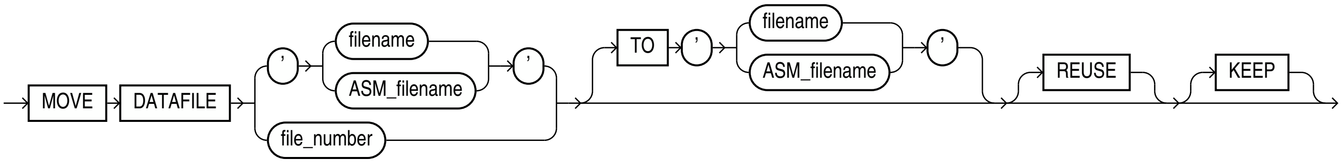
(*autoextend_clause*::=, *size_clause*::=)

***alter_tempfile_clause*::=**

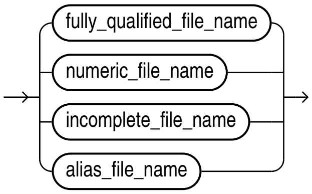


(*autoextend_clause*::=, *size_clause*::=)

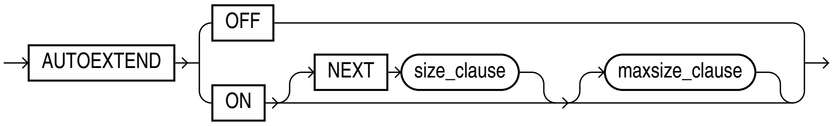
***move_datafile_clause*::=**



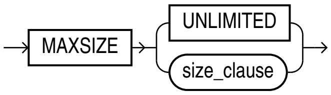
***ASM_filename*::=**



***autoextend_clause*::=**

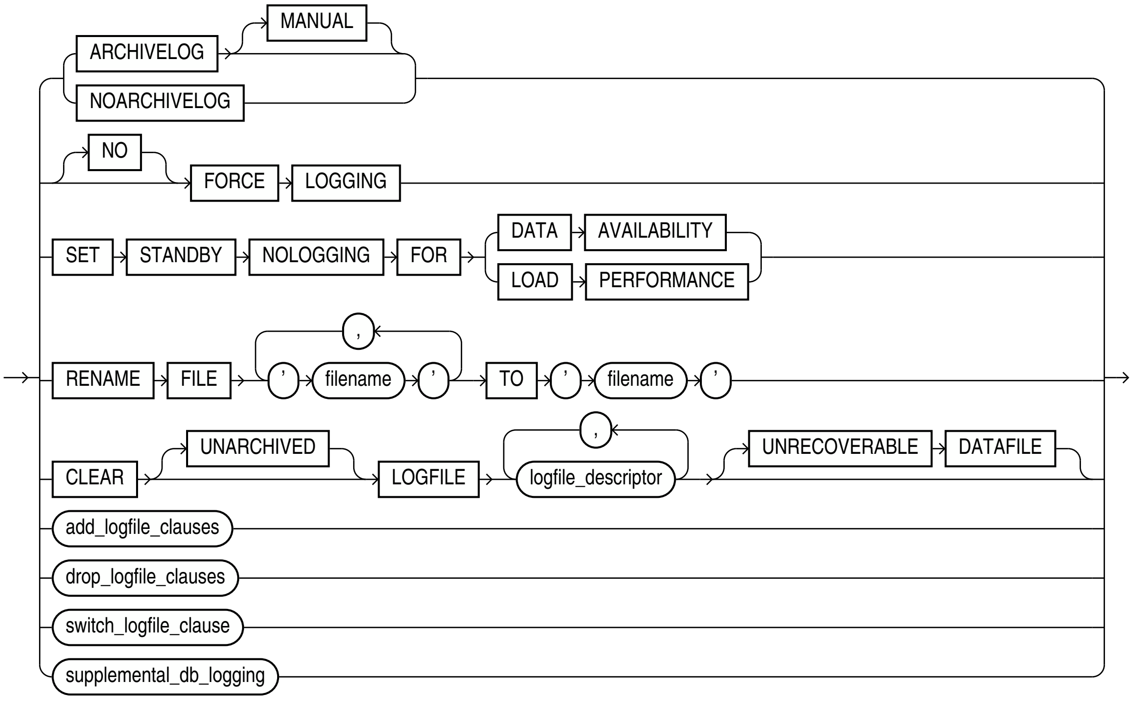


***maxsize_clause*::=**



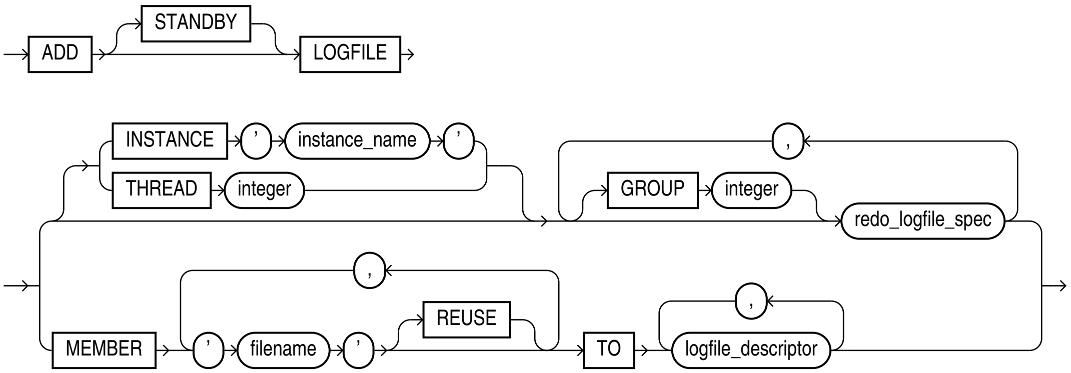
(*size_clause*::=)

***logfile_clauses*::=**



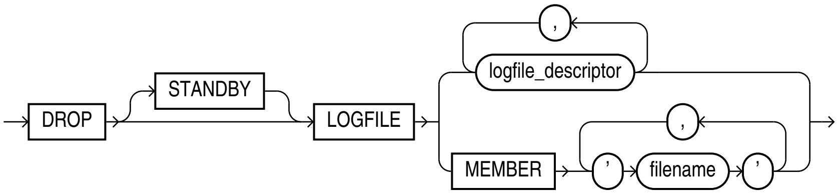
(*logfile_descriptor*::=, *add_logfile_clauses*::=, *drop_logfile_clauses*::=, *switch_logfile_clause*::=, *supplemental_db_logging*::=)

***add_logfile_clauses*::=**



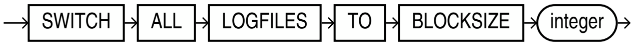
(*redo_log_file_spec*::=, *logfile_descriptor*::=)

***drop_logfile_clauses*::=**

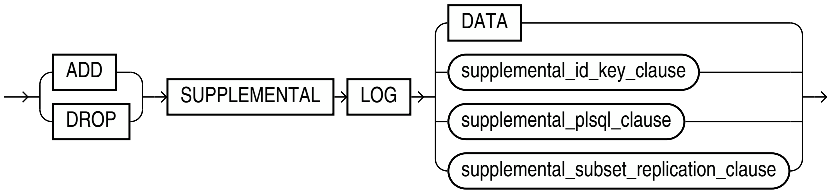


(*logfile_descriptor::=*)

***switch_logfile_clause*::=**

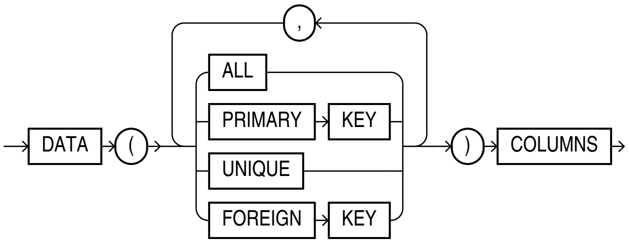


***supplemental_db_logging*::=**

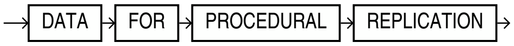

ADD
DROP
SUPPLEMENTAL
LOG
DATA
supplemental_id_key_clause
supplemental_plsql_clause
supplemental_subset_replication_clause

(*supplemental_id_key_clause::=*)

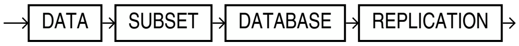
***supplemental_id_key_clause*::=**



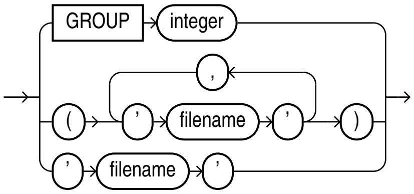
***supplemental_plsql_clause*::=**



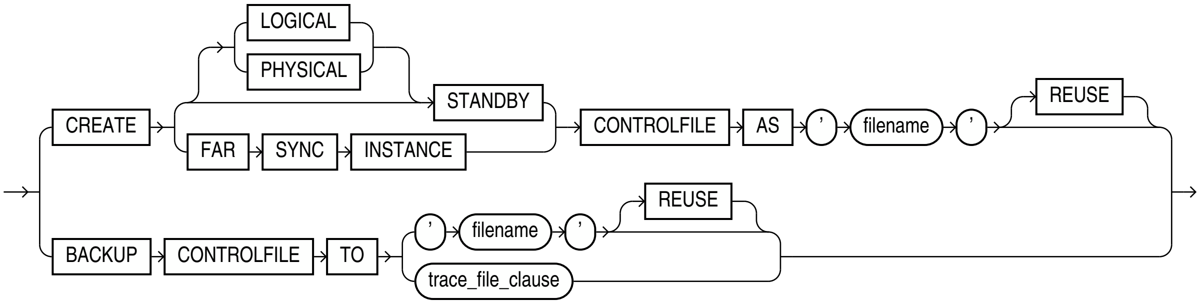
supplemental_subset_replication_clause



***logfile_descriptor*::=**

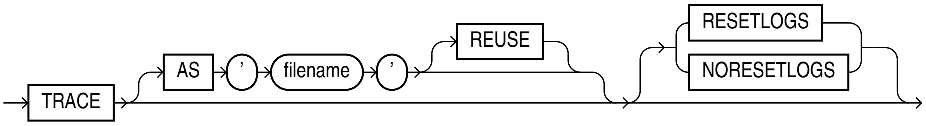


***controlfile_clauses*::=**

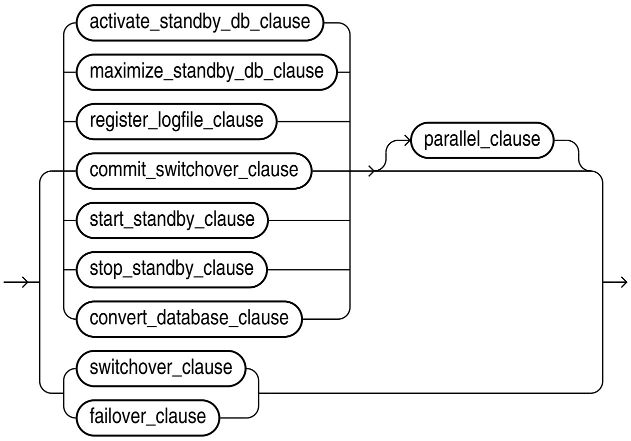


(*trace_file_clause*::=)

***trace_file_clause*::=**

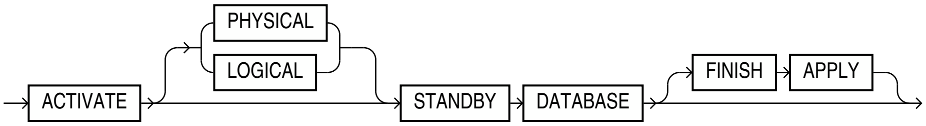


***standby_database_clauses*::=**

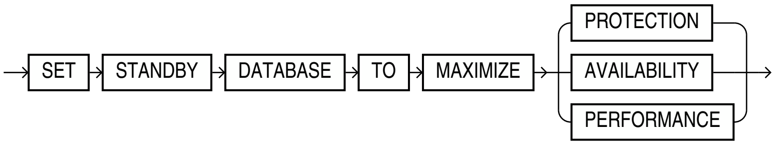


(*activate_standby_db_clause*::=, *maximize_standby_db_clause*::=, *register_logfile_clause*::=, *commit_switchover_clause*::=, *start_standby_clause*::=, *stop_standby_clause*::=, *convert_database_clause*::=, *parallel_clause*::=, *switchover_clause*::=, *failover_clause*::=)

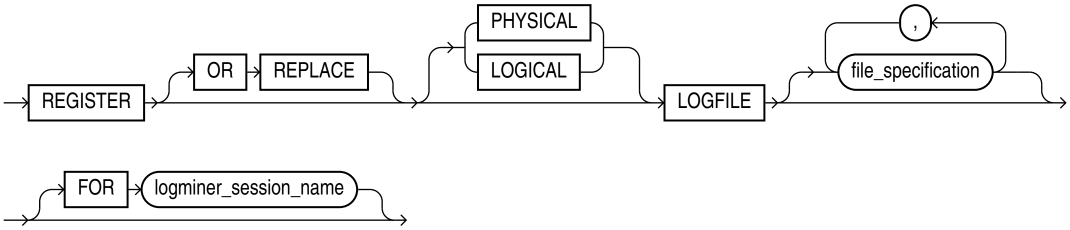
***activate_standby_db_clause*::=**



***maximize_standby_db_clause*::=**

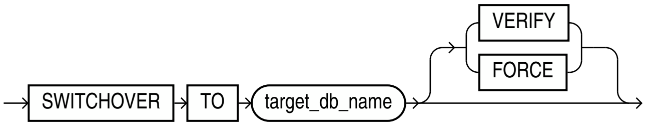


***register_logfile_clause*::=**

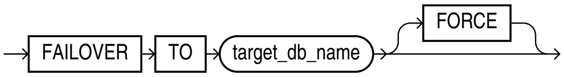


(*file_specification*::=)

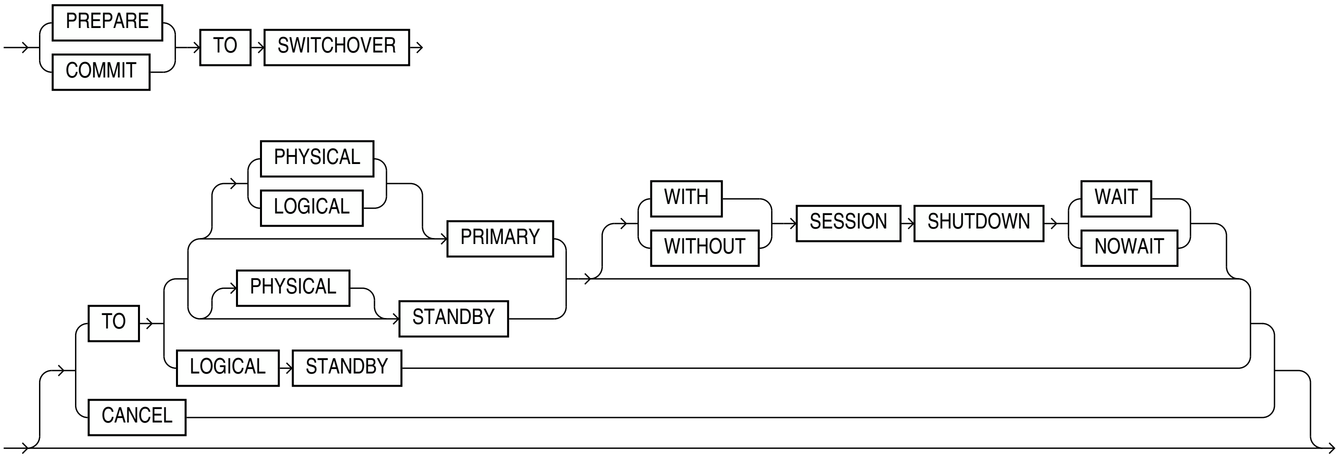
***switchover_clause*::=**



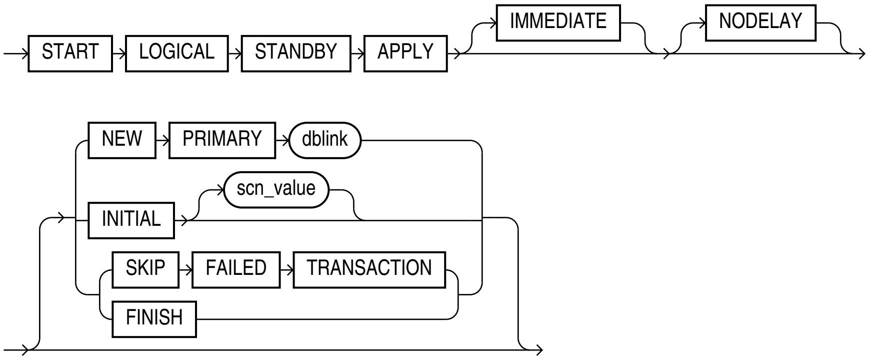
***failover_clause*::=**



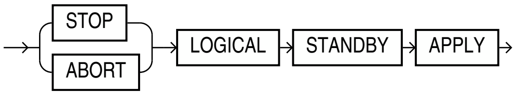
***commit_switchover_clause*::=**



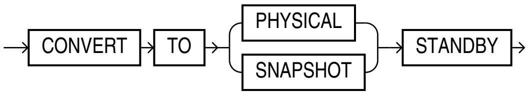
***start_standby_clause*::=**



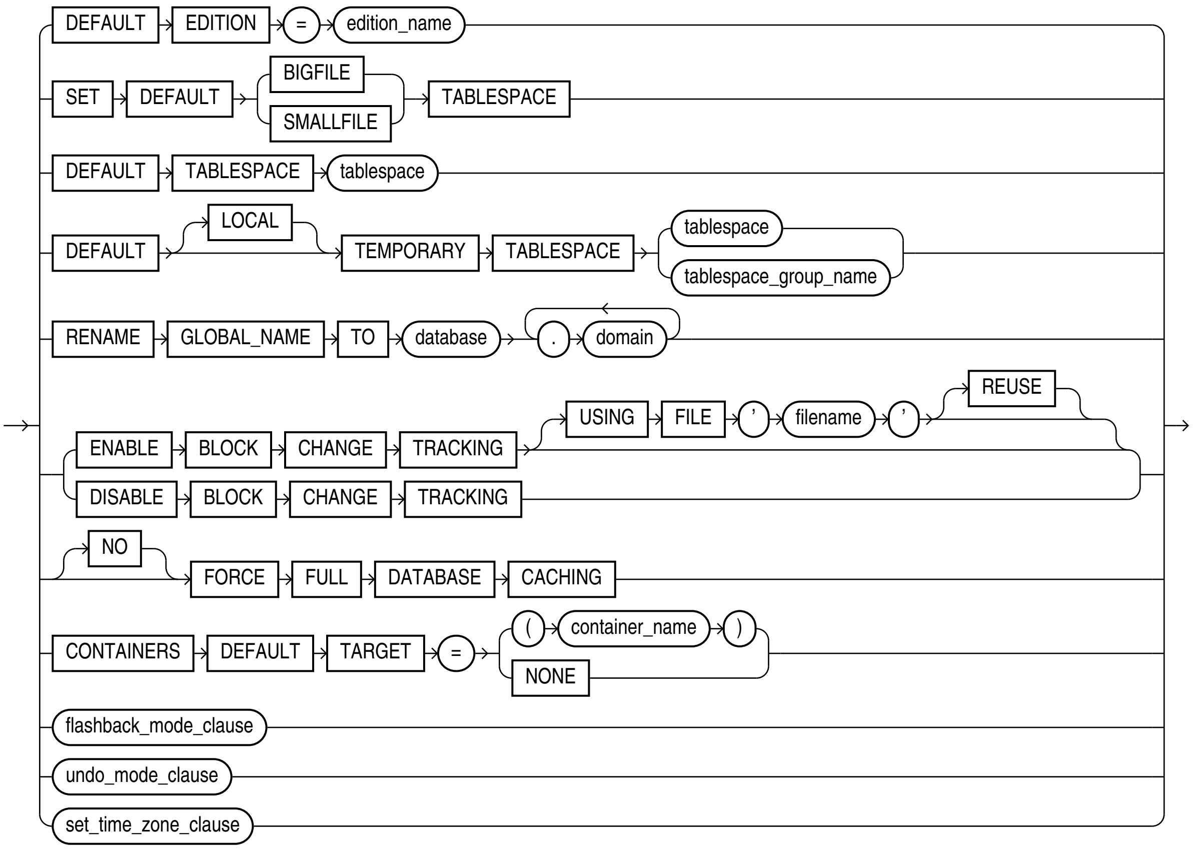
***stop_standby_clause*::=**



***convert_database_clause*::=**

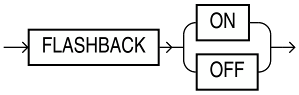


***default_settings_clauses*::=**

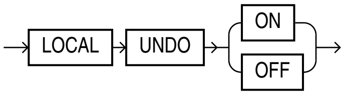


(*flashback_mode_clause*::=, *undo_mode_clause*::=, *set_time_zone_clause*::=)

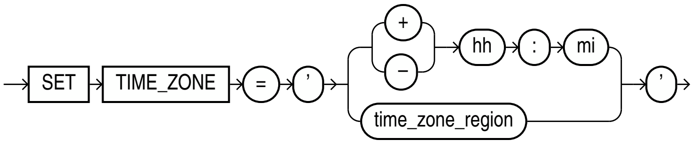
***flashback_mode_clause*::=**



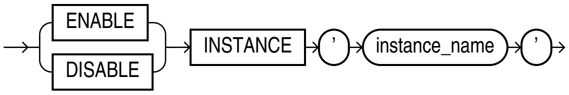
***undo_mode_clause*::=**



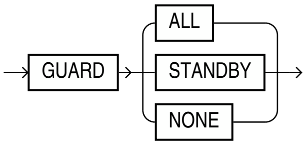
***set_time_zone_clause*::=**



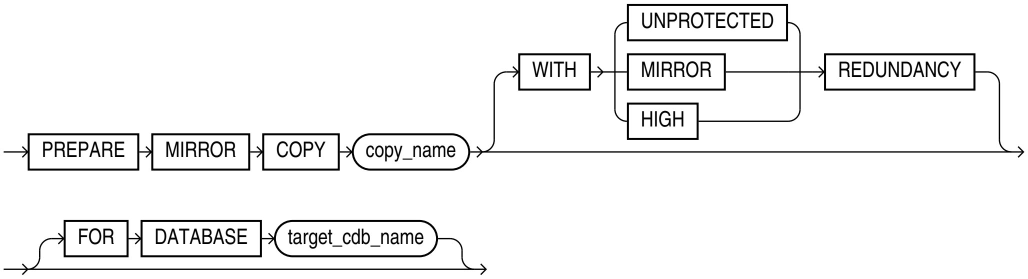
instance_clauses::=



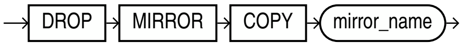
security_clause::=



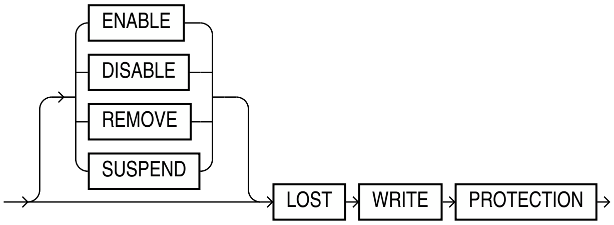
prepare_clause::=



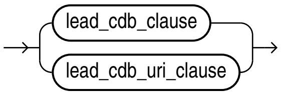
drop_mirror_copy::=



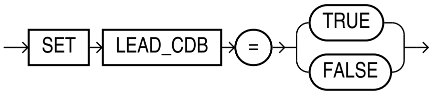
lost_write_protection ::=



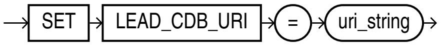
cdb_fleet_clauses::=



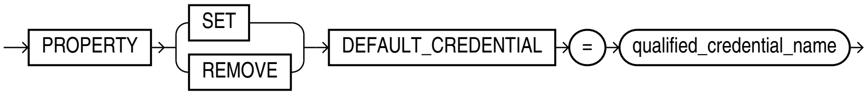
***lead_cdb_clause*::=**



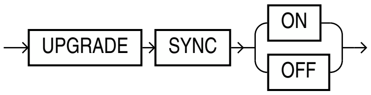
***lead_cdb_uri_clause*::=**



property_clause



***replay_upgrade_clause*::=**



Semantics

database_clause

Specify the `DATABASE` option for a non-container database.

db_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.

startup_clauses

The *`startup_clauses`* let you mount and open the database so that it is accessible to users.

MOUNT Clause

Use the `MOUNT` clause to mount the database. Do not use this clause when the database is already mounted.

MOUNT STANDBY DATABASE

You can specify `MOUNT STANDBY DATABASE` to mount a physical standby database. The keywords `STANDBY DATABASE` are optional, because Oracle Database determines automatically whether the database to be mounted is a primary or standby database. As soon as this statement executes, the standby instance can receive redo data from the primary instance.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for more information on standby databases

MOUNT CLONE DATABASE

Specify `MOUNT CLONE DATABASE` to mount the clone database.

OPEN Clause

Use the `OPEN` clause to make the database available for normal use. You must mount the database before you can open it.

If you specify only `OPEN` without any other keywords, then the default is `OPEN READ WRITE NORESETLOGS` on a primary database, logical standby database, or snapshot standby database and `OPEN READ ONLY` on a physical standby database.

OPEN READ WRITE

Specify `OPEN READ WRITE` to open the database in read/write mode, allowing users to generate redo logs. This is the default if you are opening a primary database. You cannot specify this clause for a physical standby database.

> **See Also:**
>
> "READ ONLY / READ WRITE: Example"

RESETLOGS | NORESETLOGS

This clause determines whether Oracle Database resets the current log sequence number to 1, archives any unarchived logs (including the current log), and discards any redo information that was not applied during recovery, ensuring that it will never be applied. Oracle Database uses `NORESETLOGS` automatically except in the following specific situations, which require a setting for this clause:

- You must specify `RESETLOGS`:
  - After performing incomplete media recovery or media recovery using a backup control file
  - After a previous `OPEN RESETLOGS` operation that did not complete
  - After a `FLASHBACK DATABASE` operation
- If a created control file is mounted, then you must specify `RESETLOGS` if the online logs are lost, or you must specify `NORESETLOGS` if they are not lost.

UPGRADE | DOWNGRADE

Use these `OPEN` clause parameters only if you are upgrading or downgrading a database. This clause instructs Oracle Database to modify system parameters dynamically as required for upgrade and downgrade, respectively. You can achieve the same result using the SQL*Plus `STARTUP UPGRADE` or `STARTUP DOWNGRADE` command.

When you use the `UPGRADE` or `DOWNGRADE` parameters for a CDB, the root container is opened in the specified mode, but all other containers are opened in `READ WRITE` mode.

> **See Also:**
>
> - *Oracle Database Upgrade Guide* for information on the steps required to upgrade or downgrade a database from one release to another
> - *SQL*Plus User's Guide and Reference* for information on the SQL*Plus `STARTUP` command

OPEN READ ONLY

Specify `OPEN READ ONLY` to restrict users to read-only transactions, preventing them from generating redo logs. This setting is the default when you are opening a physical standby database, so that the physical standby database is available for queries even while archive logs are being copied from the primary database site.

Restrictions on Opening a Database

The following restrictions apply to opening a database:

- You cannot open a database in `READ ONLY` mode if it is currently opened in `READ WRITE` mode by another instance.
- You cannot open a database in `READ ONLY` mode if it requires recovery.
- You cannot take tablespaces offline while the database is open in `READ ONLY` mode. However, you can take data files offline and online, and you can recover offline data files and tablespaces while the database is open in `READ ONLY` mode.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for additional information about opening a physical standby database

recovery_clauses

The *`recovery_clauses`* include post-backup operations. For all of these clauses, Oracle Database recovers the database using any incarnations of data files and log files that are known to the current control file.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for information on backing up the database and "Database Recovery: Examples"

Notes on Using the *recovery_clauses* in a CDB

When the current container is the root, you can specify all of the *recovery_clauses* to back up and recover the entire CDB.

When the current container is a PDB, you can specify the following subclauses of the *recovery_clauses* to back up and recover the PDB:

- BEGIN BACKUP
- END BACKUP
- *full_database_recovery*: You can specify only the DATABASE keyword
- *partial_database_recovery*
- The LOGFILE and CONTINUE clauses of *general_recovery*

You can also specify the preceding subclauses using the *pdb_recovery_clauses* of ALTER PLUGGABLE DATABASE. Refer to the syntax diagram *pdb_recovery_clauses* of ALTER PLUGGABLE DATABASE.

general_recovery

The *general_recovery* clause lets you control media recovery for the database or standby database or for specified tablespaces or files. You can use this clause when your instance has the database mounted, open or closed, and the files involved are not in use.

> **Note:**
>
> Parallelism is enabled by default during full or partial database recovery and logfile recovery. The database computes the degree of parallelism. You can disable parallelism of these operations by specifying NOPARALLEL, or specify a degree of parallelism with PARALLEL *integer*, as shown in the respective syntax diagrams.

Restrictions on General Database Recovery

General recovery is subject to the following restrictions:

- You can recover the entire database only when the database is closed.
- Your instance must have the database mounted in exclusive mode.
- You can recover tablespaces or data files when the database is open or closed, if the tablespaces or data files to be recovered are offline.
- You cannot perform media recovery if you are connected to Oracle Database through the shared server architecture.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for more information on RMAN media recovery and user-defined media recovery
> - *SQL*Plus User's Guide and Reference* for information on the SQL*Plus RECOVER command

AUTOMATIC

Specify AUTOMATIC if you want Oracle Database to automatically generate the name of the next archived redo log file needed to continue the recovery operation. If the LOG_ARCHIVE_DEST_*n* parameters are defined, then Oracle Database scans those that are valid and enabled for the first local destination. It uses that destination in conjunction with LOG_ARCHIVE_FORMAT to generate the target redo log filename. If the LOG_ARCHIVE_DEST_*n* parameters are not defined, then Oracle Database uses the value of the LOG_ARCHIVE_DEST parameter instead.

If the resulting file is found, then Oracle Database applies the redo contained in that file. If the file is not found, then Oracle Database prompts you for a filename, displaying the generated filename as a suggestion.

If you specify neither AUTOMATIC nor LOGFILE, then Oracle Database prompts you for a filename, displaying the generated filename as a suggestion. You can then accept the generated filename or replace it with a fully qualified filename. If you know that the archived filename differs from what Oracle Database would generate, then you can save time by using the LOGFILE clause.

FROM *'location'*

Specify FROM *'location'* to indicate the location from which the archived redo log file group is read. The value of *location* must be a fully specified file location following the conventions of your operating system. If you omit this parameter, then Oracle Database assumes that the archived redo log file group is in the location specified by the initialization parameter LOG_ARCHIVE_DEST or LOG_ARCHIVE_DEST_1.

full_database_recovery

The *full_database_recovery* clause lets you recover an entire database.

DATABASE

Specify the DATABASE clause to recover the entire database. This is the default. You can use this clause only when the database is closed.

STANDBY DATABASE

Specify the STANDBY DATABASE clause to manually recover a physical standby database using the control file and archived redo log files copied from the primary database. The standby database must be mounted but not open.

This clause recovers only online data files.

- Use the UNTIL clause to specify the duration of the recovery operation.
  - CANCEL indicates cancel-based recovery. This clause recovers the database until you issue the ALTER DATABASE statement with the RECOVER CANCEL clause.
  - TIME indicates time-based recovery. This parameter recovers the database to the time specified by the date. The date must be a character literal in the format 'YYYY-MM-DD:HH24:MI:SS'.
  - CHANGE indicates change-based recovery. This parameter recovers the database to a transaction-consistent state immediately before the system change number specified by *integer*.
  - CONSISTENT recovers the database until all online files are brought to a consistent SCN point so that the database can be open in read only mode. This clauses requires the controlfile to be a backup controlfile.
- Specify USING BACKUP CONTROLFILE if you want to use a backup control file instead of the current control file.

- Specify the SNAPSHOT TIME clause to recover the database with a storage snapshot using Storage Snapshot Optimization. This clause can be used in cases where the database was not placed in backup mode when the storage snapshot was created.
  - *date* must be a character literal in the format 'YYYY-MM-DD:HH24:MI:SS'. It must represent a time that is immediately after the snapshot was completed. If you specify the UNTIL TIME clause, then SNAPSHOT TIME *date* must be earlier than UNTIL TIME *date*.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information on recovery using Storage Snapshot Optimization

partial_database_recovery

The *partial_database_recovery* clause lets you recover individual tablespaces and data files.

TABLESPACE

Specify the TABLESPACE clause to recover only the specified tablespaces. You can use this clause if the database is open or closed, provided the tablespaces to be recovered are offline.

> **See Also:**
>
> "Using Parallel Recovery Processes: Example"

DATAFILE

Specify the DATAFILE clause to recover the specified data files. You can use this clause when the database is open or closed, provided the data files to be recovered are offline.

You can identify the data file by name or by number. If you identify it by number, then *filenumber* is an integer representing the number found in the FILE# column of the V$DATAFILE dynamic performance view or in the FILE_ID column of the DBA_DATA_FILES data dictionary view.

STANDBY {TABLESPACE | DATAFILE}

In earlier releases, you could specify STANDBY TABLESPACE or STANDBY DATAFILE to recover older backups of a specific tablespace or a specific data file on the standby to be consistent with the rest of the standby database. These two clauses are now desupported. Instead, to recover the standby database to a consistent point, but no further, use the statement ALTER DATABASE RECOVER MANAGED STANDBY DATABASE UNTIL CONSISTENT.

LOGFILE

Specify the LOGFILE '*filename*' to continue media recovery by applying the specified redo log file.

TEST

Use the TEST clause to conduct a trial recovery. A trial recovery is useful if a normal recovery procedure has encountered some problem. It lets you look ahead into the redo stream to detect possible additional problems. The trial recovery applies redo in a way similar to normal recovery, but it does not write changes to disk, and it rolls back its changes at the end of the trial recovery.

You can use this clause only if you have restored a backup taken since the last `RESETLOGS` operation. Otherwise, Oracle Database returns an error.

ALLOW ... CORRUPTION

The `ALLOW` *`integer`* `CORRUPTION` clause lets you specify, in the event of logfile corruption, the number of corrupt blocks that can be tolerated while allowing recovery to proceed.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for information on database recovery in general
> - *Oracle Data Guard Concepts and Administration* for information on managed recovery of standby databases

CONTINUE

Specify `CONTINUE` to continue multi-instance recovery after it has been interrupted to disable a thread.

Specify `CONTINUE DEFAULT` to continue recovery using the redo log file that Oracle Database would automatically generate if no other logfile were specified. This clause is equivalent to specifying `AUTOMATIC`, except that Oracle Database does not prompt for a filename.

CANCEL

Specify `CANCEL` to terminate cancel-based recovery.

managed_standby_recovery

Use the *`managed_standby_recovery`* clause to start and stop Redo Apply on a physical standby database. Redo Apply keeps the standby database transactionally consistent with the primary database by continuously applying redo received from the primary database.

A primary database transmits its redo data to standby sites. As the redo data is written to redo log files at the physical standby site, the log files become available for use by Redo Apply. You can use the *`managed_standby_recovery`* clause when your standby instance has the database mounted or is opened read-only.

Note:

Beginning with Oracle Database 12*c*, **real-time apply** is enabled by default during Redo Apply. Real-time apply recovers redo from the standby redo log files as soon as they are written, without requiring them to be archived first at the physical standby database. You can disable real-time apply with the `USING ARCHIVED LOGFILE` clause. Refer to:

- *Oracle Data Guard Concepts and Administration* for more information on real-time apply
- USING ARCHIVED LOGFILE Clause

Note:

Parallelism is enabled by default during Redo Apply. The database computes the degree of parallelism. You can disable parallelism of these operations by specifying `NOPARALLEL`, or specify a degree of parallelism with `PARALLEL` *`integer`*, as shown in the respective syntax diagrams.

Restrictions on Managed Standby Recovery

The same restrictions listed under *general_recovery* apply to this clause.

See Also:

Oracle Data Guard Concepts and Administration for more information on the use of this clause

USING ARCHIVED LOGFILE Clause

Specify `USING ARCHIVED LOGFILE` to start Redo Apply without enabling real-time apply.

DISCONNECT

Specify `DISCONNECT` to indicate that Redo Apply should be performed in the background, leaving the current session available for other tasks. The `FROM SESSION` keywords are optional and are provided for semantic clarity.

NODELAY

The `NODELAY` clause overrides the `DELAY` attribute on the `LOG_ARCHIVE_DEST_n` parameter on the primary database. If you do not specify the `NODELAY` clause, then application of the archived redo log file is delayed according to the `DELAY` attribute of the `LOG_ARCHIVE_DEST_`*`n`* setting (if any). If the `DELAY` attribute was not specified on that parameter, then the archived redo log file is applied immediately to the standby database.

If you specify real-time apply with the `USING CURRENT LOGFILE` clause, then any `DELAY` value specified for the `LOG_ARCHIVE_DEST_`*`n`* parameter at the primary for this standby is ignored, and `NODELAY` is the default.

UNTIL CHANGE Clause

Use this clause to instruct Redo Apply to recover redo data up to, but not including, the specified system change number.

UNTIL CONSISTENT

Use this clause to recover the standby database to a consistent SCN point so that the standby database can be opened in read only mode.

USING INSTANCES

This clause is applicable only for Oracle Real Application Clusters (Oracle RAC) or Oracle RAC One Node databases and allows you to start apply processes on multiple instances of the standby that are started in the same mode (`MOUNTED` or `READ ONLY`) as the instance on which the command is executed. Specify `USING INSTANCES ALL` to perform Redo Apply on all instances in an Oracle RAC standby database started in the same mode. Specify `USING INSTANCES` *`integer`* to perform Redo Apply on the specified number of instances that are started in the same mode. For *`integer`*, specify an integer value from 1 to the number of instances in the standby database. The database chooses the instances on which to perform Redo Apply; you cannot specify particular instances. For example, if you specify 4 instances from an instance that is `MOUNTED` and only 3 instances of the standby are running in the `MOUNTED` mode, then Redo Apply will only be started on 3 instances. If you omit the `USING INSTANCES` clause, then Oracle Database performs Redo Apply only on the instance where the command was executed.

FINISH

Specify `FINISH` to complete applying all available redo data in preparation for a failover.

Use the `FINISH` clause only in the event of the failure of the primary database. This clause overrides any specified delay intervals and applies all available redo immediately. After the `FINISH` command completes, this database can no longer run in the standby database role, and it must be converted to a primary database by issuing the `ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY` statement.

CANCEL

Specify `CANCEL` to stop Redo Apply immediately. Control is returned as soon as Redo Apply stops.

TO LOGICAL STANDBY Clause

Use this clause to convert a physical standby database into a logical standby database.

db_name

Specify a database name to identify the new logical standby database. If you are using a server parameter file (spfile) at the time you issue this statement, then the database will update the file with appropriate information about the new logical standby database. If you are not using an spfile, then the database issues a message reminding you to set the name of the `DB_NAME` parameter after shutting down the database. In addition, you must invoke the `DBMS_LOGSTDBY.BUILD` PL/SQL procedure on the primary database before using this clause on the standby database.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_LOGSTDBY.BUILD` procedure

KEEP IDENTITY

Use this clause if you want to use the rolling upgrade feature provided by a logical standby and also revert to the original configuration of a primary database and a physical standby. A logical standby database created using this clause provides only limited support for switchover and failover. Therefore, do not use this clause create a general-purpose logical standby database.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for more information on rolling upgrade

Deprecated Managed Standby Recovery Clauses

The following clauses appeared in the syntax of earlier releases. They have been deprecated and are no longer needed. Oracle recommends that you do not use these clauses.

FINISH FORCE, FINISH WAIT, FINISH NOWAIT

These optional forms of the `FINISH` clause are deprecated. Their semantics are presented here for backward compatibility:

- `FORCE` terminates inactive redo transport sessions that would otherwise prevent `FINISH` processing from beginning.
- `NOWAIT` returns control to the foreground process before the recovery completes
- `WAIT` (the default) returns control to the foreground process after recovery completes

When specified, these clauses are ignored. Terminal recovery now runs in the foreground and always terminates all redo transport sessions. Therefore control is not returned to the user until recovery completes.

CANCEL IMMEDIATE, CANCEL WAIT, CANCEL NOWAIT

These optional forms of the `CANCEL` clause are deprecated. Their semantics are presented here for backward compatibility:

- Include the `IMMEDIATE` keyword to stop Redo Apply *before* completely applying the current redo log file. Session control returns when Redo Apply actually stops.
- Include the `NOWAIT` keyword to return session control without waiting for the `CANCEL` operation to complete.

When specified, these clauses are ignored. Redo Apply is now always cancelled immediately and control returns to the session only after the operation completes.

USING CURRENT LOGFILE Clause

The `USING CURRENT LOGFILE` clause is deprecated. It invokes real-time apply during Redo Apply. However, this is now the default behavior and this clause is no longer useful.

BACKUP Clauses

Use these clauses to move all the data files in the database into or out of online backup mode (also called hot backup mode).

> **See Also:**
>
> ALTER TABLESPACE for information on moving all data files in an individual tablespace into and out of online backup mode

BEGIN BACKUP Clause

Specify `BEGIN BACKUP` to move all data files in the database into online backup mode. The database must be mounted and open, and media recovery must be enabled (the database must be in `ARCHIVELOG` mode).

While the database is in online backup mode, you cannot shut down the instance normally, begin backup of an individual tablespace, or take any tablespace offline or make it read only.

This clause has no effect on data files that are in offline or on read-only tablespaces.

END BACKUP Clause

Specify `END BACKUP` to take out of online backup mode any data files in the database currently in online backup mode. The database must be mounted (either open or closed) when you perform this operation.

After a system failure, instance failure, or `SHUTDOWN ABORT` operation, Oracle Database does not know whether the files in online backup mode match the files at the time the system crashed. If you know the files are consistent, then you can take either individual data files or all data files out of online backup mode. Doing so avoids media recovery of the files upon startup.

- To take an individual data file out of online backup mode, use the `ALTER DATABASE DATAFILE` ... `END BACKUP` statement. See *database_file_clauses* .
- To take all data files in a tablespace out of online backup mode, use an `ALTER TABLESPACE` ... `END BACKUP` statement.

database_file_clauses

The *`database_file_clauses`* let you modify data files and temp files. You can use any of the following clauses when your instance has the database mounted, open or closed, and the files involved are not in use. The exception is the *`move_datafile_clause`*, which allows you to move a data file that is in use.

RENAME FILE Clause

Use the `RENAME FILE` clause to rename data files, temp files, or redo log file members. You must create each filename using the conventions for filenames on your operating system before specifying this clause.

- To use this clause for a data file or temp file, the database must be mounted. The database can also be open, but the data file or temp file being renamed must be offline. In addition, you must first rename the file on the file system to the new name.
- To use this clause for logfiles, the database must be mounted but not open.

- If you have enabled block change tracking, then you can use this clause to rename the block change tracking file. The database must be mounted but not open when you rename the block change tracking file.

This clause renames only files in the control file. It does not actually rename them on your operating system. The operating system files continue to exist, but Oracle Database no longer uses them.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for information on recovery of data files and temp files
> - "Renaming a Log File Member: Example" and "Manipulating Temp Files: Example"

create_datafile_clause

Use the `CREATE DATAFILE` clause to create a new empty data file in place of an old one. You can use this clause to re-create a data file that was lost with no backup. The *`filename`* or *`filenumber`* must identify a file that is or was once part of the database. If you identify the file by number, then *`filenumber`* is an integer representing the number found in the `FILE#` column of the `V$DATAFILE` dynamic performance view or in the `FILE_ID` column of the `DBA_DATA_FILES` data dictionary view.

- Specify `AS NEW` to create an Oracle-managed data file with a system-generated filename, the same size as the file being replaced, in the default file system location for data files.
- Specify `AS` *`file_specification`* to assign a file name (and optional size) to the new data file. Use the *`datafile_tempfile_spec`* form of *`file_specification`* (see *file_specification*) to list regular data files and temp files in an operating system file system or to list Oracle Automatic Storage Management (Oracle ASM) disk group files.

If the original file (*`filename`* or *`filenumber`*) is an existing Oracle-managed data file, then Oracle Database attempts to delete the original file after creating the new file. If the original file is an existing user-managed data file, then Oracle Database does not attempt to delete the original file.

If you omit the `AS` clause entirely, then Oracle Database creates the new file with the same name and size as the file specified by *`filename`* or *`filenumber`*.

During recovery, all archived redo logs written to since the original data file was created must be applied to the new, empty version of the lost data file.

Oracle Database creates the new file in the same state as the old file when it was created. You must perform media recovery on the new file to return it to the state of the old file at the time it was lost.

Restrictions on Creating New Data Files

The creation of new data files is subject to the following restrictions:

- You cannot create a new file based on the first data file of the `SYSTEM` tablespace.
- You cannot specify the *`autoextend_clause`* of *`datafile_tempfile_spec`* in this `CREATE DATAFILE` clause.

> **See Also:**
>
> - "DATAFILE Clause" of CREATE DATABASE for information on the result of this clause if you do not specify a name for the new data file
> - *file_specification* for a full description of the file specification (*datafile_tempfile_spec*) and "Creating a New Data File: Example"

alter_datafile_clause

The DATAFILE clause lets you manipulate a file that you identify by name or by number. If you identify it by number, then *filenumber* is an integer representing the number found in the FILE# column of the V$DATAFILE dynamic performance view or in the FILE_ID column of the DBA_DATA_FILES data dictionary view. The DATAFILE clauses affect your database files as follows:

ONLINE

Specify ONLINE to bring the data file online.

OFFLINE

Specify OFFLINE to take the data file offline. If the database is open, then you must perform media recovery on the data file before bringing it back online, because a checkpoint is not performed on the data file before it is taken offline.

FOR DROP

If the database is in NOARCHIVELOG mode, then you must specify FOR DROP clause to take a data file offline. However, this clause does not remove the data file from the database. To do that, you must use an operating system command or drop the tablespace in which the data file resides. Until you do so, the data file remains in the data dictionary with the status RECOVER or OFFLINE.

If the database is in ARCHIVELOG mode, then Oracle Database ignores the FOR DROP clause.

RESIZE

Specify RESIZE if you want Oracle Database to attempt to increase or decrease the size of the data file to the specified absolute size in bytes. There is no default, so you must specify a size. You can also use this command to resize datafiles in shadow tablespaces, that store lost write data.

If sufficient disk space is not available for the increased size, or if the file contains data beyond the specified decreased size, then Oracle Database returns an error.

> **See Also:**
>
> "Resizing a Data File: Example"

END BACKUP

Specify `END BACKUP` to take the data file out of online backup mode. The `END BACKUP` clause is described more fully at the top level of the syntax of `ALTER DATABASE`. See "END BACKUP Clause".

ENCRYPT | DECRYPT

Use these clauses to perform offline encryption or decryption of the data file using Transparent Data Encryption (TDE). In any given tablespace, either all data files must be encrypted or all data files must be unencrypted.

Before issuing either of these clauses, the database must be mounted. The database can also be open, but the tablespace that contains the data file being encrypted or decrypted must be offline. The TDE master key must be loaded into database memory.

- Specify `ENCRYPT` to encrypt an unencrypted data file. The data file is encrypted using the `AES128` algorithm.
- Specify `DECRYPT` to decrypt a data file. The data file must have been previously encrypted with the `ALTER DATABASE DATAFILE` ... `ENCRYPT` statement.

Restrictions on Encrypting and Decrypting Data Files

The following restrictions apply to the `ENCRYPT` and `DECRYPT` clauses:

- You cannot encrypt or decrypt a temporary data file of a temporary tablespace. Instead, you must drop the temporary tablespace and recreate it as an encrypted tablespace.
- Oracle recommends against encrypting the data files of an undo tablespace. Doing so prevents the keystore from being closed, which prevents the database from functioning. Furthermore, this practice is unnecessary because all undo records that are associated with an encrypted tablespace are already automatically encrypted in the undo tablespace.

> **Note:**
>
> The use of the `ENCRYPT` or `DECRYPT` clause is only one step in a series of steps for performing offline encryption or decryption of a data file. Refer to *Transparent Data Encryption* for the complete set of steps before you use either of these clauses.

alter_tempfile_clause

Use the `TEMPFILE` clause to resize your temporary data file or specify the *`autoextend_clause`*, with the same effect as for a permanent data file. The database must be open. You can identify the temp file by name or by number. If you identify it by number, then *`filenumber`* is an integer representing the number found in the `FILE#` column of the `V$TEMPFILE` dynamic performance view.

> **Note:**
>
> On some operating systems, Oracle does not allocate space for a temp file until the temp file blocks are actually accessed. This delay in space allocation results in faster creation and resizing of temp files, but it requires that sufficient disk space is available when the temp files are later used. To avoid potential problems, before you create or resize a temp file, ensure that the available disk space exceeds the size of the new temp file or the increased size of a resized temp file. The excess space should allow for anticipated increases in disk space use by unrelated operations as well. Then proceed with the creation or resizing operation.

DROP

Specify `DROP` to drop *`tempfile`* from the database. The tablespace remains.

If you specify `INCLUDING DATAFILES`, then Oracle Database also deletes the associated operating system files and writes a message to the alert log for each such deleted file. You can achieve the same result using an `ALTER TABLESPACE` ... `DROP TEMPFILE` statement. Refer to the `ALTER TABLESPACE` DROP Clause for more information.

move_datafile_clause

Use the `MOVE DATAFILE` clause to move an online data file to a new location. The database can be open and accessing the data file when you perform this operation. The database creates a copy of the data file when it is performing this operation. Ensure that there is adequate disk space for the original data file and the copy before using this clause.

You can specify the original data file using the *`file_name`*, *`ASM_filename`*, or *`file_number`*. Refer to *ASM_filename* for information on ASM file names. If you identify the file by number, then *`file_number`* is an integer representing the number found in the `FILE#` column of the `V$DATAFILE` dynamic performance view or in the `FILE_ID` column of the `DBA_DATA_FILES` data dictionary view.

Use the `TO` clause to specify the new *`file_name`* or *`ASM_filename`*. If you are using Oracle Managed Files, then you can omit the `TO` clause. In this case, Oracle Database creates a unique name for the data file and saves it in the directory specified by the `DB_CREATE_FILE_DEST` initialization parameter.

If you specify `REUSE`, then the new data file is created even if it already exists.

If you specify `KEEP`, then the original data file will be kept after the `MOVE DATAFILE` operation. You cannot specify `KEEP` if the original data file is an Oracle Managed File. You can specify `KEEP` if the new data file is an Oracle Managed File.

autoextend_clause

Use the *`autoextend_clause`* to enable or disable the automatic extension of a new or existing data file or temp file. Refer to *file_specification* for information about this clause.

logfile_clauses

The logfile clauses let you add, drop, or modify log files.

ARCHIVELOG

Specify `ARCHIVELOG` if you want the contents of a redo log file group to be archived before the group can be reused. This mode prepares for the possibility of media recovery. Use this clause only after shutting down your instance normally, or immediately with no errors, and then restarting it and mounting the database.

MANUAL

Specify `MANUAL` to indicate that Oracle Database should create redo log files, but the archiving of the redo log files is controlled entirely by the user. This clause is provided for backward compatibility, for example for users who archive directly to tape. If you specify `MANUAL`, then:

- Oracle Database does not archive redo log files when a log switch occurs. You must handle this manually.
- You cannot have specified a standby database as an archivelog destinations. As a result, the database cannot be in `MAXIMUM PROTECTION` or `MAXIMUM AVAILABILITY` standby protection mode.

If you omit this clause, then Oracle Database automatically archives the redo log files to the destination specified in the `LOG_ARCHIVE_DEST_`*n* initialization parameters.

NOARCHIVELOG

Specify `NOARCHIVELOG` if you do not want the contents of a redo log file group to be archived so that the group can be reused. This mode does not prepare for recovery after media failure. Use this clause only if your instance has the database mounted but not open.

[NO] FORCE LOGGING

Use this clause to put the database into or take the database out of `FORCE LOGGING` mode. The database must be mounted or open.

In `FORCE LOGGING` mode, Oracle Database logs all changes in the database except changes in temporary tablespaces and temporary segments. This setting takes precedence over and is independent of any `NOLOGGING` or `FORCE LOGGING` settings you specify for individual tablespaces and any `NOLOGGING` settings you specify for individual database objects.

If you specify `FORCE LOGGING`, then Oracle Database waits for all ongoing unlogged operations to finish.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information on when to use `FORCE LOGGING` mode

SET STANDBY NOLOGGING

Standby nologging instructs the database to not log operations that qualify to be done without logging. The database sends the data blocks created by the operation to each qualifying standby database in the Data Guard configuration, to prevent missed data on the standby and keep it in sync with the primary.

Use this clause to determine how nonlogged tasks are handled . You can choose one of two logging modes for a database when you create the database, and you can change the logging mode of a database from one mode to the other.

- `SET STANDBY NOLOGGING FOR LOAD PERFORMANCE` to put the database 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 the load process slows, then the data is not sent but automatically fetched 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 database 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 or via block images in the redo, in case the network connection fails. 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.

Restrictions on Setting Standby Nologging

The`SET STANDBY NOLOGGING` clause cannot be used at the same time as `FORCE LOGGING`.

RENAME FILE Clause

This clause has the same function for logfiles that it has for data files and temp files. See "RENAME FILE Clause".

CLEAR LOGFILE Clause

Use the `CLEAR LOGFILE` clause to reinitialize an online redo log, optionally without archiving the redo log. `CLEAR LOGFILE` is similar to adding and dropping a redo log, except that the statement may be issued even if there are only two logs for the thread and may be issued for the current redo log of a closed thread.

For a standby database, if the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`, and if any of the log files are Oracle Managed Files, Oracle Database will create as many Oracle-managed log files as are in the control file. The log file members will reside in the current default log file destination.

- You must specify `UNARCHIVED` if you want to reuse a redo log that was not archived.

> **Note:**
>
> Specifying `UNARCHIVED` makes backups unusable if the redo log is needed for recovery.

- You must specify `UNRECOVERABLE DATAFILE` if you have taken the data file offline with the database in `ARCHIVELOG` mode (that is, you specified `ALTER DATABASE` ... `DATAFILE OFFLINE` without the `DROP` keyword), and if the unarchived log to be cleared is needed to recover the data file before bringing it back online. In this case, you must drop the data file and the entire tablespace once the `CLEAR LOGFILE` statement completes.

  Do not use `CLEAR LOGFILE` to clear a log needed for media recovery. If it is necessary to clear a log containing redo after the database checkpoint, then you must first perform incomplete media recovery. The current redo log of an open thread can be cleared. The current log of a closed thread can be cleared by switching logs in the closed thread.

  If the `CLEAR LOGFILE` statement is interrupted by a system or instance failure, then the database may hang. In this case, reissue the statement after the database is restarted. If the failure occurred because of I/O errors accessing one member of a log group, then that member can be dropped and other members added.

> **See Also:**
>
> "Clearing a Log File: Example"

add_logfile_clauses

Use these clauses to add redo log file groups to the database and to add new members to existing redo log file groups.

ADD LOGFILE Clause

Use the `ADD LOGFILE` clause to add one or more redo log file groups to the online redo log or standby redo log.

> **See Also:**
>
> - "LOGFILE Clause" of `CREATE DATABASE` for information on the result of this clause for Oracle Managed Files if you do not specify a name for the new log file group
> - "Adding Redo Log File Groups: Examples"
> - *Oracle Data Guard Concepts and Administration* for more information on standby redo logs

STANDBY

Use the `STANDBY` clause to add a redo log file group to the standby redo log. If you do not specify this clause, then a log file group is added to the online redo log.

INSTANCE

The `INSTANCE` clause is applicable only for Oracle Real Application Clusters (Oracle RAC) or Oracle RAC One Node databases. Specify the name of the instance for which you want to add a redo log file group. The instance name is a string of up to 80 characters. Oracle Database automatically uses the thread that is mapped to the specified instance. If no thread is mapped to the specified instance, then Oracle Database automatically acquires an available unmapped thread and assigns it to that instance. If you do not specify this clause, then Oracle Database executes the command as if you had specified the current instance. If the specified instance has no current thread mapping and there are no available unmapped threads, then Oracle Database returns an error.

THREAD

When adding a redo log file group to the standby redo log, use the `THREAD` clause to assign the log file group to a specific primary database redo thread. Query the `V$INSTANCE` view on the primary database to determine which redo threads have been opened, and specify one of these thread numbers.

You can also use the `THREAD` clause to assign a log file group to a specific redo thread when adding the log file group to the online redo log. This usage has been deprecated. The `INSTANCE` clause achieves the same purpose and is easier to use.

GROUP

The GROUP clause uniquely identifies the redo log file group among all groups in all threads and can range from 1 to the value specified for MAXLOGFILES in the CREATE DATABASE statement. You cannot add multiple redo log file groups having the same GROUP value. If you omit this parameter, then Oracle Database generates its value automatically. You can examine the GROUP value for a redo log file group through the dynamic performance view V$LOG.

redo_log_file_spec

Each *redo_log_file_spec* specifies a redo log file group containing one or more members (copies). If you do not specify a filename for the new log file, then Oracle Database creates Oracle Managed Files according to the rules described in the "LOGFILE Clause" of CREATE DATABASE.

> **See Also:**
>
> - *file_specification*
> - *Oracle Database Reference* for information on dynamic performance views

ADD LOGFILE MEMBER Clause

Use the ADD LOGFILE MEMBER clause to add new members to existing redo log file groups. Each new member is specified by *'filename'*. If the file already exists, then it must be the same size as the other group members and you must specify REUSE. If the file does not exist, then Oracle Database creates a file of the correct size. You cannot add a member to a group if all of the members of the group have been lost through media failure.

STANDBY

You must specify STANDBY when adding a member to a standby redo log file group. Otherwise, Oracle Database returns an error.

You can use the *logfile_descriptor* clause to specify an existing redo log file group in one of two ways:

GROUP *integer*

Specify the value of the GROUP parameter that identifies the redo log file group.

filename(s)

List all members of the redo log file group. You must fully specify each filename according to the conventions of your operating system.

> **See Also:**
>
> - "LOGFILE Clause" of CREATE DATABASE for information on the result of this clause for Oracle Managed Files if you do not specify a name for the new log file group
> - "Adding Redo Log File Group Members: Example"

drop_logfile_clauses

Use these clauses to drop redo log file groups or redo log file members.

DROP LOGFILE Clause

Use the `DROP LOGFILE` clause to drop all members of a redo log file group. If you use this clause to drop Oracle Managed Files, then Oracle Database also removes all log file members from disk. Specify a redo log file group as indicated for the `ADD LOGFILE MEMBER` clause.

- To drop the current log file group, you must first issue an `ALTER SYSTEM SWITCH LOGFILE` statement.
- You cannot drop a redo log file group if it needs archiving.
- You cannot drop a redo log file group if doing so would cause the redo thread to contain less than two redo log file groups.

> **See Also:**
>
> ALTER SYSTEM and "Dropping Log File Members: Example"

DROP LOGFILE MEMBER Clause

Use the `DROP LOGFILE MEMBER` clause to drop one or more redo log file members. Each *'filename'* must fully specify a member using the conventions for filenames on your operating system.

- To drop a log file in the current log, you must first issue an `ALTER SYSTEM SWITCH LOGFILE` statement. Refer to ALTER SYSTEM for more information.
- You cannot use this clause to drop all members of a redo log file group that contains valid data. To perform that operation, use the `DROP LOGFILE` clause.

> **See Also:**
>
> "Dropping Log File Members: Example"

switch_logfile_clause

This clause is useful when you are migrating the database to disks with a different block size that the block size of the current database. Use this clause to switch logfiles to a different block size for all externally enabled threads, including both open and closed threads. If you are migrating the database to use 4KB sector disks, then you must specify 4096 for *integer*. If you are unmigrating the database back to using 512B sector disks, then you must specify 512 for *integer*.

This clause is an extension of the existing `ALTER SYSTEM SWITCH LOGFILE` statement. That statement switches logs for a single thread. This clause switches logfiles for all externally enabled threads, including both open and closed threads.

Before using this clause, you must already have created at least two redo log groups with the same target block size on the migration target disk.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information on migrating the database to disks with a different block size, and "Adding a Log File: Example"

supplemental_db_logging

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

ADD SUPPLEMENTAL LOG Clause

Specify `ADD SUPPLEMENTAL LOG DATA` to enable **minimal supplemental logging**. Specify `ADD SUPPLEMENTAL LOG` *`supplemental_id_key_clause`* to enable column data logging in addition to minimal supplemental logging. Specify `ADD SUPPLEMENTAL LOG` *`supplemental_plsql_clause`* to enable supplemental logging of PL/SQL calls. Oracle Database does not enable either minimal supplemental logging or supplemental logging by default.

Minimal supplemental logging ensures that LogMiner (and any products building on LogMiner technology) will have sufficient information to support chained rows and various storage arrangements such as cluster tables.

If the redo generated on one database is to be the source of changes (to be mined and applied) at another database, as is the case with logical standby, then the affected rows need to be identified using column data (as opposed to rowids). In this case, you should specify the *`supplemental_id_key_clause`*.

You can query the appropriate columns in the `V$DATABASE` view to determine whether any supplemental logging has already been enabled.

You can use this clause when the database is open. However, Oracle Database will invalidate all DML cursors in the cursor cache, which will have an effect on performance until the cache is repopulated.

If you use this clause in a CDB, then the current container must be the root and the operation will be performed on the entire CDB.

You can enable supplemental logging levels at PDB without minimal supplemental logging enabled at `CDB$ROOT`. Dropping all supplemental logging from `CDB$ROOT` will not disable supplemental logging enabled at the PDB level.

For a full discussion of the *`supplemental_id_clause`*, refer to supplemental_id_key_clause in the documentation on `CREATE TABLE`.

> **See Also:**
>
> - *Oracle Data Guard Concepts and Administration* for information on supplemental logging on the primary database to support a logical standby database
> - *Oracle Database Utilities* for examples using the *`supplemental_db_logging`* clause syntax

DROP SUPPLEMENTAL LOG Clause

Use this clause to stop supplemental logging.

- Specify `DROP SUPPLEMENTAL LOG DATA` to instruct Oracle Database to stop placing minimal additional log information into the redo log stream whenever an update operation occurs. If Oracle Database is doing column data supplemental logging specified with the *`supplemental_id_key_clause`*, then you must first stop the column data supplemental logging with the `DROP SUPPLEMENTAL LOG` *`supplemental_id_key_clause`* and then specify this clause.
- Specify `DROP SUPPLEMENTAL LOG` *`supplemental_id_key_clause`* to drop some or all of the system-generated supplemental log groups. You must specify the *`supplemental_id_key_clause`* if the supplemental log groups you want to drop were added using that clause.
- Specify `DROP SUPPLEMENTAL LOG` *`supplemental_plsql_clause`* disable supplemental logging of PL/SQL calls.

If you use this clause in a CDB, then the current container must be the root and the operation will be performed on the entire CDB.

ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION of `ALTER DATABASE` enables low impact minimal supplemental logging.

- You can execute this DDL only when the *`enable_goldengate_replication`* parameter is TRUE, and database compatible is 19.0 or higher.
- This DDL implicitly adds DB-level minimal supplemental logging, similar to other DB-level supplemental logging DDLs.
- In case of CDB, this DDL can be executed in both `CDB$ROOT` and pluggable databases.
- When executed in `CDB$ROOT`, it enables low impact minimal supplemental logging for entire database. Low impact minimal supplemental logging will be enabled for all the pluggable databases regardless of the PDB level setting for subset database replication.
- When executed in pluggable database, it's same as `ALTER PLUGGABLE DATABASE ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION`. See *ALTER PLUGGABLE DATABASE* for details.

DROP SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION of `ALTER DATABASE` disables low impact minimal supplemental logging.

- You can execute this DDL only when the*`enable_goldengate_replication`* parameter is TRUE, and database compatible should be 19.0 or higher.
- You must have explicitly enabled other supplemental log data. This restriction ensures that disabling low impact minimal supplemental logging never disables minimal supplemental logging.
- Once this DDL is executed, the minimal supplemental logging will go back to its current behavior.
- In case of CDB, this DDL can be executed in both `CDB$ROOT` and pluggable databases.
- When executed in `CDB$ROOT` , it disables low impact minimal supplemental logging at the database level. For each pluggable database, whether low impact supplemental logging is enabled depends on the PDB-level setting for subset database replication.
- When executed in pluggable database, the behavior is the same as `ALTER PLUGGABLE DATABASE DROP SUPPLEMENTAL LOG DATA SUBSET DATABASE REPLICATION`. See *ALTER PLUGGABLE DATABASE* for details.

See Also:

Oracle Data Guard Concepts and Administration for information on supplemental logging

controlfile_clauses

The *controlfile_clauses* let you create or back up a control file.

CREATE CONTROLFILE Clause

The CREATE CONTROLFILE clause lets you create a control file.

- Specify PHYSICAL STANDBY to create a control file to be used to maintain a physical database. This is the default if you specify STANDBY and do not specify PHYSICAL or LOGICAL.
- Specify LOGICAL STANDBY to create a control file to be used to maintain a logical database.
- Specify FAR SYNC INSTANCE to create a control file to be used to maintain a Data Guard far sync instance.

If the file already exists, then you must specify REUSE. In an Oracle RAC environment, the control file must be on shared storage.

See Also:

Oracle Data Guard Concepts and Administration for more information on creating control files

BACKUP CONTROLFILE Clause

Use the BACKUP CONTROLFILE clause to back up the current control file. The database must be open or mounted when you specify this clause.

TO '*filename*'

Use this clause to specify a binary backup of the control file. You must fully specify the *filename* using the conventions for your operating system. If the specified file already exists, then you must specify REUSE. In an Oracle RAC environment, *filename* must be on shared storage.

A binary backup contains information that is not captured if you specify TO TRACE, such as the archived log history, offline range for read-only and offline tablespaces, and backup sets and copies (if you use RMAN). If the COMPATIBLE initialization parameter is 10.2 or higher, binary control file backups include temp file entries.

TO TRACE

Specify TO TRACE if you want Oracle Database to write SQL statements to a trace file rather than making a physical backup of the control file. You can use SQL statements written to the trace file to start up the database, re-create the control file, and recover and open the database appropriately, based on the created control file. If you issue an ALTER DATABASE BACKUP CONTROLFILE TO TRACE statement while block change tracking is enabled, then the resulting trace file will contain a command to reenable block change tracking.

This statement issues an implicit ALTER DATABASE REGISTER LOGFILE statement, which creates incarnation records if the archived log files reside in the current archivelog destinations.

The trace file will also include ALTER DATABASE REGISTER LOGFILE statements for existing logfiles that reside in the current archivelog destinations. This will implicitly create database incarnation records for the branches of redo to which the logfiles apply.

You can copy the statements from the trace file into a script file, edit the statements as necessary, and use the script if all copies of the control file are lost (or to change the size of the control file).

- Specify AS *filename* if you want Oracle Database to place the trace output into a file called *filename* rather than into the standard trace file.
- Specify REUSE to allow Oracle Database to overwrite any existing file called *filename*.
- RESETLOGS indicates that the SQL statement written to the trace file for starting the database is ALTER DATABASE OPEN RESETLOGS. This setting is valid only if the online logs are unavailable.
- NORESETLOGS indicates that the SQL statement written to the trace file for starting the database is ALTER DATABASE OPEN NORESETLOGS. This setting is valid only if all the online logs are available.

If you cannot predict the future state of the online logs, then specify neither RESETLOGS nor NORESETLOGS. In this case, Oracle Database puts both versions of the script into the trace file, and you can choose which version is appropriate when the script becomes necessary.

The trace files are stored in a subdirectory determined by the DIAGNOSTIC_DEST initialization parameter. You can find the name and location of the trace file to which the CREATE CONTROLFILE statements were written by looking in the alert log. You can also find the directory for trace files by querying the NAME and VALUE columns of the V$DIAG_INFO dynamic performance view.

See Also:

Oracle Database Administrator's Guide for information on viewing the alert log

standby_database_clauses

Use these clauses to activate the standby database or to specify whether it is in protected or unprotected mode.

See Also:

Oracle Data Guard Concepts and Administration for descriptions of the physical and logical standby database and for information on maintaining and using standby databases

activate_standby_db_clause

Use the ACTIVATE STANDBY DATABASE clause to convert a standby database into a primary database.

> **Note:**
>
> Before using this command, refer to *Oracle Data Guard Concepts and Administration* for important usage information.

PHYSICAL

Specify `PHYSICAL` to activate a physical standby database. This is the default.

LOGICAL

Specify `LOGICAL` to activate a logical standby database. If you have more than one logical standby database, then you should first ensure that the same log data is available on all the standby systems.

FINISH APPLY

This clause applies only to logical standby databases. Use it to initiate **terminal apply**, which is the application of any remaining redo to bring the logical standby database to the same state as the primary database. When terminal apply is complete, the database completes the switchover from logical standby to primary database.

If you require immediate restoration of the database in spite of data loss, then omit this clause. The database will execute the switchover from logical standby to primary database immediately without terminal apply.

maximize_standby_db_clause

Use this clause to specify the level of protection for the data in your database environment. You specify this clause from the primary database.

> **Note:**
>
> The `PROTECTED` and `UNPROTECTED` keywords have been replaced for clarity but are still supported. `PROTECTED` is equivalent to `TO MAXIMIZE PROTECTION`. `UNPROTECTED` is equivalent to `TO MAXIMIZE PERFORMANCE`.

TO MAXIMIZE PROTECTION

This setting establishes **maximum protection mode** and offers the highest level of data protection. A transaction does not commit until all data needed to recover that transaction has been written to at least one physical standby database that is configured to use the `SYNC` log transport mode. If the primary database is unable to write the redo records to at least one such standby database, then the primary database is shut down. This mode guarantees zero data loss, but it has the greatest potential impact on the performance and availability of the primary database.

Restriction on Establishing Maximum Protection Mode

You can specify `TO MAXIMIZE PROTECTION` on an open database only if the current data protection mode is `MAXIMUM AVAILABILITY` and there is at least one synchronized standby database.

TO MAXIMIZE AVAILABILITY

This setting establishes **maximum availability mode** and offers the next highest level of data protection. A transaction does not commit until all data needed to recover that transaction has been written to at least one physical or logical standby database that is configured to use the SYNC log transport mode. Unlike maximum protection mode, the primary database does not shut down if it is unable to write the redo records to at least one such standby database. Instead, the protection is lowered to maximum performance mode until the fault has been corrected and the standby database has caught up with the primary database. This mode guarantees zero data loss unless the primary database fails while in maximum performance mode. Maximum availability mode provides the highest level of data protection that is possible without affecting the availability of the primary database.

TO MAXIMIZE PERFORMANCE

This setting establishes **maximum performance mode** and is the default setting. A transaction commits before the data needed to recover that transaction has been written to a standby database. Therefore, some transactions may be lost if the primary database fails and you are unable to recover the redo records from the primary database. This mode provides the highest level of data protection that is possible without affecting the performance of the primary database.

To determine the current mode of the database, query the PROTECTION_MODE column of the V$DATABASE dynamic performance view.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for full information on using these standby database settings

register_logfile_clause

Specify the REGISTER LOGFILE clause from the standby database to manually register log files from the failed primary. Use the *redo_log_file_spec* form of *file_specification* (see *file_specification*) to list regular redo log files in an operating system file system or to list Oracle ASM disk group redo log files.

When a log file is from an unknown incarnation, the REGISTER LOGFILE clause causes an incarnation record to be added to the V$DATABASE_INCARNATION view. If the newly registered log file belongs to an incarnation having a higher RESETLOGS_TIME than the current RECOVERY_TARGET_INCARNATION#, then the REGISTER LOGFILE clause also causes RECOVERY_TARGET_INCARNATION# to be changed to correspond to the newly added incarnation record.

OR REPLACE

Specify OR REPLACE to allow an existing archivelog entry in the standby database to be updated, for example, when its location or file specification changes. The system change numbers of the entries must match exactly, and the original entry must have been created by the managed standby log transmittal mechanism.

FOR *logminer_session_name*

This clause is useful in a Streams environment. It lets you register the log file with one specified LogMiner session.

switchover_clause

> **Caution:**
>
> Before using this command, refer to *Oracle Data Guard Concepts and Administration* for complete usage information.

Use this clause to perform a switchover to a physical standby database. Specify this clause from the primary database. For *target_db_name*, specify the DB_UNIQUE_NAME of the standby database.

VERIFY

Use this clause to verify that a physical standby database is ready for a switchover. Specify this clause from the primary database. For *target_db_name*, specify the DB_UNIQUE_NAME of the standby database. If the standby database is ready for a switchover, then the "Database Altered" message is returned. Otherwise, an error message that will assist you in preparing the standby database for a switchover is returned.

FORCE

Use this clause if a previous switchover command failed and created a configuration with no primary database. Specify this clause from the physical standby database that you want to convert to the primary database. For *target_db_name*, specify the DB_UNIQUE_NAME of the database that you want to convert to the primary database.

failover_clause

> **Caution:**
>
> Before using this command, refer to *Oracle Data Guard Concepts and Administration* for complete usage information.

Use this clause to perform a failover to a physical standby database. Specify this clause from the standby database. For *target_db_name*, specify the DB_UNIQUE_NAME of the standby database.

FORCE

This clause has meaning only when the failover target is serviced by a Data Guard far sync instance. Use this clause when a previous failover command failed and the reason for the failure cannot be resolved. It instructs the failover to ignore any failures encountered when interacting with the Data Guard far sync instance and proceed with the failover, if at all possible.

commit_switchover_clause

Use this clause to perform database role transitions in a Data Guard configuration.

> **Caution:**
>
> Before using this command, refer to *Oracle Data Guard Concepts and Administration* for complete usage information.

PREPARE TO SWITCHOVER

This clause prepares a primary database to become a logical standby database or a logical standby database to become a primary database.

- Specify `PREPARE TO SWITCHOVER TO LOGICAL STANDBY` on a primary database.
- Specify `PREPARE TO SWITCHOVER TO PRIMARY DATABASE` on a logical standby database.

COMMIT TO SWITCHOVER

This clause switches a primary database to a standby database role or switches a standby database to the primary database role.

- Specify `COMMIT TO SWITCHOVER TO PHYSICAL STANDBY` or `COMMIT TO SWITCHOVER TO LOGICAL STANDBY` on a primary database.
- Specify `COMMIT TO SWITCHOVER TO PRIMARY DATABASE` on a standby database.

PHYSICAL

This clause is always optional. Use of this clause with the `COMMIT TO SWITCHOVER TO PRIMARY` clause has been deprecated.

LOGICAL

This clause is specified with the `PREPARE TO SWITCHOVER` or `COMMIT TO SWITCHOVER` clauses when switching a primary database to the logical standby database role. Use of this clause with the `COMMIT TO SWITCHOVER TO PRIMARY` clause has been deprecated.

WITH SESSION SHUTDOWN

This clause causes all database sessions to be closed and uncommitted transactions to be rolled back before performing a database role transition.

WITHOUT SESSION SHUTDOWN

This clause prevents a requested role transition from occurring if there are any database sessions. This is the default.

WAIT

Specify this clause to wait for a role transition to complete before returning control to the user.

NOWAIT

Specify this clause to return control to the user without waiting for a role transition to complete. This is the default.

CANCEL

Specify this clause to reverse the effect of a previously specified `PREPARE TO SWITCHOVER` statement.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for full information on switchover between primary and standby databases

start_standby_clause

Specify the `START LOGICAL STANDBY APPLY` clause to begin applying redo logs to a logical standby database. This clause enables primary key, unique index, and unique constraint supplemental logging as well as PL/SQL call logging.

- Specify `IMMEDIATE` to apply redo data from the current standby redo log file.
- Specify `NODELAY` if you want Oracle Database to ignore a delay for this apply. This is useful if the primary database is no longer present, which would otherwise require a PL/SQL call to be made.
- Specify `INITIAL` the first time you apply the logs to the standby database.
- The `NEW PRIMARY` clause is needed in two situations:
  - On a failover to a logical standby, specify this clause on a logical standby not participating in the failover operation, and on the old primary database after it has been reinstated as a logical standby database.
  - During a rolling upgrade using a logical standby database (which uses an unprepared switchover operation), specify this clause after the original primary database has been upgraded to the new database software.
- Specify `SKIP FAILED` [`TRANSACTION`] to skip the last transaction in the events table and restart the apply.
- Specify `FINISH` to force the standby redo logfile information into archived logs. If the primary database becomes disabled, then you can then apply the data in the redo log files.

stop_standby_clause

Use this clause to stop the log apply services. This clause applies only to logical standby databases, not to physical standby databases. Use the `STOP` clause to stop the apply in an orderly fashion.

convert_database_clause

Use this clause to convert a database from one form to another.

- Specify `CONVERT TO PHYSICAL STANDBY` to convert a primary database, a logical standby database, or a snapshot standby database into a physical standby database.

  Perform these steps before specifying this clause:

  - On an Oracle Real Application Clusters (Oracle RAC) database, shut down all but one instance.
  - Ensure that the database is mounted, but not open.

  The database is dismounted after conversion and must be restarted.
- Specify `CONVERT TO SNAPSHOT STANDBY` to convert a physical standby database into a snapshot standby database.

  Ensure that redo apply is stopped before specifying this clause.

Note:

A snapshot standby database must be opened at least once in read/write mode before it can be converted into a physical standby database.

See Also:

Oracle Data Guard Concepts and Administration for more information about standby databases

default_settings_clauses

Use these clauses to modify the default settings of the database.

DEFAULT EDITION Clause

Use this clause to designate the specified edition as the default edition for the database. The specified edition must already have been created and must be `USABLE`. The change takes place immediately and is visible to all nodes in an Oracle RAC environment. New database sessions automatically start out in the specified edition. The new setting persists across database shutdown and startup.

When you designate an edition as the database default edition, all users can use the edition, as though the `USE` object privilege were granted on the specified edition to the role `PUBLIC`.

You can determine the current default edition of the database with the following query:

```
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES
  WHERE PROPERTY_NAME = 'DEFAULT_EDITION';
```

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`

CHARACTER SET, NATIONAL CHARACTER SET

You can no longer change the database character set or the national character set using the `ALTER DATABASE` statement. Refer to *Oracle Database Globalization Support Guide* for information on database character set migration.

SET DEFAULT TABLESPACE Clause

Use this clause to specify or change the default type of subsequently created tablespaces. Specify `BIGFILE` or `SMALLFILE` to indicate whether the tablespaces should be bigfile or smallfile tablespaces.

- A **bigfile tablespace** contains only one data file or temp file, which can contain up to approximately 4 billion (2^{32}) blocks. The maximum size of the single data file or temp file is 128 terabytes (TB) for a tablespace with 32K blocks and 32TB for a tablespace with 8K blocks.

- A **smallfile tablespace** is a traditional Oracle tablespace, which can contain 1022 data files or temp files, each of which can contain up to approximately 4 million (2^{22}) blocks.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about bigfile tablespaces
> - "Setting the Default Type of Tablespaces: Example"

DEFAULT TABLESPACE Clause

Specify this clause to establish or change the default permanent tablespace of the database. The tablespace you specify must already have been created. After this operation completes, Oracle Database automatically reassigns to the new default tablespace all non-SYSTEM users. All objects subsequently created by those users will by default be stored in the new default tablespace. If you are replacing a previously specified default tablespace, then you can move the previously created objects from the old to the new default tablespace, and then drop the old default tablespace if you want to.

DEFAULT [LOCAL] TEMPORARY TABLESPACE Clause

Specify this clause to change the default shared temporary tablespace of the database to a new tablespace or tablespace group, or to change the default local temporary tablespace to a new tablespace.

- Specify *tablespace* to indicate the new default temporary tablespace for the database. After this operation completes, Oracle Database automatically reassigns to the new default temporary tablespace all users who had been assigned to the old default temporary tablespace. You can then drop the old default temporary tablespace if you want to. Specify DEFAULT TEMPORARY TABLESPACE to change the default shared temporary tablespace. Specify DEFAULT LOCAL TEMPORARY TABLESPACE to change the default local temporary tablespace.
- Specify *tablespace_group_name* to indicate that all tablespaces in the tablespace group specified by *tablespace_group_name* are now default shared temporary tablespaces for the database. After this operation completes, users who have not been explicitly assigned a default temporary tablespace can create temporary segments in any of the tablespaces that are part of *tablespace_group_name*. You cannot drop an old default temporary tablespace if it is part of the default temporary tablespace group. Local temporary tablespaces cannot be part of a tablespace group.

To learn the name of the current default temporary tablespace or default temporary tablespace group, query the TEMPORARY_TABLESPACE column of the ALL_, DBA_, or USER_USERS data dictionary views.

Restrictions on Default Temporary Tablespaces

Default temporary tablespaces are subject to the following restrictions:

- The tablespace you assign or reassign as the default temporary tablespace must have a standard block size.
- If the SYSTEM tablespace is locally managed, then the tablespace you specify as the default temporary tablespace must also be locally managed.

See Also:

- *Oracle Database Administrator's Guide* for information on tablespace groups
- "Changing the Default Temporary Tablespace: Examples"

instance_clauses

In an Oracle Real Application Clusters environment, specify `ENABLE INSTANCE` to enable the thread that is mapped to the specified database instance. The thread must have at least two redo log file groups, and the database must be open.

Specify `DISABLE INSTANCE` to disable the thread that is mapped to the specified database instance. The name of the instance is a string of up to 80 characters. If no thread is currently mapped to the specified instance, then Oracle Database returns an error. The database must be open, but you cannot disable a thread if an instance using it has the database mounted.

See Also:

Oracle Real Application Clusters Administration and Deployment Guide for more information on enabling and disabling instances

RENAME GLOBAL_NAME Clause

Specify `RENAME GLOBAL_NAME` to change the global name of the database. The database must be open. The *`database`* is the new database name and can be as long as eight bytes. The optional *`domain`* specifies where the database is effectively located in the network hierarchy. If you specify a domain name, then the components of the domain name must be legal identifiers. See "Database Object Naming Rules " for information on valid identifiers.

Note:

Renaming your database does not change global references to your database from existing database links, synonyms, and stored procedures and functions on remote databases. Changing such references is the responsibility of the administrator of the remote databases.

See Also:

"Changing the Global Database Name: Example"

BLOCK CHANGE TRACKING Clauses

The **block change tracking** feature causes Oracle Database to keep track of the physical locations of all database updates on both the primary database and any physical standby database. You must enable block change tracking on each database for which you want tracking to be performed. The tracking information is maintained in a separate file called the block change tracking file. If you are using Oracle Managed Files, then Oracle Database automatically creates the block change tracking file in the location specified by DB_CREATE_FILE_DEST. If you are not using Oracle Managed Files, then you must specify the change tracking filename. Oracle Database uses change tracking data for some internal tasks, such as increasing the performance of incremental backups. You can enable or disable block change tracking with the database either open or mounted, in either archivelog or NOARCHIVELOG mode.

ENABLE BLOCK CHANGE TRACKING

This clause enables block change tracking and causes Oracle Database to create a block change tracking file.

- Specify USING FILE '*filename*' if you want to name the block change tracking file instead of letting Oracle Database generate a name for it. You must specify this clause if you are not using Oracle Managed Files.
- Specify REUSE to allow Oracle Database to overwrite an existing block change tracking file of the same name.

> **Note:**
>
> On a standby database, the block change tracking only becomes effective when the managed recovery starts. If the recovery is already active on the standby database when you issue the command, the recovery must be stopped and then started again to benefit from the fast incremental backups.

DISABLE BLOCK CHANGE TRACKING

Specify this clause if you want Oracle Database to stop tracking changes and delete the existing block change tracking file.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for information on setting up block change tracking and "Enabling and Disabling Block Change Tracking: Examples"

[NO] FORCE FULL DATABASE CACHING

Use this clause to enable or disable the force full database caching mode. In contrast to the default mode, which is automatic, the force full database caching mode considers the entire database, including NOCACHE LOBs, as eligible for caching in the buffer cache.

The database must be mounted but not open. In an Oracle RAC environment, the database must be mounted but not open in the current instance and unmounted in all other instances.

- Specify FORCE FULL DATABASE CACHING to enable the force full database caching mode.
- Specify NO FORCE FULL DATABASE CACHING to disable the force full database caching mode. This is the default mode.

You can determine whether the force full database caching mode is enabled by querying the FORCE_FULL_DB_CACHING column of the V$DATABASE dynamic performance view.

> **See Also:**
>
> - *Oracle Database Concepts* for more information on the force full database caching mode
> - *Oracle Database Administrator's Guide* to learn how to enable the force full database caching mode
> - *Oracle Database Reference* for more information on the `V$DATABASE` dynamic performance view

CONTAINERS DEFAULT TARGET

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

- For *`container_name`*, specify the name of the default container. The default container can be any container in the CDB, including the CDB root, a PDB, an 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 a CDB root without specifying containers in the `WHERE` clause, the DML statement affects the default container for the CDB.

flashback_mode_clause

Use this clause to put the database in or take the database out of `FLASHBACK` mode. You can specify this clause only if the database is in `ARCHIVELOG` mode and you have already prepared a fast recovery area for the database. You can specify this clause when the database is mounted or open. This clause cannot be specified on a physical standby database if redo apply is active.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for information on preparing the fast recovery area for Flashback operations

FLASHBACK ON

Use this clause to put the database in `FLASHBACK` mode. When the database is in `FLASHBACK` mode, Oracle Database automatically creates and manages Flashback Database logs in the fast recovery area. Users with `SYSDBA` system privilege can then issue a `FLASHBACK DATABASE` statement.

FLASHBACK OFF

Use this clause to take the database out of `FLASHBACK` mode. Oracle Database stops logging Flashback data and deletes all existing Flashback Database logs. Any attempt to issue a `FLASHBACK DATABASE` will fail with an error.

undo_mode_clause

This clause is valid only when you are connected to a CDB. It lets you change the undo mode for the CDB. The CDB must be in `OPEN UPGRADE` mode.

- Specify `LOCAL UNDO ON` to change the CDB to use local undo mode.
- Specify `LOCAL UNDO OFF` to change the CDB to use shared undo mode.

> **See Also:**
>
> - `CREATE DATABASE` *undo_mode_clause* for the full semantics of this clause
> - *Oracle Database Administrator's Guide* for the complete steps for configuring a CDB to use local undo mode or shared undo mode

set_time_zone_clause

This clause has the same semantics in `CREATE DATABASE` and `ALTER DATABASE` statements. When used in with `ALTER DATABASE`, this clause resets the time zone of the database. To determine the time zone of the database, query the built-in function DBTIMEZONE . After setting or changing the time zone with this clause, you must restart the database for the new time zone to take effect.

Oracle Database normalizes all new `TIMESTAMP WITH LOCAL TIME ZONE` data to the time zone of the database when the data is stored on disk.Oracle Database does not automatically update existing data in the database to the new time zone. Therefore, you cannot reset the database time zone if there is any `TIMESTAMP WITH LOCAL TIME ZONE` data in the database. You must first delete or export the `TIMESTAMP WITH LOCAL TIME ZONE` data and then reset the database time zone. For this reason, Oracle does not encourage you to change the time zone of a database that contains data.

For a full description of this clause, refer to *set_time_zone_clause* in the documentation on `CREATE DATABASE`.

security_clause

Use the *`security_clause`* (`GUARD`) to protect data in the database from being changed. You can override this setting for a current session using the `ALTER SESSION DISABLE GUARD` statement. Refer to ALTER SESSION for more information.

ALL

Specify `ALL` to prevent all users other than `SYS` from making any changes to the database.

STANDBY

Specify `STANDBY` to prevent all users other than `SYS` from making changes to any database object being maintained by logical standby. This setting is useful if you want report operations to be able to modify data as long as it is not being replicated by logical standby.

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration* for information on logical standby

NONE

Specify `NONE` if you want normal security for all data in the database.

> **Note:**
>
> Oracle strongly recommends that you not use this setting on a logical standby database.

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 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.

Prepare a Database : Example

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

drop_mirror_copy

Use this clause to discard mirror copies of data 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

Specify this clause to enable lost write protection for data files. You can enable, remove, and suspend lost write protection for data files.

Example: Turn on Lost Write for a Datafile

The example turns on lost write on datafile `td_file.df`.

```
 ALTER DATABASE DATAFILE td_file.df ENABLE LOST WRITE PROTECTION
```

Note that the lost write database is zeroed out. It is not initialized with the contents of the current data file.

You can turn off lost write protection for a datafile in two ways, with the `REMOVE` or `SUSPEND` options.

1. The `REMOVE` option stops lost write protection for the data file. Additionally, it removes all references to lost write protection including tracking data from the shadow tablespace.

Example: Remove Lost Write for a Datafile

```
ALTER DATABASE DATAFILE td_file.df REMOVE LOST WRITE PROTECTION
```

2. The SUSPEND option disables updates and lost write checking, but leaves the tracking data in the shadow tablespace. If you suspend lost write protection for a short time, lost write protection for the data file is stopped during the suspended period. This means that no lost write data is gathered, and no blocks are checked. If you turn on lost write protection for the data file later, there will be no records of SCN updates made to the blocks in the datafile during the suspended period. Note that the SUSPEND option does not deallocate the lost write storage.

   Example: Suspend Lost Write for a Datafile

   ```
   ALTER DATABASE DATAFILE td_file.df SUSPEND LOST WRITE PROTECTION
   ```

You can enable lost write protection for container databases and pluggable databases.

Example: Turn on Lost Write for a Database

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

Example: Turn off Lost Write for a Database

```
ALTER 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.

cdb_fleet_clauses

Specify the *cdb_fleet_clauses* to set a Lead CDB in a collection of different CDBs.

lead_cdb_clause

Use this clause to designate a CDB as the Lead CDB in a CDB fleet. The database property LEAD_CDB indicates that the current CDB is a Lead CDB, and can be found in DATABASE_PROPERTIES view.

There is a new parameter in SYS_CONTEXT named IS_LEAD_CDB which can be used to determine if the current session is connected to a Lead CDB in a CDB fleet.

lead_cdb_uri_clause

Use this clause to specify the connection URI for the Lead CDB in a CDB fleet. It is used to register a Member CDB with the Lead CDB of the fleet.

The database link name specified in dblink must exist in the CDB ROOT of the Member CDB joining the CDB fleet. It is used to synchronize PDB metadata with the Lead CDB in the fleet.

The uri_string specified is stored as a database property named LEAD_CDB_URI and can be found in DATABASE_PROPERTIES view.

There is a new parameter in SYS_CONTEXT named IS_LEAD_CDB which can be used to determine if the current session is connected to a Member CDB in a CDB fleet.

property_clause

Specify this clause to set or remove database properties visible through DATABASE_PROPERTIES or CDB_PROPERTIES views.

replay_upgrade_clause

Use this clause to enable or disable replay upgrade on the database.

If UPGRADE SYNC is ON, then replay upgrade and upgrade on open is enabled.

Examples

READ ONLY / READ WRITE: Example

The following statement opens the database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

The following statement opens the database in read/write mode and clears the online redo logs:

```
ALTER DATABASE OPEN READ WRITE RESETLOGS;
```

Using Parallel Recovery Processes: Example

The following statement performs tablespace recovery using parallel recovery processes:

```
ALTER DATABASE
   RECOVER TABLESPACE tbs_03
   PARALLEL;
```

Adding Redo Log File Groups: Examples

The following statement adds a redo log file group with two members and identifies it with a GROUP parameter value of 3:

```
ALTER DATABASE
  ADD LOGFILE GROUP 3
    ('diska:log3.log' ,
     'diskb:log3.log') SIZE 50K;
```

The following statement adds a redo log file group containing two members to thread 5 (in a Real Application Clusters environment) and assigns it a GROUP parameter value of 4:

```
ALTER DATABASE
    ADD LOGFILE THREAD 5 GROUP 4
        ('diska:log4.log',
         'diskb:log4:log');
```

Adding Redo Log File Group Members: Example

The following statement adds a member to the redo log file group added in the previous example:

```
ALTER DATABASE
   ADD LOGFILE MEMBER 'diskc:log3.log'
   TO GROUP 3;
```

Dropping Log File Members: Example

The following statement drops one redo log file member added in the previous example:

```
ALTER DATABASE
    DROP LOGFILE MEMBER 'diskb:log3.log';
```

The following statement drops all members of the redo log file group 3:

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

Renaming a Log File Member: Example

The following statement renames a redo log file member:

```
ALTER DATABASE
    RENAME FILE 'diskc:log3.log' TO 'diskb:log3.log';
```

The preceding statement only changes the member of the redo log group from one file to another. The statement does not actually change the name of the file diskc:log3.log to diskb:log3.log. Before issuing this statement, you must change the name of the file through your operating system.

Setting the Default Type of Tablespaces: Example

The following statement specifies that subsequently created tablespaces be created as bigfile tablespaces by default:

```
ALTER DATABASE
    SET DEFAULT BIGFILE TABLESPACE;
```

Changing the Default Temporary Tablespace: Examples

The following statement makes the tbs_05 tablespace (created in "Creating a Temporary Tablespace: Example") the default temporary tablespace of the database. This statement either establishes a default temporary tablespace if none was specified at create time, or replaces an existing default temporary tablespace with tbs_05:

```
ALTER DATABASE
   DEFAULT TEMPORARY TABLESPACE tbs_05;
```

Alternatively, a group of tablespaces can be defined as the default temporary tablespace by using a tablespace group. The following statement makes the tablespaces in the tablespace group tbs_group_01 (created in "Adding a Temporary Tablespace to a Tablespace Group: Example") the default temporary tablespaces of the database:

```
ALTER DATABASE
   DEFAULT TEMPORARY TABLESPACE tbs_grp_01;
```

Creating a New Data File: Example

The following statement creates a new data file tbs_f04.dbf based on the file tbs_f03.dbf. Before creating the new data file, you must take the existing data file (or the tablespace in which it resides) offline.

```
ALTER DATABASE
    CREATE DATAFILE 'tbs_f03.dbf'
                 AS 'tbs_f04.dbf';
```

Manipulating Temp Files: Example

The following takes offline the temp file temp02.dbf created in Adding and Dropping Data Files and Temp Files: Examples and then renames the temp file:

```
ALTER DATABASE TEMPFILE 'temp02.dbf' OFFLINE;

ALTER DATABASE RENAME FILE 'temp02.dbf' TO 'temp03.dbf';
```

The statement renaming the temp file requires that you first create the file temp03.dbf on the operating system.

Changing the Global Database Name: Example

The following statement changes the global name of the database and includes both the database name and domain:

```
ALTER DATABASE
    RENAME GLOBAL_NAME TO demo.world.example.com;
```

Enabling and Disabling Block Change Tracking: Examples

The following statement enables block change tracking and causes Oracle Database to create a block change tracking file named `tracking_file` and overwrite the file if it already exists:

```
ALTER DATABASE
  ENABLE BLOCK CHANGE TRACKING
    USING FILE 'tracking_file' REUSE;
```

The following statement disables block change tracking and deletes the existing block change tracking file:

```
ALTER DATABASE
  DISABLE BLOCK CHANGE TRACKING;
```

Resizing a Data File: Example

The following statement attempts to change the size of data file `diskb:tbs_f5.dbf`:

```
ALTER DATABASE
    DATAFILE 'diskb:tbs_f5.dbf' RESIZE 10 M;
```

Clearing a Log File: Example

The following statement clears a log file:

```
ALTER DATABASE
    CLEAR LOGFILE 'diskc:log3.log';
```

Database Recovery: Examples

The following statement performs complete recovery of the entire database, letting Oracle Database generate the name of the next archived redo log file needed:

```
ALTER DATABASE
  RECOVER AUTOMATIC DATABASE;
```

The following statement explicitly names a redo log file for Oracle Database to apply:

```
ALTER DATABASE
    RECOVER LOGFILE 'diskc:log3.log';
```

The following statement performs time-based recovery of the database:

```
ALTER DATABASE
    RECOVER AUTOMATIC UNTIL TIME '2001-10-27:14:00:00';
```

Oracle Database recovers the database until 2:00 p.m. on October 27, 2001.

For an example of recovering a tablespace, see "Using Parallel Recovery Processes: Example".

ALTER DATABASE DICTIONARY

Purpose

To encrypt obfuscated database link passwords and use the TDE framework to manage the encryption key.

A LOB locator (pointer to the location of a large object (LOB) value) can be assigned a signature to secure the LOB.

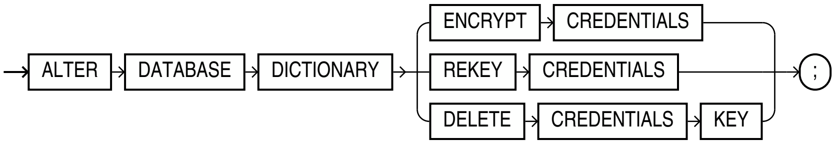
Prerequisites

- The TDE keystore must exist. The DDL first checks that the TDE:
  - Keystore exists.
  - Keystore is open.
  - Master Encryption Key exists in the TDE keystore.

    If any of the checks fail, the DDL fails. When this happens you must create a TDE keystore and provision a TDE Master Key. For more see the *Database Security Guide*.
- The instance initialization parameter `COMPATIBLE` must be set to 12.2.0.2.
- You must have `SYSKM` privileges to execute the command.

Syntax

alter_database_dictionary::=



Semantics

alter_database_dictionary_encrypt_credentials::=

This DDL encrypts existing and future obfuscated sensitive information in data dictionaries, for example database link passwords stored in `SYS.LINKS$`.

It performs the following actions:

- Inserts a new entry in `ENC$` corresponding to `SYS.LINK$`.
- It creates and initializes the SGA variable.
- De-obfuscates obfuscated passwords in `SYS.LINK$`.
- Encrypts the de-obfuscated passwords using the generated encryption key in `ENC$` for `SYS.LINK$`.
- Sets the flag to indicate a valid/usable dblink entry in `SYS.LINK$`.

When you use this DDL with LOB locator signature keys, they are always encrypted. A LOB locator (pointer to the location of a large object (LOB) value) can be assigned a signature to secure the LOB.

alter_database_dictionary_rekey_credentials::=

This DDL is used to change the data encryption key. It is applied to `SYS.LINK$` and any other tables covered under the data dictionary encryption framework.

You can also use this DDL to regenerate the LOB locator signature key for LOB locators. If the database is in restricted mode, then Oracle Database regenerates a new LOB signature key and encrypts it with the new encryption key. If the database is in non-restricted mode, then a new signature key is not regenerated but instead, Oracle Database uses a new encryption key to encrypt the existing LOB signature key.

alter_database_dictionary_delete_credentials_key::=

This DDL marks encrypted passwords unusuable. That means that current password entries in `SYS.LINK$` are marked unusable. It deletes the key in `ENC$` that was used to encrypt the credentials, and clears the SGA variable to prevent future encryption.

You can also use this DDL to delete the encrypted LOB locator signature key and then regenerate a new LOB signature key in obfuscated form.

> **See Also:**
>
> *Managing Security for Application Developers* in the *Database Security Guide*

ALTER DATABASE LINK

Purpose

Use the `ALTER DATABASE LINK` statement to modify a fixed-user database link when the password of the connection or authentication user changes.

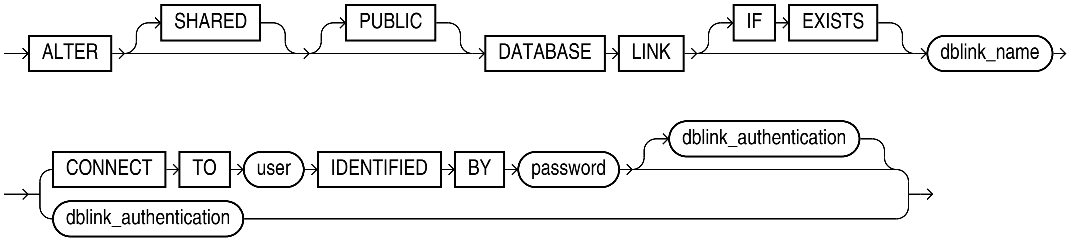
> **Note:**
>
> - You cannot use this statement to change the connection or authentication user associated with the database link. To change *`user`*, you must re-create the database link.
> - You cannot use this statement to change the password of a connection or authentication user. You must use the ALTER USER statement for this purpose, and then alter the database link with the `ALTER DATABASE LINK` statement.
> - This statement is valid only for fixed-user database links, not for connected-user or current user database links. See CREATE DATABASE LINK for more information on these two types of database links.

Prerequisites

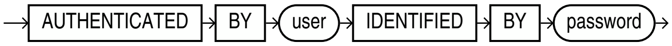
To alter a private database link, you must have the `ALTER DATABASE LINK` system privilege. To alter a public database link, you must have the `ALTER PUBLIC DATABASE LINK` system privilege.

Syntax

***alter_database_link*::=**



dblink_authentication



Semantics

The `ALTER DATABASE LINK` statement is intended only to update fixed-user database links with the current passwords of connection and authentication users. Therefore, any clauses valid in a `CREATE DATABASE LINK` statement that do not appear in the syntax diagram above are not valid in an `ALTER DATABASE LINK` statement. The semantics of all of the clauses permitted in this statement are the same as the semantics for those clauses in `CREATE DATABASE LINK`. Refer to CREATE DATABASE LINK for this information.

IF EXISTS

Specify `IF EXISTS` to alter an existing database link.

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

IDENTIFIED BY

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

Examples

The following statements show the valid variations of the `ALTER DATABASE LINK` statement:

```
ALTER DATABASE LINK private_link
  CONNECT TO hr IDENTIFIED BY hr_new_password;

ALTER PUBLIC DATABASE LINK public_link
  CONNECT TO scott IDENTIFIED BY scott_new_password;

ALTER SHARED PUBLIC DATABASE LINK shared_pub_link
  CONNECT TO scott IDENTIFIED BY scott_new_password
  AUTHENTICATED BY hr IDENTIFIED BY hr_new_password;

ALTER SHARED DATABASE LINK shared_pub_link
  CONNECT TO scott IDENTIFIED BY scott_new_password;
```

ALTER DIMENSION

Purpose

Use the `ALTER DIMENSION` statement to change the hierarchical relationships or dimension attributes of a dimension.

> **See Also:**
>
> CREATE DIMENSION and DROP DIMENSION

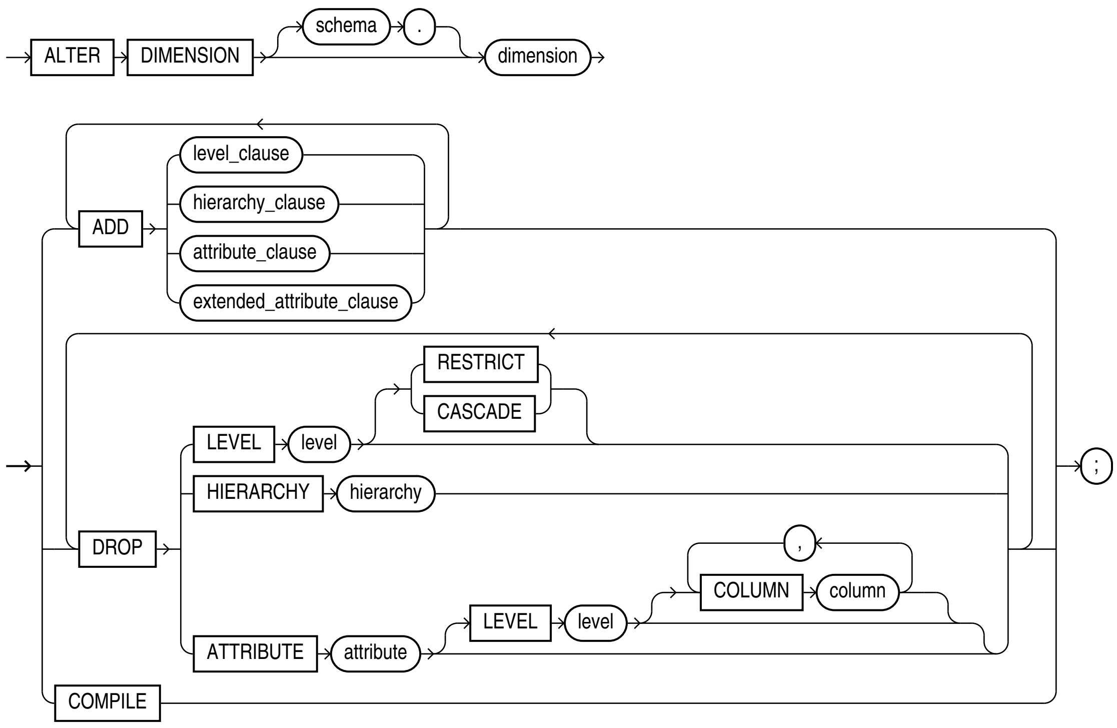
Prerequisites

The dimension must be in your schema or you must have the `ALTER ANY DIMENSION` system privilege to use this statement.

A dimension is always altered under the rights of the owner.

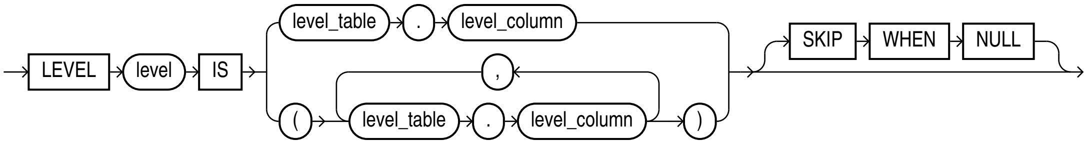
Syntax

***alter_dimension*::=**

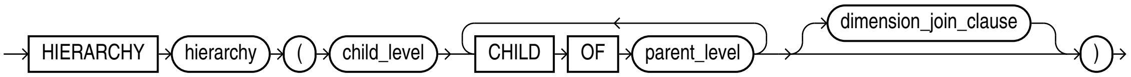


(*level_clause*::=, *hierarchy_clause*::=, *attribute_clause*::=, *extended_attribute_clause*::=)

***level_clause*::=**

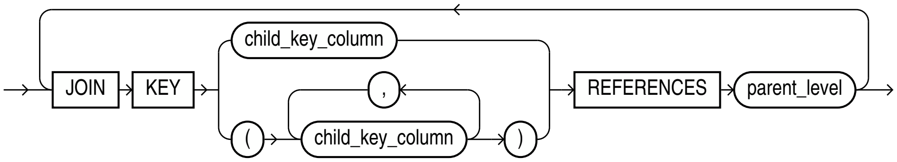


***hierarchy_clause*::=**

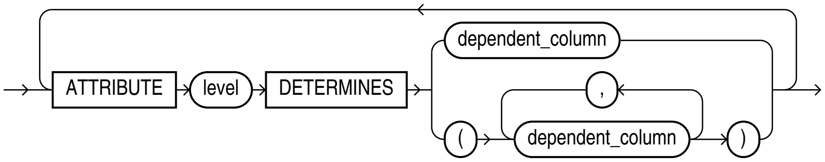


(*dimension_join_clause::=*)

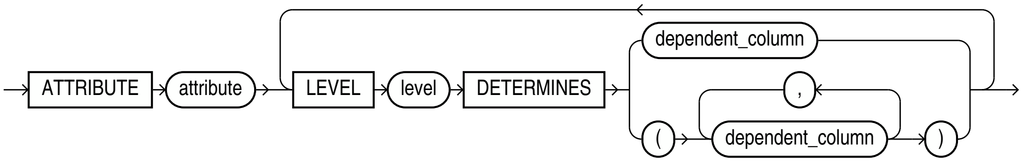
***dimension_join_clause*::=**



***attribute_clause*::=**



***extended_attribute_clause*::=**



Semantics

The following keywords, parameters, and clauses have meaning unique to `ALTER DIMENSION`. Keywords, parameters, and clauses that do not appear here have the same functionality that they have in the `CREATE DIMENSION` statement. Refer to CREATE DIMENSION for more information.

schema

Specify the schema of the dimension you want to modify. If you do not specify *schema*, then Oracle Database assumes the dimension is in your own schema.

dimension

Specify the name of the dimension. This dimension must already exist.

ADD

The ADD clauses let you add a level, hierarchy, or attribute to the dimension. Adding one of these elements does not invalidate any existing materialized view.

Oracle Database processes ADD LEVEL clauses prior to any other ADD clauses.

DROP

The DROP clauses let you drop a level, hierarchy, or attribute from the dimension. Any level, hierarchy, or attribute you specify must already exist.

Within one attribute, you can drop one or more level-to-column relationships associated with one level.

Restriction on DROP

If any attributes or hierarchies reference a level, then you cannot drop the level until you either drop all the referencing attributes and hierarchies or specify CASCADE.

CASCADE

Specify CASCADE if you want Oracle Database to drop any attributes or hierarchies that reference the level, along with the level itself.

RESTRICT

Specify RESTRICT if you want to prevent Oracle Database from dropping a level that is referenced by any attributes or hierarchies. This is the default.

COMPILE

Specify COMPILE to explicitly recompile an invalidated dimension. Oracle Database automatically compiles a dimension when you issue an ADD clause or DROP clause. However, if you alter an object referenced by the dimension (for example, if you drop and then re-create a table referenced in the dimension), Oracle Database invalidates, and you must recompile it explicitly.

Examples

Modifying a Dimension: Examples

The following examples modify the customers_dim dimension in the sample schema sh:

```
ALTER DIMENSION customers_dim
   DROP ATTRIBUTE country;

ALTER DIMENSION customers_dim
   ADD LEVEL zone IS customers.cust_postal_code
   ADD ATTRIBUTE zone DETERMINES (cust_city);
```

ALTER DISKGROUP

> **Note:**
>
> This SQL statement is valid only if you are using Oracle ASM and you have started an Oracle ASM instance. You must issue this statement from within the Oracle ASM instance, not from a normal database instance. For information on starting an Oracle ASM instance, refer to *Oracle Automatic Storage Management Administrator's Guide*.

Purpose

The `ALTER DISKGROUP` statement lets you perform a number of operations on a disk group or on the disks in a disk group.

> **See Also:**
>
> - CREATE DISKGROUP for information on creating disk groups
> - *Oracle Automatic Storage Management Administrator's Guide* for information on Oracle ASM and using disk groups to simplify database administration

Prerequisites

You must have an Oracle ASM instance started from which you issue this statement. The disk group to be modified must be mounted.

You can issue all `ALTER DISKGROUP` clauses if you have the `SYSASM` system privilege. You can issue specific clauses as follows:

- The `SYSOPER` privilege permits the following subset of the `ALTER DISKGROUP` operations: *`diskgroup_availability`*, *`rebalance_diskgroup_clause`*, *`check_diskgroup_clause`* (without the `REPAIR` option).
- If you are connected as `SYSDBA`, you have limited privileges to use this statement. The following operations are always granted to users connected as `SYSDBA`:
  - `ALTER DISKGROUP` ... `ADD DIRECTORY`
  - `ALTER DISKGROUP` ... `ADD`/`ALTER`/`DROP TEMPLATE` (nonsystem templates only)
  - `ALTER DISKGROUP` ... `ADD USERGROUP`
  - `SELECT`
  - `SHOW PARAMETER`

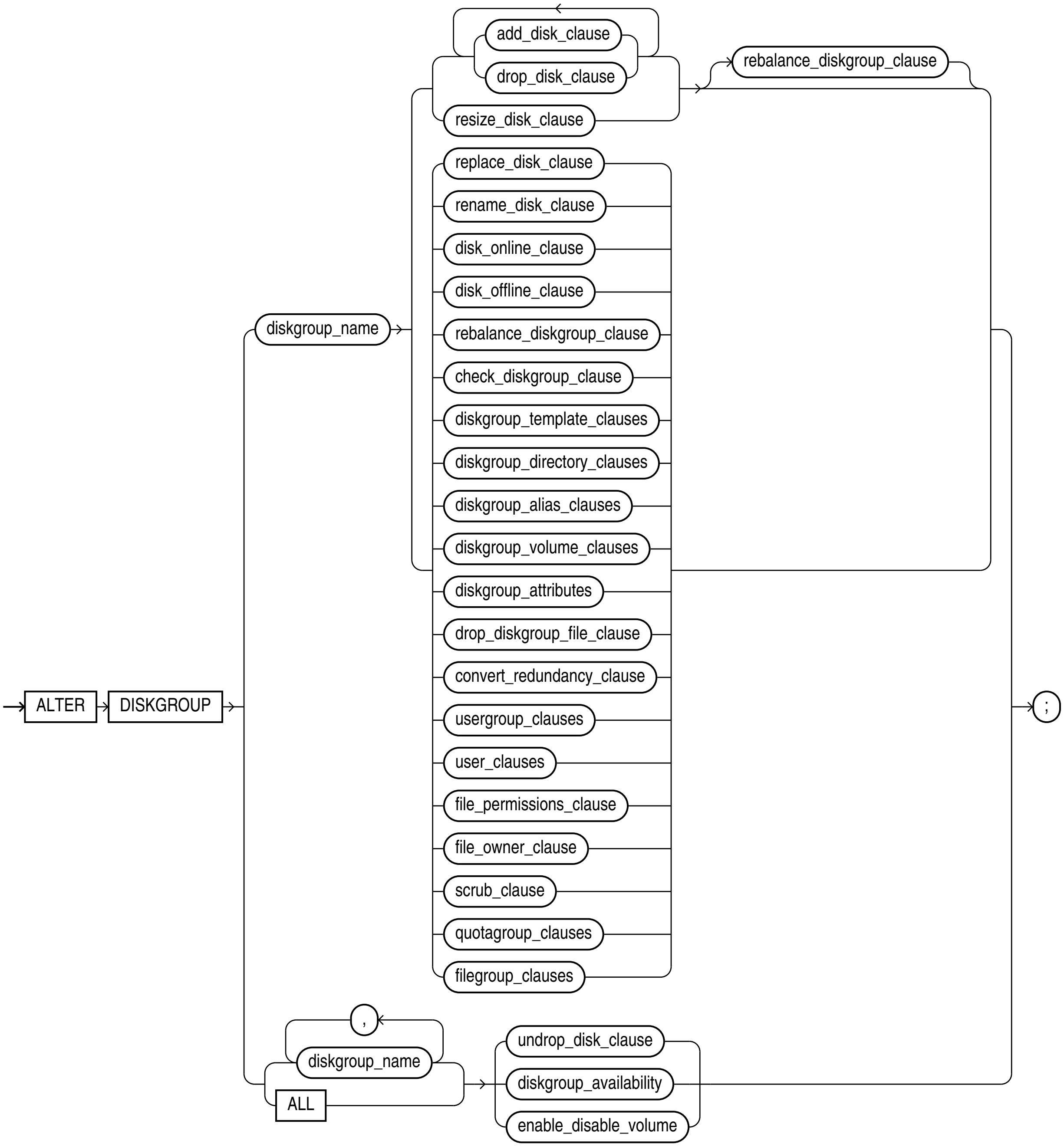
Table 10-1 shows additional privileges granted to users connected as `SYSDBA` under the conditions shown:

Table 10-1 Conditional Diskgroup Privileges for SYSDBA

ALTER DISKGROUP Operation	Condition
DROP FILE	User must have read-write permission on the file.
ADD ALIAS	User must have read-write permission on the related file.
RENAME ALIAS	User must have read-write permission on the related file.
DROP ALIAS	User must have read-write permission on the related file.
RENAME DIRECTORY	Directory must contain only aliases and no files. User must have DROP ALIAS permissions on all aliases under the directory.
DROP DIRECTORY	Directory must contain only aliases and no files. User must have DROP ALIAS permissions on all aliases under the directory.
DROP USERGROUP	User must be the owner of the user group.
MODIFY USERGROUP ADD MEMBER	User must be the owner of the user group.
MODIFY USERGROUP DROP MEMBER	User must be the owner of the user group.
SET PERMISSION	User must be the owner of the file.
SET OWNER GROUP	User must be the owner of the file and a member of the user group.

Syntax

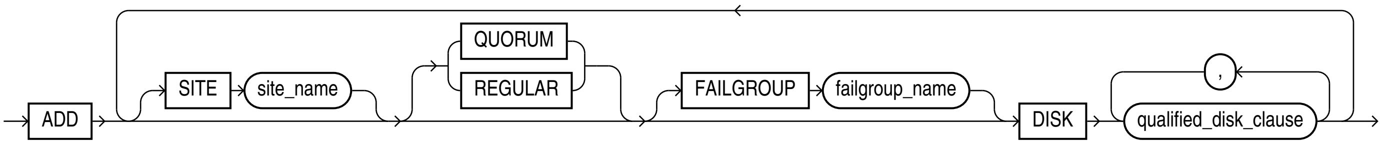
***alter_diskgroup*::=**



(*add_disk_clause*::=, *drop_disk_clause*::=, *resize_disk_clause*::=, *replace_disk_clause*::=, *rename_disk_clause*::=, *disk_online_clause*::=, *disk_offline_clause*::=, *rebalance_diskgroup_clause*::=, *check_diskgroup_clause*::=, *diskgroup_template_clauses*::=, *diskgroup_directory_clauses*::=, *diskgroup_alias_clauses*::=, *diskgroup_volume_clauses*::=, *diskgroup_attributes*::=, *modify_diskgroup_file*::=, *drop_diskgroup_file_clause*::=, *convert_redundancy_clause*::=, *usergroup_clauses*::=, *user_clauses*::=, *file_permissions_clause*::=, *file_owner_clause*::=, *scrub_clause*::=, *quotagroup_clauses*::=,

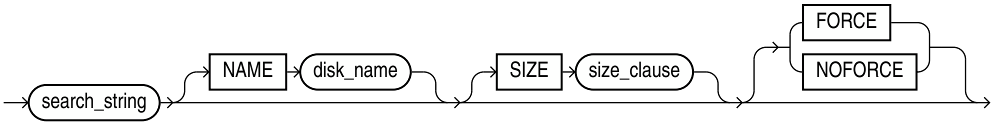
(*filegroup_clauses*::=, *undrop_disk_clause*::=, *diskgroup_availability*::=, *enable_disable_volume*::=)

***add_disk_clause*::=**



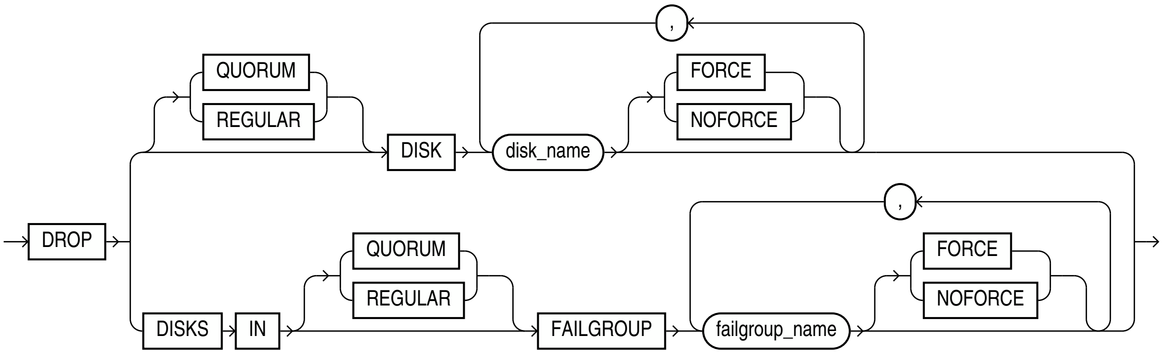
(*qualified_disk_clause*::=)

***qualified_disk_clause*::=**

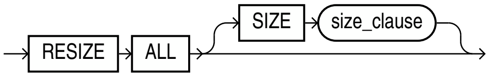


(*size_clause*::=)

***drop_disk_clause*::=**

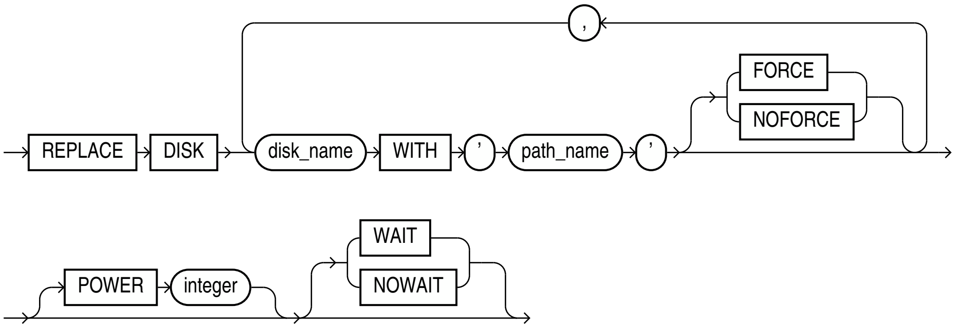


***resize_disk_clause*::=**

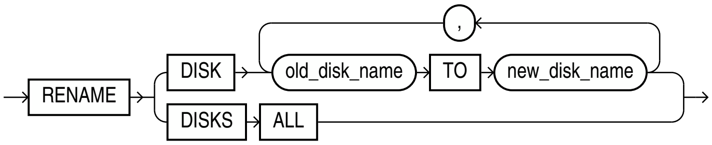


(*size_clause*::=)

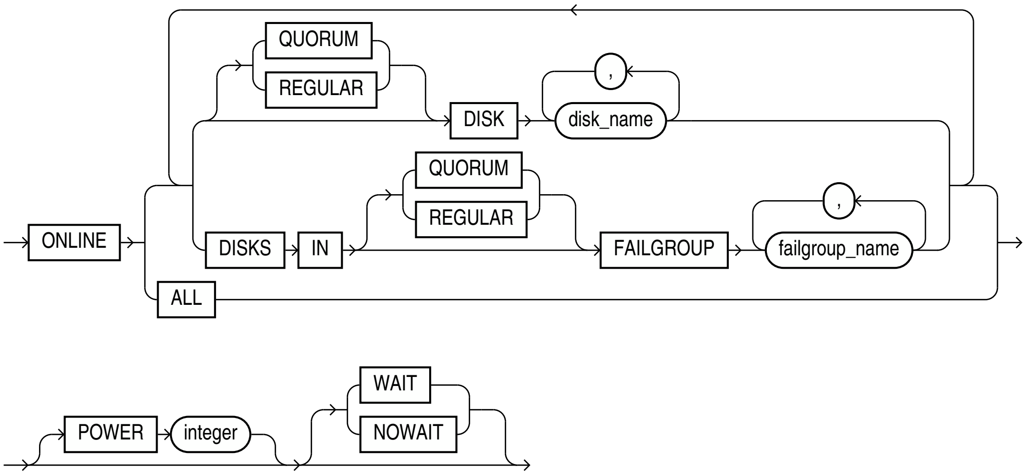
***replace_disk_clause*::=**



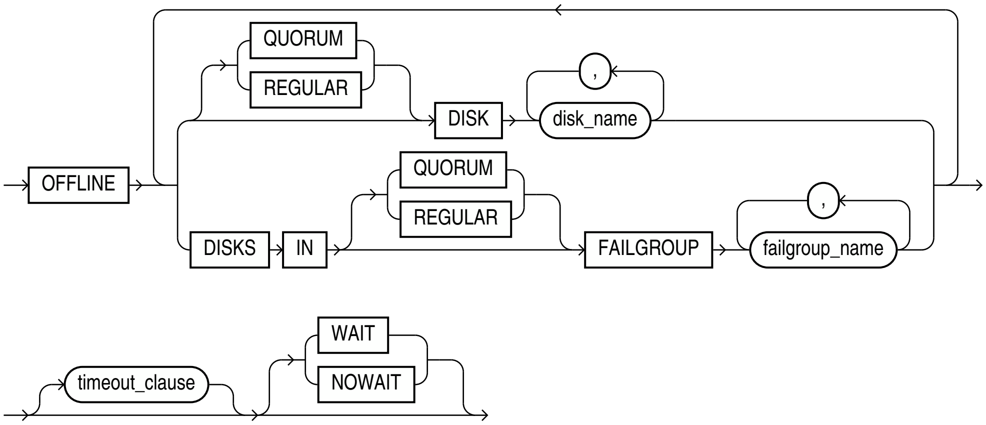
***rename_disk_clause*::=**



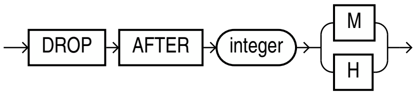
***disk_online_clause*::=**



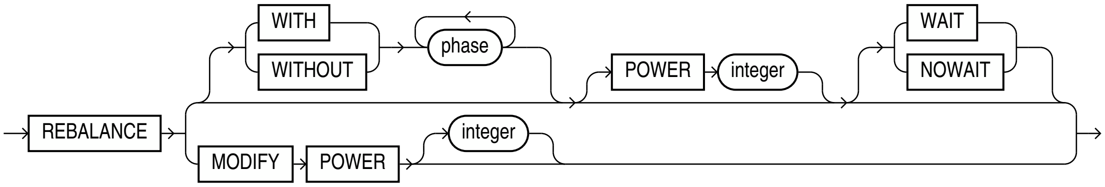
disk_offline_clause::=



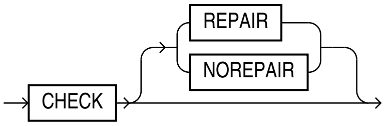
timeout_clause::=



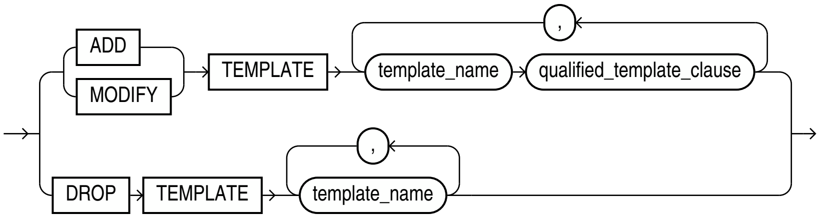
rebalance_diskgroup_clause::=



check_diskgroup_clause::=

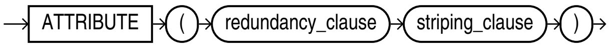


diskgroup_template_clauses::=

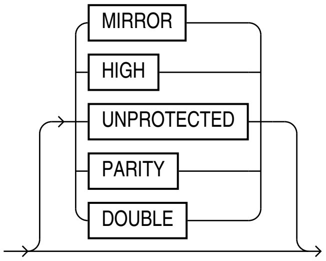


(*qualified_template_clause::=*)

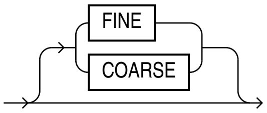
qualified_template_clause::=



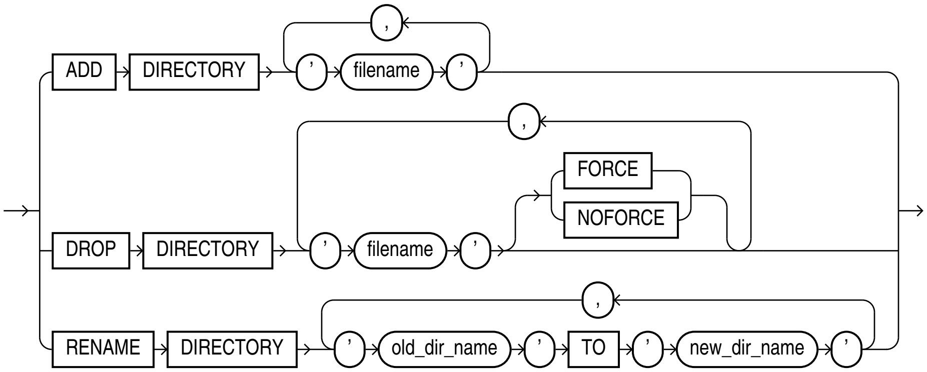
redundancy_clause::=



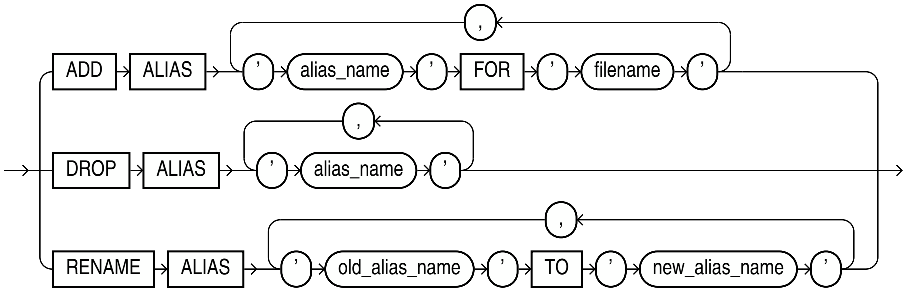
striping_clause::=



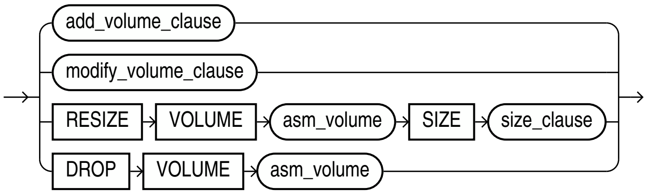
diskgroup_directory_clauses::=



diskgroup_alias_clauses::=

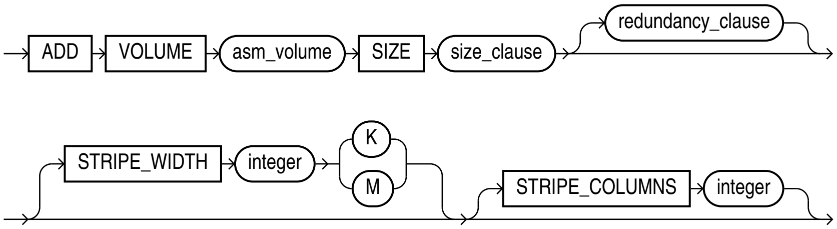


***diskgroup_volume_clauses*::=**



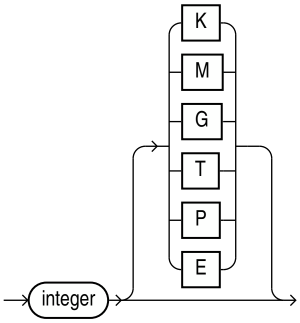
(*add_volume_clause*::=, *modify_volume_clause*::=

***add_volume_clause*::=**

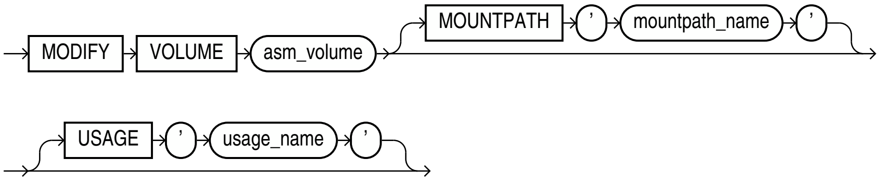


(*size_clause*::=, *redundancy_clause*::=,)

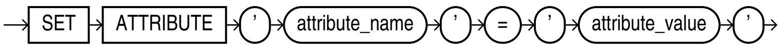
***size_clause*::=**



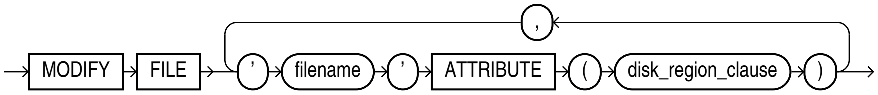
***modify_volume_clause*::=**



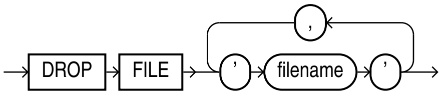
***diskgroup_attributes*::=**



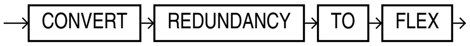
***modify_diskgroup_file*::=**



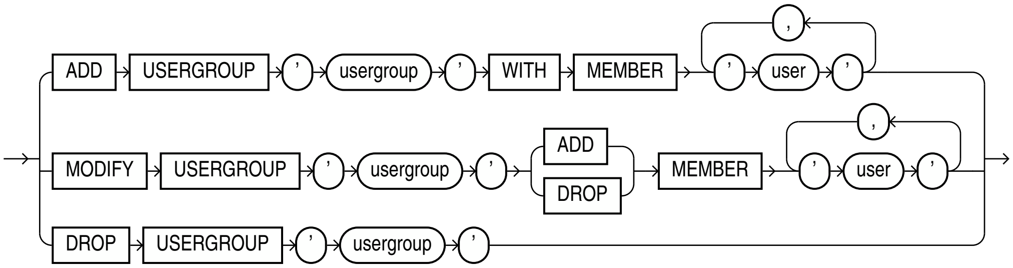
***drop_diskgroup_file_clause*::=**



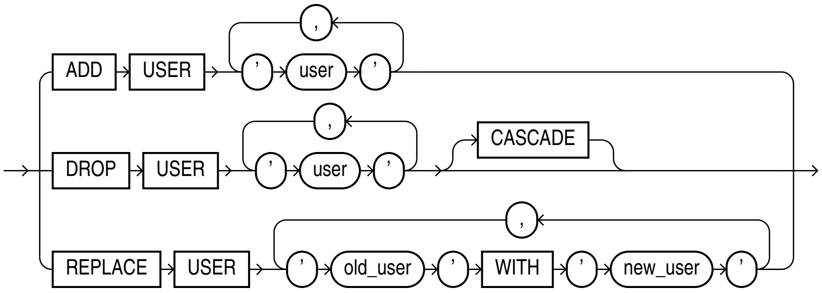
***convert_redundancy_clause*::=**



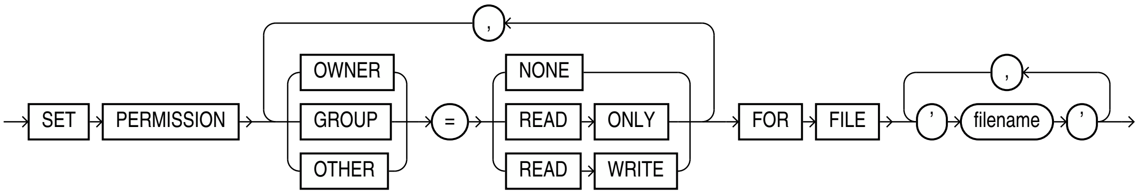
***usergroup_clauses*::=**



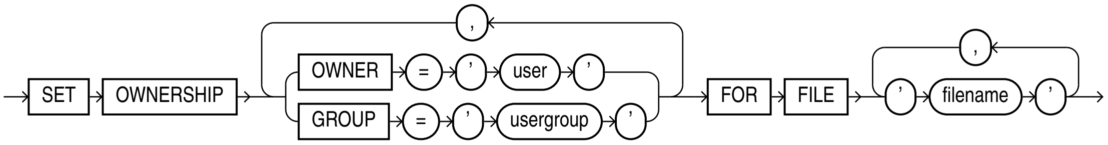
***user_clauses*::=**



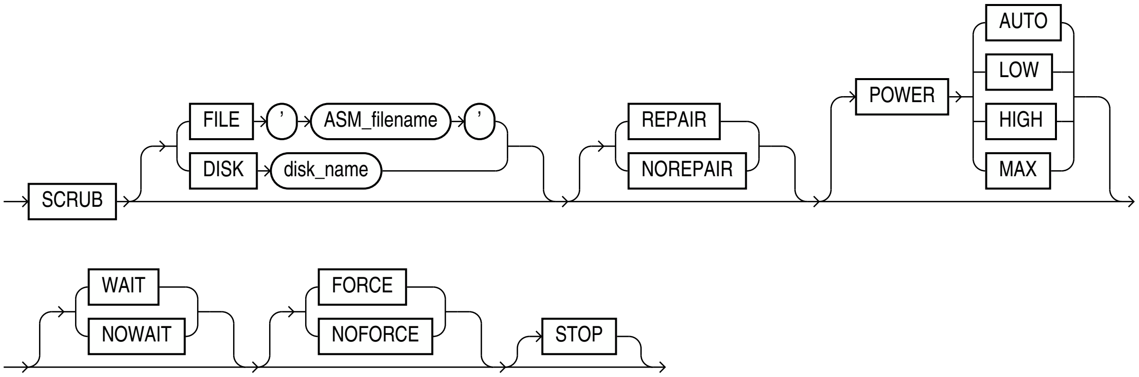
file_permissions_clause::=



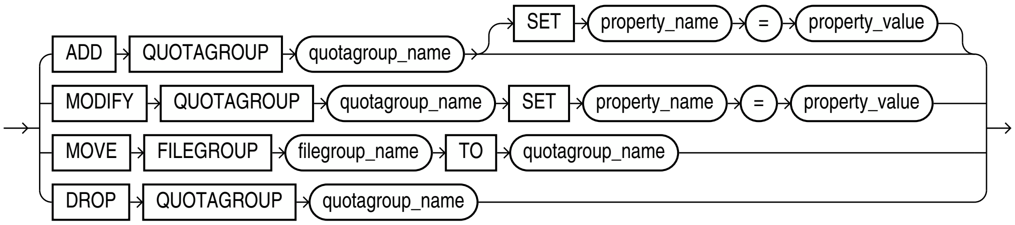
file_owner_clause::=



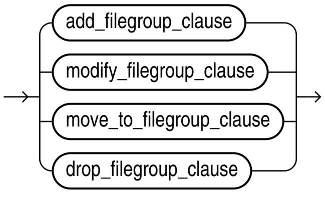
scrub_clause::=



quotagroup_clauses::=

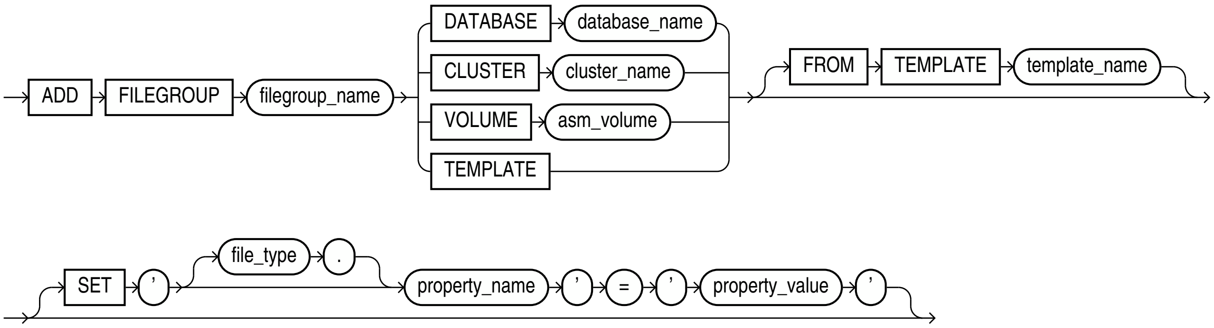


filegroup_clauses::=

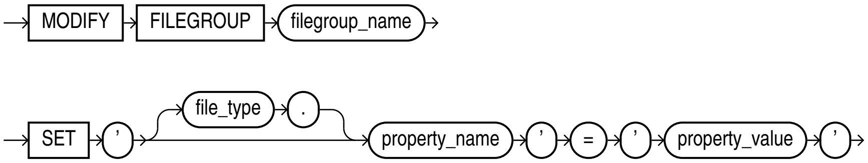


(*add_filegroup_clause*::=, *modify_filegroup_clause*::=, *move_to_filegroup_clause*::=, *drop_filegroup_clause*::=)

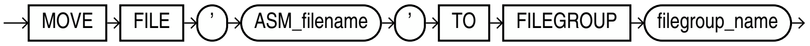
add_filegroup_clause::=



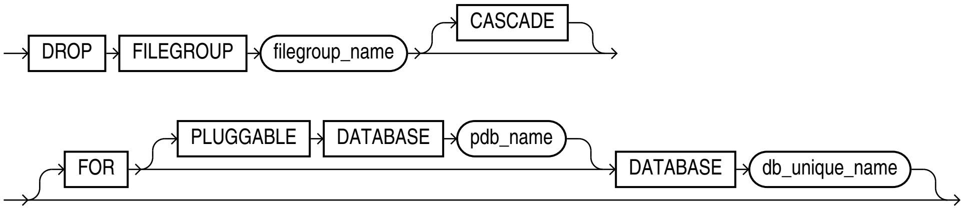
modify_filegroup_clause::=



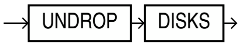
move_to_filegroup_clause::=



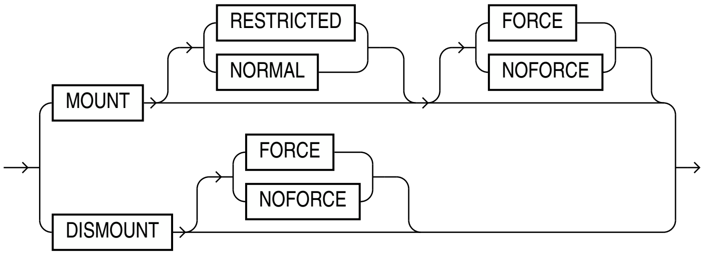
drop_filegroup_clause::=



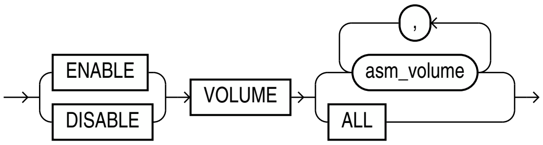
undrop_disk_clause::=



***diskgroup_availability*::=**



***enable_disable_volume*::=**



Semantics

diskgroup_name

Specify the name of the disk group you want to modify. To determine the names of existing disk groups, query the `V$ASM_DISKGROUP` dynamic performance view.

add_disk_clause

Use this clause to add one or more disks to the disk group and specify attributes for the newly added disk. Oracle ASM automatically rebalances the disk group as part of this operation.

You cannot use this clause to change the failure group of a disk. Instead you must drop the disk from the disk group and then add the disk back into the disk group as part of the new failure group.

To determine the names of the disks already in this disk group, query the `V$ASM_DISK` dynamic performance view.

QUORUM | REGULAR

The semantics of these keyword are the same as the semantics in a `CREATE DISKGROUP` statement. See QUORUM | REGULAR for more information on these keywords.

You cannot change this qualifier for an existing disk or disk group. Therefore, you cannot specify in this clause a keyword different from the keyword that was specified when the disk group was created.

> **See Also:**
>
> *Oracle Automatic Storage Management Administrator's Guide* for more information about the use of these keywords

FAILGROUP Clause

Use this clause to assign the newly added disk to a failure group. If you omit this clause and you are adding the disk to a normal or high redundancy disk group, then Oracle Database automatically adds the newly added disk to its own failure group. The implicit name of the failure group is the same as the operating system independent disk name (see "NAME Clause").

You cannot specify this clause if you are creating an external redundancy disk group.

qualified_disk_clause

This clause has the same semantics in CREATE DISKGROUP and ALTER DISKGROUP statements. For complete information on this clause, refer to *qualified_disk_clause* in the documentation on CREATE DISKGROUP.

drop_disk_clause

Use this clause to drop one or more disks from the disk group.

DROP DISK

The DROP DISK clause lets you drop one or more disks from the disk group and automatically rebalance the disk group. When you drop a disk, Oracle ASM relocates all the data from the disk and clears the disk header so that it no longer is part of the disk group. The disk header is not cleared if you specify the FORCE keyword.

Specify *disk_name* as shown in the NAME column of the V$ASM_DISK dynamic performance view.

If a disk to be dropped is a quorum disk or belongs to a quorum failure group, then you must specify QUORUM in order to drop the disk. See QUORUM | REGULAR.

DROP DISKS IN FAILGROUP

The DROP DISKS IN FAILGROUP clause lets you drop all the disks in the specified failure group. The behavior is otherwise the same as that for the DROP DISK clause.

If the specified failure group is a quorum failure group, then you must specify the QUORUM keyword in order to drop the disks. See QUORUM | REGULAR.

FORCE | NOFORCE

These keywords let you specify when the disk is considered to be no longer part of the disk group. The default and recommended setting is NOFORCE.

- When you specify NOFORCE, Oracle ASM reallocates all of the extents of the disk to other disks and then expels the disk from the disk group and rebalances the disk group.

> **Note:**
>
> DROP DISK ... NOFORCE returns control to the user before the disk can be safely reused or removed from the system. To ensure that the drop disk operation has completed, query the V$ASM_DISK view to verify that HEADER_STATUS has the value FORMER. Do not attempt to remove or reuse a disk if STATE has the value DROPPING. Query the V$ASM_OPERATION view for approximate information on how long it will take to complete the rebalance resulting from dropping the disk.If you also specify REBALANCE ... WAIT (see *rebalance_diskgroup_clause*), then the statement will not return until the rebalance operation is complete and the disk has been cleared. However, you should always verify that the HEADER_STATUS column of V$ASM_DISK is FORMER, because of the unlikely event the rebalance operations fails.

- When you specify FORCE, Oracle Database expels the disk from the disk group immediately. It then reconstructs the data from the redundant copies on other disks, reallocates the data to other disks, and rebalances the disk group.

  The FORCE clause can be useful, for example, if Oracle ASM can no longer read the disk to be dropped. However, it is more time consuming than a NOFORCE drop, and it can leave portions of a file with reduced protection. You cannot specify FORCE for an external redundancy disk group at all, because in the absence of redundant data on the disk, Oracle ASM must read the data from the disk before it can be dropped.

The rebalance operation invoked when a disk is dropped is time consuming, whether or not you specify FORCE or NOFORCE. You can monitor the progress by querying the V$ASM_OPERATION dynamic performance view. Refer to *rebalance_diskgroup_clause* for more information on rebalance operations.

resize_disk_clause

Use this clause to specify a new size for every disk in a disk group. This clause lets you override the size returned by the operating system or the size you specified previously for the disks.

SIZE

Specify the new size in kilobytes, megabytes, gigabytes, or terabytes. You cannot specify a size greater than the capacity of the disk. If you specify a size smaller than the disk capacity, then you limit the amount of disk space Oracle ASM will use. If you omit this clause, then Oracle ASM attempts programatically to determine the size of the disks.

replace_disk_clause

Use this clause to replace one or more disks in the disk group. This clause allows you to replace disks with a single operation, which is more efficient than dropping and adding each disk.

For *disk_name*, specify the name of the disk you want to replace. This name is assigned to the replacement disk. You can view disk names by querying the NAME column of the V$ASM_DISK dynamic performance view.

For *path_name*, specify the full path name for the replacement disk.

FORCE

Specify `FORCE` if you want Oracle ASM to add the replacement disk to the disk group even if the replacement disk is already a member of a disk group.

> **Note:**
>
> Using `FORCE` in this way may destroy existing disk groups.

NOFORCE

Specify `NOFORCE` if you want Oracle ASM to return an error if the replacement disk is already a member of a disk group. `NOFORCE` is the default.

POWER

The `POWER` clause has the same semantics here as for a manual rebalancing of a disk group, except that the power value cannot be set to 0. See POWER.

WAIT | NOWAIT

The `WAIT` and `NOWAIT` keywords have the same semantics here as for a manual rebalancing of a disk group. See WAIT | NOWAIT.

rename_disk_clause

Use this clause to rename one or more disks in the disk group. The disk group must be in the `MOUNT RESTRICTED` state and all disks in the disk group must be online.

RENAME DISK

Specify this clause to rename one or more disks. For each disk, specify the *old_disk_name* and *new_disk_name*. If *new_disk_name* already exists, then this operation fails.

RENAME DISKS ALL

Specify this clause to rename all disks in the disk group to a name of the form *diskgroupname_####*, where *####* is the disk number. Disk names that are already in the *diskgroupname_####* format are not changed.

disk_online_clause

Use this clause to bring one or more disks online and rebalance the disk group.

ONLINE DISK

The `ONLINE DISK` clause lets you bring one or more specified disks online and rebalance the disk group.

Specify *disk_name* as shown in the `NAME` column of the `V$ASM_DISK` dynamic performance view.

The `QUORUM` and `REGULAR` keywords have the same semantics here as they have when adding a disk to a disk group. See QUORUM | REGULAR.

ONLINE DISKS IN FAILGROUP

The `ONLINE DISKS IN FAILGROUP` clause lets you bring all disks in the specified failure group online and rebalance the disk group.

If the specified failure group is a quorum failure group, then you must specify the QUORUM keyword in order to bring the disks online. See QUORUM | REGULAR.

ALL

The ALL clause lets you bring all disks in the disk group online and rebalance the disk group.

POWER

The POWER clause has the same semantics here as for a manual rebalancing of a disk group. See POWER

WAIT | NOWAIT

The WAIT and NOWAIT keywords have the same semantics here as for a manual rebalancing of a disk group. See WAIT | NOWAIT.

disk_offline_clause

Use the *disk_offline_clause* to take one or more disks offline. This clause fails if the redundancy level of the disk group would be violated by taking the specified disks offline.

OFFLINE DISK

The OFFLINE DISK clause lets you take one or more specified disks offline.

Specify *disk_name* as shown in the NAME column of the V$ASM_DISK dynamic performance view.

The QUORUM and REGULAR keywords have the same semantics here as they have when adding a disk to a disk group. See QUORUM | REGULAR.

OFFLINE DISKS IN FAILGROUP

The OFFLINE DISKS IN FAILGROUP clause lets you take all disks in the specified failure group offline.

If the specified failure group is a quorum failure group, then you must specify the QUORUM keyword in order to take the disks offline. See QUORUM | REGULAR.

timeout_clause

By default, Oracle ASM drops a disk shortly after it is taken offline. You can delay this operation by specifying the *timeout_clause*, which gives you the opportunity to repair the disk and bring it back online. You can specify the timeout value in units of minute or hour. If you omit the unit, then the default is hour.

You can change the timeout period by specifying this clause multiple times. Each time you specify it, Oracle ASM measures the time from the most recent previous *disk_offline_clause* while the disk group is mounted. To learn how much time remains before Oracle ASM will drop an offline disk, query the REPAIR_TIMER column of V$ASM_DISK.

This clause overrides any previous setting of the disk_repair_time attribute. Refer to Table 13-2 for more information about disk group attributes.

> **See Also:**
>
> *Oracle Automatic Storage Management Administrator's Guide* for more information about taking Oracle ASM disks online and offline

WAIT | NOWAIT

Specify WAIT for all disk handles to close all instances before offline returns. The default wait timeout is 300 seconds (5 minutes).

rebalance_diskgroup_clause

Use this clause to manually rebalance the disk group. During a rebalance operation, Oracle ASM redistributes data files evenly across all drives. This clause is rarely necessary, because Oracle ASM allocates files evenly and automatically rebalances disk groups when the storage configuration changes. However, it is useful if you want to perform a controlled rebalance operation. It allows you to include or exclude certain phases of a rebalance operation, pause and restart a rebalance operation, and adjust the power of a rebalance operation.

WITH | WITHOUT

A rebalance operation consists of the following phases: RESTORE (includes the RESYNC, RESILVER, or REBUILD phases), BALANCE, PREPARE, and COMPACT.

You can use the WITH or WITHOUT clause to instruct Oracle ASM to include or exclude specific phases of a rebalance operation. For example, if you have time constraints, you can include only the RESTORE phase. Or, if you are using flash storage disk groups or disk groups with flash cache, you can exclude the COMPACT phase, which is not beneficial for such disk groups.

- Use the WITH clause to include only the specified phases of a rebalance operation. You can specify any of phases RESTORE, BALANCE, PREPARE, and COMPACT. It is acceptable, but not necessary, to specify RESTORE, because the RESTORE phase always occurs.
- Use the WITHOUT clause to exclude the specified phases of a rebalance operation. You can specify any of the phases BALANCE, PREPARE, and COMPACT. You cannot specify RESTORE, because the RESTORE phase must always occur.

The order in which you specify multiple phases in the WITH or WITHOUT clause does not matter. Oracle ASM will perform the phases of the rebalance operation in the proper order. You cannot specify the RESYNC, RESILVER, or REBUILD phases; they are part of the RESTORE phase.

If you omit the WITH and WITHOUT clauses, then Oracle ASM performs all phases of the rebalance operation.

You can monitor the progress of the rebalance operation by querying the V$ASM_OPERATION dynamic performance view.

See *Oracle Automatic Storage Management Administrator's Guide* for more information on the phases of a rebalance operation.

POWER

This clause lets you specify the power, or speed, of the rebalance operation. It also lets you stop the rebalance operation.

For *integer*, specify a value from 0 to 1024:

- A value of 1 through 1024 specifies the power at which Oracle ASM is to perform the rebalance operation, with 1 representing the lowest possible power and 1024 representing the highest possible power.
- A value of 0 stops an active rebalance operation. No further rebalancing will occur until the start of another manual or automatic rebalance operation on the disk group, and at that time the rebalance operation will start from the beginning. If you would like to have the option of later resuming the rebalance operation from where it left off, then instead stop the

rebalance operation by specifying MODIFY POWER 0. See the clause MODIFY POWER for more information.

If you omit the POWER clause, then the default power is determined as follows:

- For flex disk groups, Oracle ASM rebalances each file group according the value of its POWER_LIMIT property. If the POWER_LIMIT property is not set for a file group, then Oracle ASM uses the value of the ASM_POWER_LIMIT initialization parameter for the file group.
- For all other types of disk groups, if you omit the POWER clause, then Oracle ASM rebalances the disk group according to the value of the ASM_POWER_LIMIT initialization parameter.

WAIT | NOWAIT

Use this clause to specify when, in the course of the rebalance operation, control should be returned to the user.

- Specify WAIT if you want control returned to the user after the rebalance operation has finished. You can explicitly terminate a rebalance operation running in WAIT mode, although doing so does not undo any completed disk add, drop, or resize operations in the same statement.
- Specify NOWAIT if you want control returned to the user immediately after the statement is issued. This is the default.

MODIFY POWER

Use this clause to pause, resume, or change the power of an active rebalance operation.

You can specify *integer* as follows:

- Specify 0 to pause the rebalance operation. When you pause a rebalance operation in this manner, you can subsequently resume the operation from the phase where it left off by issuing an ALTER DISKGROUP ... MODIFY POWER ... statement. If you subsequently start a manual rebalance operation on the disk group using the clause POWER, or an automatic rebalance operation for the disk group occurs, then the rebalance operation will start at the beginning.
- Specify 1 through 1024 to specify the power of the rebalance operation, with 1 representing the lowest possible power and 1024 representing the highest possible power. If a rebalance operation is running, then Oracle ASM changes the power without interrupting the operation. If a rebalance operation was previously paused with the MODIFY POWER 0 clause, then the rebalance operation resumes at the specified power.
- Omit *integer* to specify the default power. If a rebalance operation is running, then Oracle ASM changes the power to the default power without interrupting the operation. If a rebalance operation was previously paused with the MODIFY POWER 0 clause, then the rebalance operation resumes at the default power. Refer to the clause POWER for information on how the default power is determined.

> **See Also:**
>
> - *Oracle Database Reference* for more information on the ASM_POWER_LIMIT initialization parameter and the V$ASM_OPERATION dynamic performance view
> - Rebalancing a Disk Group: Example

check_diskgroup_clause

The *check_diskgroup_clause* lets you verify the internal consistency of Oracle ASM disk group metadata. The disk group must be mounted. Oracle ASM displays summary errors and writes the details of the detected errors in the alert log.

The CHECK keyword performs the following operations:

- Checks the consistency of the disk.
- Cross checks all the file extent maps and allocation tables for consistency.
- Checks that the alias metadata directory and file directory are linked correctly.
- Checks that the alias directory tree is linked correctly.
- Checks that Oracle ASM metadata directories do not have unreachable allocated blocks.

REPAIR | NOREPAIR

This clause lets you instruct Oracle ASM whether or not to attempt to repair any errors found during the consistency check. The default is NOREPAIR. The NOREPAIR setting is useful if you want to be alerted to any inconsistencies but do not want Oracle ASM to take any automatic action to resolve them.

Deprecated Clauses

In earlier releases, you could specify CHECK for ALL, DISK, DISKS IN FAILGROUP, or FILE. Those clauses have been deprecated as they are no longer needed. If you specify them, then their behavior is the same as in earlier releases and a message is added to the alert log. However, Oracle recommends that you do not introduce these clauses into your new code, as they are scheduled for desupport. The deprecated clauses are these:

- ALL checks all disks and files in the disk group.
- DISK checks one or more specified disks in the disk group.
- DISKS IN FAILGROUP checks all disks in a specified failure group.
- FILE checks one or more specified files in the disk group. You must use one of the reference forms of the filename. Refer to *ASM_filename* for information on the reference forms of Oracle ASM filenames.

diskgroup_template_clauses

A template is a named collection of attributes. When you create a disk group, Oracle ASM associates a set of initial system default templates with that disk group. The attributes defined by the template are applied to all files in the disk group. Table 10-2 lists the system default templates and the attributes they apply to the various file types. The *diskgroup_template_clauses* described following the table let you change the template attributes and create new templates.

You cannot use this clause to change the attributes of a disk group file after it has been created. Instead, you must use Recovery Manager (RMAN) to copy the file into a new file with the new attributes.

Table 10-2   Oracle Automatic Storage Management System Default File Group Templates

Template Name	File Type	Mirroring Level in External Redundancy Disk Groups	Mirroring Level in Normal Redundancy Disk Groups	Mirroring Level in High Redundancy Disk Groups	Striped
`CONTROLFILE`	Control files	Unprotected	3-way mirror	3-way mirror	`FINE`
`DATAFILE`	Data Files and copies	Unprotected	2-way mirror	3-way mirror	`COARSE`
`ONLINELOG`	Online logs	Unprotected	2-way mirror	3-way mirror	`COARSE`
`ARCHIVELOG`	Archive logs	Unprotected	2-way mirror	3-way mirror	`COARSE`
`TEMPFILE`	Temp files	Unprotected	2-way mirror	3-way mirror	`COARSE`
`BACKUPSET`	Data File backup pieces, data file incremental backup pieces, and archive log backup pieces	Unprotected	2-way mirror	3-way mirror	`COARSE`
`PARAMETERFILE`	SPFILE	Unprotected	2-way mirror	3-way mirror	`COARSE`
`DATAGUARDCONFIG`	Disaster recovery configurations (used in standby databases)	Unprotected	2-way mirror	3-way mirror	`COARSE`
`FLASHBACK`	Flashback logs	Unprotected	2-way mirror	3-way mirror	`COARSE`
`CHANGETRACKING`	Block change tracking data (used during incremental backups)	Unprotected	2-way mirror	3-way mirror	`COARSE`
`DUMPSET`	Data Pump dumpset	Unprotected	2-way mirror	3-way mirror	`COARSE`
`XTRANSPORT`	Cross-platform converted data file	Unprotected	2-way mirror	3-way mirror	`COARSE`
`AUTOBACKUP`	Automatic backup files	Unprotected	2-way mirror	3-way mirror	`COARSE`
`ASMPARAMETERFILE`	SPFILE	Unprotected	2-way mirror	3-way mirror	`COARSE`
`OCRFILE`	Oracle Cluster Registry file	Unprotected	2-way mirror	3-way mirror	`COARSE`

ADD TEMPLATE

Use this clause to add one or more named templates to a disk group. To determine the names of existing templates, query the `V$ASM_TEMPLATE` dynamic performance view.

MODIFY TEMPLATE

Use this clause to modify the attributes of a system default or user-defined disk group template. Only the specified attributes are altered. Unspecified properties retain their current values.

> **Note:**
>
> In earlier releases, the keywords `ALTER TEMPLATE` were used instead of `MODIFY TEMPLATE`. The `ALTER` keyword is still supported for backward compatibility, but is replaced with `MODIFY` for consistency with other Oracle SQL.

template_name

Specify the name of the template to be added or modified. The maximum length of a template name is 30 characters. The name must satisfy the requirements listed in "Database Object Naming Rules ".

redundancy_clause

Specify `PARITY` for single parity protection for write-once file types like archive logs and backup sets. If parity protection is not specified, the default redundancy for write-once file types will continue to be derived from system templates.

Specify `DOUBLE` for double parity for write-once file types like archive logs and backup sets. If parity protection is not specified, the default redundancy for write-once file types will continue to be derived from system templates.

Example:

```
ALTER DISKGROUP <diskgroup_name> MODIFY TEMPLATE <archivelog> ATTRIBUTE (DOUBLE)
```

The redundancy of write-once file types may be changed to support parity protection later as needed.

Specify the redundancy level of the newly added or modified template:

- `MIRROR`: Files to which this template are applied are protected by mirroring their data blocks. In normal redundancy disk groups, each primary extent has one mirror extent (2-way mirroring). For high redundancy disk groups, each primary extent has two mirror extents (3-way mirroring). You cannot specify `MIRROR` for templates in external redundancy disk groups.
- `HIGH`: Files to which this template are applied are protected by mirroring their data blocks. Each primary extent has two mirror extents (3-way mirroring) for both normal redundancy and high redundancy disk groups. You cannot specify `HIGH` for templates in external redundancy disk groups.
- `UNPROTECTED`: Files to which this template are applied are not protected by Automated Storage Management from media failures. Disks taken offline, either through system action or by user command, can cause loss of unprotected files. `UNPROTECTED` is the only valid setting for external redundancy disk groups. `UNPROTECTED` may not be specified for templates in high redundancy disk groups. Oracle discourages the use of unprotected files in high and normal redundancy disk groups.
- `PARITY`: Specify the property `PARITY` for single parity for write-once file types only.

If you omit the redundancy clause, then the value defaults to `MIRROR` for a normal redundancy disk group, `HIGH` for a high redundancy disk group, and `UNPROTECTED` for an external redundancy disk group.

striping_clause

Specify how the files to which this template are applied will be striped:

- `FINE`: Files to which this template are applied are striped every 128KB. This striping mode is not valid for an Oracle ASM spfile.
- `COARSE`: Files to which this template are applied are striped every 1MB. This is the default value.

DROP TEMPLATE

Use this clause to drop one or more templates from the disk group. You can use this clause to drop only user-defined templates, not system default templates.

diskgroup_directory_clauses

Before you can create alias names for Oracle ASM filenames (see *diskgroup_alias_clauses*), you must specify the full directory structure in which the alias name will reside. The `diskgroup_directory_clauses` let you create and manipulate such a directory structure.

ADD DIRECTORY

Use this clause to create a new directory path for hierarchically named aliases. Use a slash (/) to separate components of the directory. Each directory component can be up to 48 bytes in length and must not contain the slash character. You cannot use a space for the first or last character of any component. The total length of the directory path cannot exceed 256 bytes minus the length of any alias name you intend to create in this directory (see *diskgroup_alias_clauses*).

DROP DIRECTORY

Use this clause to drop a directory for hierarchically named aliases. Oracle ASM will not drop the directory if it contains any alias definitions unless you also specify `FORCE`. This clause is not valid for dropping directories created as part of a system alias. Such directories are labeled with the value `Y` in the `SYSTEM_CREATED` column of the `V$ASM_ALIAS` dynamic performance view.

RENAME DIRECTORY

Use this clause to change the name of a directory for hierarchically named aliases. This clause is not valid for renaming directories created as part of a system alias. Such directories are labeled with the value `Y` in the `SYSTEM_CREATED` column of the `V$ASM_ALIAS` dynamic performance view.

diskgroup_alias_clauses

When an Oracle ASM file is created, either implicitly or by user specification, Oracle ASM assigns to the file a fully qualified name ending in a dotted pair of numbers (see *file_specification*). The `diskgroup_alias_clauses` let you create more user-friendly alias names for the Oracle ASM filenames. You cannot specify an alias name that ends in a dotted pair of numbers, as this format is indistinguishable from an Oracle ASM filename.

Before specifying this clause, you must first create the directory structure appropriate for your naming conventions (see *diskgroup_directory_clauses*). The total length of the alias name, including the directory prefix, is limited to 256 bytes. Alias names are case insensitive but case retentive.

ADD ALIAS

Use this clause to create an alias name for an Oracle ASM filename. The `alias_name` consists of the full directory path and the alias itself. To determine the names of existing Oracle ASM aliases, query the `V$ASM_ALIAS` dynamic performance view. Refer to *ASM_filename* for information on Oracle ASM filenames.

DROP ALIAS

Use this clause to remove an alias name from the disk group directory. Each alias name consists of the full directory path and the alias itself. The underlying file to which the alias refers remains unchanged.

RENAME ALIAS

Use this clause to change the name of an existing alias. The *alias_name* consists of the full directory path and the alias itself.

Restriction on Dropping and Renaming Aliases

You cannot drop or rename a system-generated alias. To determine whether an alias was system generated, query the SYSTEM_CREATED column of the V$ASM_ALIAS dynamic performance view.

diskgroup_volume_clauses

Use these clauses to manipulate logical Oracle ASM Dynamic Volume Manager (Oracle ADVM) volumes corresponding to physical volume devices. To use these clauses, Oracle ASM must be started and the disk group being modified must be mounted.

> **See Also:**
>
> *Oracle Automatic Storage Management Administrator's Guide* for more information about disk group volumes, including examples

add_volume_clause

Use this clause to add a volume to the disk group.

For *asm_volume*, specify the name of the volume. The name can contain only alphanumeric characters and the first character must be alphabetic. The maximum length of the name is platform dependent. Refer to *Oracle Automatic Storage Management Administrator's Guide* for more information.

For *size_clause*, specify the size of the Oracle ADVM volume. The Oracle ASM instance determines whether sufficient space exists to create the volume. If sufficient space does not exist, then the Oracle ASM instance returns an error. If sufficient space does exist, then all nodes in the cluster with an Oracle ASM instance running and the disk group mounted are notified of the addition. Oracle ASM creates and enables on those nodes a volume device that can be used to create and mount file systems.

The following optional settings are also available:

- In the *redundancy_clause*, specify the redundancy level of the Oracle ADVM volume. You can specify this clause only when creating a volume in a normal redundancy disk group. You can specify the following volume redundancy levels:
  - MIRROR: 2-way mirroring of the volume. This is the default.
  - HIGH: 3-way mirroring of the volume.
  - UNPROTECTED: No mirroring of the volume.

  You cannot specify the *redundancy_clause* when creating a volume in a high redundancy disk group or an external redundancy disk group. If you do so, then an error will result. In high redundancy disk groups, Oracle Database automatically sets the volume redundancy to HIGH (3-way mirroring). In external redundancy disk groups, Oracle Database automatically sets the volume redundancy to UNPROTECTED (no mirroring).
- In the STRIPE_WIDTH clause, specify a stripe width for the Oracle ADVM volume. The valid range is from 4KB to 1MB, at intervals of the power of 2. The default value is 128K.

- In the STRIPE_COLUMNS clause, specify the number of stripes in a stripe set of the Oracle ADVM volume. The valid range is 1 to 8. The default is 4. If STRIPE_COLUMNS is set to 1, then striping becomes disabled. In this case, the stripe width is the extent size of the volume. This volume extent size is 64 times the allocation unit (AU) size of the disk group.

modify_volume_clause

Use this clause to modify the characteristics of an existing Oracle ADVM volume. You must specify at least one of the following clauses:

- In the MOUNTPATH clause, specify the mountpath name associated with the volume. The *mountpath_name* can be up to 1024 characters.
- In the USAGE clause, specify the usage name associated with the volume. The *usage_name* can be up to 30 characters.

RESIZE VOLUME Clause

Use this clause to change the size of an existing Oracle ADVM volume. In an Oracle ASM cluster, the new size is propagated to all nodes. If an Oracle Advanced Cluster File System (ACFS) exists on the volume, then you must use the acfsutil size command instead of the ALTER DISKGROUP statement.

> **See Also:**
>
> *Oracle Automatic Storage Management Administrator's Guide* for more information about the acfsutil size command

DROP VOLUME Clause

Use this clause to remove the Oracle ASM file that is the storage container for an existing Oracle ADVM volume. In an Oracle ASM cluster, all nodes with an Oracle ASM instance running and with this disk group open are notified of the drop operation, which results in removal of the volume device. If the volume file is open, then this clause returns an error.

diskgroup_attributes

Use this clause to specify attributes for the disk group. Table 13-2 lists the attributes you can set with this clause. Refer to the CREATE DISKGROUP "ATTRIBUTE Clause " for information on the behavior of this clause.

drop_diskgroup_file_clause

Use this clause to drop a file from the disk group. Oracle ASM also drops all aliases associated with the file being dropped. You must use one of the reference forms of the filename. Most Oracle ASM files do not need to be manually deleted because, as Oracle Managed Files, they are removed automatically when they are no longer needed. Refer to *ASM_filename* for information on the reference forms of Oracle ASM filenames.

You cannot drop a disk group file it if is the spfile that was used to start up the current instance or any instance in the Oracle ASM cluster.

convert_redundancy_clause

You can use this clause to convert a `NORMAL REDUNDANCY` or `HIGH REDUNDANCY` disk group to a `FLEX REDUNDANCY` disk group. The disk group must have at least three failure groups before you start the conversion.

usergroup_clauses

Use these clauses to add a user group to the disk group, remove a user group from the disk group, or to add a member to or drop a member from an existing user group.

> **See Also:**
>
> *Oracle Automatic Storage Management Administrator's Guide* for detailed information about user groups and members, including examples

ADD USERGROUP

Use this clause to add a user group to the disk group. You must have `SYSASM` or `SYSDBA` privilege to create a user group. The maximum length of a user group name is 63 bytes. If you specify the user name, then it must be in the OS password file and its length cannot exceed 32 characters.

MODIFY USERGROUP

Use these clauses to add a member to or drop a member from an existing user group. You must be an Oracle ASM administrator (with SYSASM privilege) or the creator (with SYSDBA privilege) of the user group to use these clauses. The user name must be an existing user in the OS password file.

DROP USERGROUP

Use this clause to drop an existing user group from the disk group. You must be an Oracle ASM administrator (with SYSASM privilege) or the creator (with SYSDBA privilege) of the user group to use this clause. Dropping a user group may leave a disk group file without a valid user group. In this case, you can update the disk group file manually to add a new, valid group using the *file_permissions_clause*.

user_clauses

Use these clauses to add a user to, drop a user from, or replace a user in a disk group.

> **Note:**
>
> When administering users with SQL*Plus, the users must be existing operating system users and their user names must have corresponding operating system user IDs. However, only users in the same cluster as the Oracle ASM instance can be validated.

ADD USER

Use this clause to add one or more operating system (OS) users to an Oracle ASM disk group and give those users access privileges on the disk group. A user name must be an existing user in the OS password file and its length cannot exceed 32 characters. If a specified user already exists in the disk group, as shown by `V$ASM_USER`, then the command records an error and continues to add other users, if any have been specified. This command is seldom needed, because the OS user running the database instance is added to a disk group automatically when the instance accesses the disk group. However, this clause is useful when adding users that are not associated with a particular database instance.

DROP USER

Use this clause to drop one or more users from the disk group. If a specified user is not in the disk group, then this clause records an error and continues to drop other users, if any are specified. If the user owns any files, then you must specify the `CASCADE` keyword, which drops the user and all the user's files. If any files owned by the user are open, then `DROP USER CASCADE` fails with an error.

To delete a user without deleting the files owned by that user, change the owner of each of these files to another user and then issue an `ALTER DISKGROUP` ... `DROP USER` statement on the user. Alternatively, you can issue an `ALTER DISKGROUP` ... `REPLACE USER` statement to replace the user you want to drop with a user that currently does not exist in the disk group. This operation has the side effect of making the new user the owner of files that were previously owned by the dropped user.

REPLACE USER

Use this clause to replace *`old_user`* with *`new_user`* in the disk group. All files that are currently owned by *`old_user`* will become owned by *`new_user`*, and *`old_user`* will be dropped from the disk group. *`old_user`* must exist in the disk group and *`new_user`* must not exist in the disk group.

file_permissions_clause

Use this clause to change the permission settings of a disk group file. The three classes of permissions are owner, user group, and other. You must be the file owner or the Oracle ASM administrator to use this clause.

If you change the permission settings of an open file, then the operation currently running on the file will complete using the old permission settings. The new permission settings will take effect when re-authentication is required.

file_owner_clause

Use this clause to set the owner or user group for a specified file. You must be the Oracle ASM administrator to change the owner of the file. You must be the owner of the file or the Oracle ASM administrator to change the user group of a file. In addition, to change the associated user group of a file, the specified user group must already exist in the disk group, and the owner of the file must be a member of that user group.

If you use this clause on an open file, then the following conditions apply:

- If you change the owner or user group of an open file, then the operation currently running on the file will complete using the old owner or user group. The new owner or user group will take effect when re-authentication is required.
- If you change the owner of an open file, then the new owner of the file cannot be dropped from the disk group until the instance has been restarted. In an Oracle ASM cluster, the new owner of the file cannot be dropped until all instances in the cluster have been restarted.
- If you change the owner of an open file, then the old owner cannot be dropped while the file is still open, even after the ownership of the file has changed.

scrub_clause

Use this clause to scrub a disk group. The scrub operation checks for logical data corruptions and repairs the corruptions automatically in normal and high redundancy disks groups.

- Use the `FILE` clause to scrub the specified Oracle ASM file in the disk group. You must use one of the reference forms of the *`ASM_filename`*. Refer to *ASM_filename* for information on the reference forms of Oracle ASM filenames.
- Use the `DISK` clause to scrub the specified disk in the disk group.
- If you do not specify `FILE` or `DISK`, then all files and disks in the disk group are scrubbed.

REPAIR | NOREPAIR

Specify `REPAIR` to attempt to repair any errors found during the logical data corruption check. Specify `NOREPAIR` to be alerted of any corruptions; Oracle ASM will not take any action to resolve them. The default is `NOREPAIR`.

POWER

Use the `POWER` clause to specify the power level of the scrub operation. Valid values are `AUTO`, `LOW`, `HIGH`, and `MAX`. If you omit this clause, then the power level defaults to `AUTO` and the power adjusts to the optimum level for the system.

WAIT | NOWAIT

Specify `WAIT` to allow the scrub operation to complete before returning control to the user. Specify `NOWAIT` to add the operation to the scrubbing queue and return control to the user immediately. The default is `NOWAIT`.

FORCE | NOFORCE

Specify `FORCE` to process the command even if the system I/O load is high or scrubbing has been disabled at the system level. Specify `NOFORCE` to process the command normally. The default is `NOFORCE`.

STOP

Specify `STOP` if you want to stop an ongoing scrub operation.

You can monitor the progress of the scrub operation by querying the `V$ASM_ OPERATION` dynamic performance view.

> **See Also:**
>
> *Oracle Automatic Storage Management Administrator's Guide* for more information on scrubbing disk groups and "Scrubbing a Disk Group: Example"

quotagroup_clauses

Use these clauses to add a quota group to the disk group, modify a quota group, move a file group into a quota group, or drop a quota group.

A quota group is a collection of file groups. A file group is a container for all files of a database within one disk group. A quota group has a specified quota limit, which is the maximum amount of storage space that its file groups can collectively use. Therefore, a quota group enables you to define the quota limit for a group of databases within a disk group. The sum of the quota limits for all quota groups in a disk group can exceed the storage capacity of the disk group.

Each disk group contains a default quota group named `GENERIC`. If you create a file group and do not specify its quota group, then the file group belongs to the `GENERIC` quota group. Oracle ASM automatically creates the `GENERIC` quota group when you create a disk group with the `compatible.asm` attribute set to `12.2` or higher, or when you set `compatible.asm` to `12.2` or higher for an existing disk group. Initially, the quota limit for `GENERIC` is `UNLIMITED`. You can subsequently modify this quota limit with the `MODIFY QUOTAGROUP` clause.

ADD QUOTAGROUP

Use this clause to create a quota group and add it to the disk group. For *`quotagroup_name`*, specify the name of the new quota group.

The `SET` clause allows you to set the quota limit for the quota group.

- For *`property_name`*, specify `QUOTA`.
- For *`property_value`*, specify one of the following clauses:
  - Specify *`size_clause`* to set a number of bytes for the quota limit. The minimum value you can specify is 1 byte. You can specify a value that is greater than the storage size of the disk group. In this case, storage use is limited by the current size of the disk group. However, if you subsequently increase the storage space for the disk group to a size that exceeds the quota limit, then the quota limit will be enforced. Refer to *size_clause* for the syntax and semantics of this clause. Note that specifying 0 bytes is equivalent to specifying `UNLIMITED`.
  - Specify `UNLIMITED` if you do not want to set a quota limit. In this case, storage use is limited by the storage size of the disk group.

If you omit the `SET` clause, then the default is `SET QUOTA=UNLIMITED`.

MODIFY QUOTAGROUP

Use this clause to modify the quota limit for a quota group. For *`quotagroup_name`*, specify the name of the quota group you want to modify. You can modify the quota limit for any quota group, including the `GENERIC` quota group. The `SET` clause has the same semantics here as for the `ADD QUOTAGROUP` clause. The quota limit can be set below the amount of space currently used by the quota group. This action prevents any additional space from being allocated for files described by file groups associated with this quota group.

MOVE FILEGROUP

Use this clause to move a file group from one quota group to another. For *`filegroup_name`*, specify the file group you want to move. For *`quotagroup_name`*, specify the name of the destination quota group. If the move operation causes the amount of used storage space in the destination quota group to exceed the quota limit, then the operation succeeds, but no new storage allocations can take place in the file groups within the quota group. This capability enables you to stop any files described by a specific file group from allocating additional space.

DROP QUOTAGROUP

Use this clause to drop a quota group from the disk group. For *`quotagroup_name`*, specify the quota group you want to drop. The quota group must not contain any file groups. You cannot drop the quota group `GENERIC`.

See Also:

Automatic Storage Management Administrator's Guide for more information on quota groups

filegroup_clauses

The *filegroup_clauses* are valid only for flex disk groups. Use these clauses to create a file group, modify a file group, move a file into a file group, or drop a file group. A file group is a container for all files of a database within one disk group. A file group must belong to a quota group.

Each disk group has a default file group with FILEGROUP_NUMBER = 0.

add_filegroup_clause

Use this clause to create a file group.

For *filegroup_name*, specify the name of the new file group. The maximum length of a file group name is 127 characters. The name must satisfy the requirements listed in "Database Object Naming Rules ", with the following addition: File group names are not case sensitive, even if you specify them with quotation marks. They are always stored internally as uppercase. File group names must be unique within a disk group.

- Use the DATABASE clause to specify the database (non-CDB, CDB, or PDB) with which the file group is associated.
- Use the CLUSTER clause to specify the cluster with which the file group is associated.
- Use the VOLUME clause to specify the volume with which the file group is associated.
- Use the TEMPLATE clause to create a file group template with which the file group is associated. You can use the template to customize a set of file group properties, that can then be inherited by one or more databases.

You cannot associate more than one file group in the same disk group with the same database, cluster, volume, or template. If the database, cluster, volume, or template does not exist at the time of file group creation, then the file group will be automatically associated with it when it is subsequently created. Database, cluster, volume, and template names must satisfy the requirements listed in "Database Object Naming Rules ".

The SET clause allows you to set properties for the file group. If you do not specify the SET clause for a property, then the default value is assigned. You can specify the *file_type* for any property for which a file type applies. If you do not specify *file_type* for such a property, then the property applies to all file types. For complete information on file group properties and their default values, see *Oracle Automatic Storage Management Administrator's Guide*.

Example 1: Create a file group from a file group template to inherit properties from the template

```
ALTER DISKGROUP hmdg ADD FILEGROUP fgtem TEMPLATE SET 'datafile.redundancy'='unprotected'
    ALTER DISKGROUP hmdg ADD FILEGROUP fgdb DATABASE NONE FROM TEMPLATE fgtem
```

Example 2: Create a file group or a tablespace from a file group template to inherit properties from the template

```
ALTER DISKGROUP hmdg ADD FILEGROUP fgtem2 TEMPLATE
    CREATE TABLESPACE tbs1 datafile '+hmdg(fg$fgtem2)/dbs/tbs1.f' size 1M
```

modify_filegroup_clause

Use this clause to modify file group properties. For *filegroup_name*, specify the name of the file group you want to modify. You can modify properties for any file group, including the default file group. Any that you do not specify with this clause remain unchanged. The SET clause has the same semantics here as for the *add_filegroup_clause*.

move_to_filegroup_clause

Use this clause to move a file to a file group. If the file is currently associated with a different file group, then it is disassociated from that file group. The target file group must have enough space available to contain the file. You must be the owner of the file and the target file group.

drop_filegroup_clause

Use this clause to drop an empty file group. For *filegroup_name*, specify the name of the file group you want to drop.

CASCADE

Use the keyword CASCADE to drop a file group that is not empty. When a file group is dropped with the keyword CASCADE, every file associated with the file group is automatically dropped.

FOR DATABASE Clause

- If the file group being dropped is associated with a pluggable database, then the FOR PLUGGABLE DATABASE clause names the pluggable database in *pdb_name*. You must then specify the FOR DATABASE clause with *db_unique_name*, the name of the container database that contains the pluggable database.
- If the file group being dropped is associated with a non-CDB database, then the FOR DATABASE clause names the non-CDB database in *db_unique_name*.
- You can specify the CASCADE clause with FOR PLUGGABLE DATABASE and FOR DATABASE.

  Use the CASCADE option to delete every file associated with the file group as part of the file group drop.

  When a file group of type DATABASE is dropped using the CASCADE option, ASM checks the OMF (Oracle Managed File) file name of the files being deleted to verify that they belong to the database or pluggable database that is associated with the file group being dropped.

> **See Also:**
>
> *Automatic Storage Management Administrator's Guide* for more information on file groups

undrop_disk_clause

Use this clause to cancel the drop of disks from the disk group. You can cancel the pending drop of all the disks in one or more disk groups (by specifying *diskgroup_name*) or of all the disks in all disk groups (by specifying ALL).

This clause is not relevant for disks that have already been completely dropped from the disk group or for disk groups that have been completely dropped. This clause results in a long-running operation. You can see the status of the operation by querying the `V$ASM_OPERATION` dynamic performance view.

> **See Also:**
>
> `V$ASM_OPERATION` for more information on the details of long-running Oracle ASM operations

diskgroup_availability

Use this clause to make one or more disk groups available or unavailable to the database instances running on the same node as the Oracle ASM instance. This clause does not affect the status of the disk group on other nodes in a cluster.

MOUNT

Specify `MOUNT` to mount the disk groups in the local Oracle ASM instance. Specify `ALL MOUNT` to mount all disk groups specified in the `ASM_DISKGROUPS` initialization parameter. File operations can only be performed when a disk group is mounted. If Oracle ASM is running in a cluster or a standalone server managed by Oracle Grid Infrastructure for a standalone server, then the `MOUNT` clause automatically brings the corresponding resource online.

RESTRICTED | NORMAL

Use these clauses to determine the manner in which the disk groups are mounted.

- In the `RESTRICTED` mode, the disk group is mounted in single-instance exclusive mode. No other Oracle ASM instance in the same cluster can mount that disk group. In this mode the disk group is not usable by any Oracle ASM client.
- In the `NORMAL` mode, the disk group is mounted in shared mode, so that other Oracle ASM instances and clients can access the disk group. This is the default.

FORCE | NOFORCE

Use these clauses to determine the circumstances under which the disk groups are mounted.

- In the `FORCE` mode, Oracle ASM attempts to mount the disk group even if it cannot discover all of the devices that belong to the disk group. This setting is useful if some of the disks in a normal or high redundancy disk group became unavailable while the disk group was dismounted. When `MOUNT FORCE` succeeds, Oracle ASM takes the missing disks offline.

  If Oracle ASM discovers *all* of the disks in the disk group, then `MOUNT FORCE` fails. Therefore, use the `MOUNT FORCE` setting only if some disks are unavailable. Otherwise, use `NOFORCE`.

  In normal- and high-redundancy disk groups, disks from one failure group can be unavailable and `MOUNT FORCE` will succeed. Also in high-redundancy disk groups, two disks in two different failure groups can be unavailable and `MOUNT FORCE` will succeed. Any other combination of unavailable disks causes the operation to fail, because Oracle ASM cannot guarantee that a valid copy of all user data or metadata exists on the available disks.
- In the `NOFORCE` mode, Oracle ASM does not attempt to mount the disk group unless it can discover all the member disks. This is the default.

See Also:

ASM_DISKGROUPS for more information about adding disk group names to the initialization parameter file

DISMOUNT

Specify DISMOUNT to dismount the specified disk groups. Oracle ASM returns an error if any file in the disk group is open unless you also specify FORCE. Specify ALL DISMOUNT to dismount all currently mounted disk groups. File operations can only be performed when a disk group is mounted. If Oracle ASM is running in a cluster or a standalone server managed by Oracle Grid Infrastructure for a standalone server, then the DISMOUNT clause automatically takes the corresponding resource offline.

FORCE

Specify FORCE if you want Oracle ASM to dismount the disk groups even if some files in the disk group are open.

enable_disable_volume

Use this clause to enable or disable one or more volumes in the disk group.

- For each volume you enable, Oracle ASM creates a volume device file on the local node that can be used to create or mount the file system.
- For each volume you disable, Oracle ASM deletes the device file on the local node. If the volume file is open on the local node, then the DISABLE clause returns an error.

Use the ALL keyword to enable or disable all volumes in the disk group. If you specify ALTER DISKGROUP ALL ..., then you must use the ALL keyword in this clause as well.

See Also:

Oracle Automatic Storage Management Administrator's Guide for more information about disk group volumes

Examples

The following examples require a disk group called dgroup_01. They assume that ASM_DISKSTRING is set to /devices/disks/*. In addition, they assume the Oracle user has read/write permission to /devices/disks/d100. Refer to "Creating a Diskgroup: Example" to create dgroup_01.

Adding a Disk to a Disk Group: Example

To add a disk, d100, to a disk group, dgroup_01, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  ADD DISK '/devices/disks/d100';
```

Dropping a Disk from a Disk Group: Example

To drop a disk, dgroup_01_0000, from a disk group, dgroup_01, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  DROP DISK dgroup_01_0000;
```

Undropping a Disk from a Disk Group: Example

To cancel the drop of disks from a disk group, `dgroup_01`, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  UNDROP DISKS;
```

Resizing a Disk Group: Example

To resize every disk in a disk group, `dgroup_01`, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  RESIZE ALL
  SIZE 36G;
```

Rebalancing a Disk Group: Example

To manually rebalance a disk group, `dgroup_01`, and permit Oracle ASM to execute the rebalance as fast as possible, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  REBALANCE POWER 11 WAIT;
```

The `WAIT` keyword causes the database to wait for the disk group to be rebalanced before returning control to the user.

Verifying the Internal Consistency of Disk Group Metadata: Example

To verify the internal consistency of Oracle ASM disk group metadata and instruct Oracle ASM to repair any errors found, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  CHECK ALL
  REPAIR;
```

Adding a Named Template to a Disk Group: Example

To add a named template, `template_01` to a disk group, `dgroup_01`, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  ADD TEMPLATE template_01
    ATTRIBUTES (UNPROTECTED COARSE);
```

Changing the Attributes of a Disk Group Template: Example

To modify the attributes of a system default or user-defined disk group template, `template_01`, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  MODIFY TEMPLATE template_01
    ATTRIBUTES (FINE);
```

Dropping a User-Defined Template from a Disk Group: Example

To drop a user-defined template, `template_01`, from a disk group, `dgroup_01`, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  DROP TEMPLATE template_01;
```

Creating a Directory Path for Hierarchically Named Aliases: Example

To specify the directory structure in which alias names will reside, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  ADD DIRECTORY '+dgroup_01/alias_dir';
```

Creating an Alias Name for an Oracle ASM Filename: Example

To create a user alias by specifying the numeric Oracle ASM filename, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  ADD ALIAS '+dgroup_01/alias_dir/datafile.dbf'
    FOR '+dgroup_01.261.1';
```

Scrubbing a Disk Group: Example

To scrub a disk group, `dgroup_01`, issue the following statement. This statement attempts to repair any errors found during the logical data corruption check and allows the scrub operation to complete before returning control to the user.

```
ALTER DISKGROUP dgroup_01
  SCRUB REPAIR WAIT;
```

Dismounting a Disk Group: Example

To dismount a disk group, `dgroup_01`, issue the following statement. This statement dismounts the disk group even if one or more files are active:

```
ALTER DISKGROUP dgroup_01
  DISMOUNT FORCE;
```

Mounting a Disk Group: Example

To mount a disk group, `dgroup_01`, issue the following statement:

```
ALTER DISKGROUP dgroup_01
  MOUNT;
```

ALTER DOMAIN

Purpose

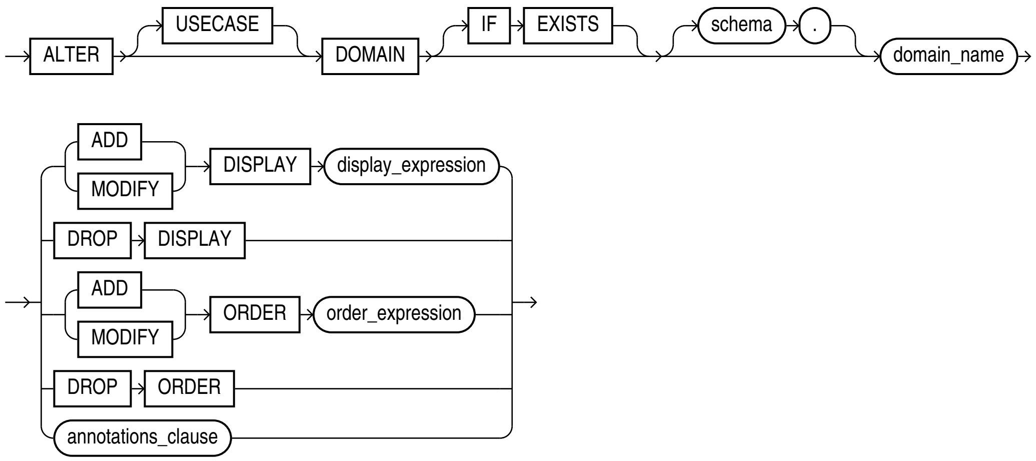
Use this statement to make changes to a domain. When you alter a domain note that checks and catalog changes are made on objects dependent on the domain.

Prerequisites

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

Syntax

alter_domain **::=**



annotations_clause

For the full syntax and semantics of the *`annotations_clause`* see *annotations_clause*.

Semantics

USECASE

This keyword is optional and is provided for semantic clarity. It indicates that the domain is to describe a data use case.

IF EXISTS

Specify `IF EXISTS` to alter an existing domain.

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

ADD DISPLAY

Adds the `display_expression` to the domain. Raises an error if the domain already has a `display_expression` .

Invalidates all SQL statements referencing `DOMAIN_DISPLAY` for an expression of the given domain.

For domain functions see Domain Functions

MODIFY DISPLAY

Changes the domain's display expression to *`display_expression`* and invalidates all SQL statements referencing `DOMAIN_DISPLAY` for an expression of the given domain.

Raises an error if the domain does not have an associated display expression

Invalidates all SQL statements referencing `DOMAIN_DISPLAY` for an expression of the given domain.

Both ALTER DOMAIN ADD DISPLAY and ALTER DOMAIN MODIFY DISPLAY type-check the display expression against all the allowed data types of the domain.

DROP DISPLAY

Raises an error if the domain does not have a display expression. Raises an error If the domain has dependent flexible domain.

Otherwise it removes the display expression from the domain's description, and invalidates all SQL statements referencing DOMAIN_DISPLAY for an expression of the given domain.

ADD MODIFY DROP ORDER

The semantics of these DDLs are the same as for DISPLAY, when translated to the ORDER expression and DOMAIN_ORDER function.

annotations_clause

> **See Also:**
>
> - For the full semantics of the annotations clause see *annotations_clause*.
> - CREATE DOMAIN

Examples

The following statement changes the display expression of the domain day_of_week. It raises an error if the domain does not have a display expression:

```
ALTER DOMAIN day_of_week
      MODIFY DISPLAY LOWER(day_of_week);
```

The following statement removes the display expression from the domain day_of_week. It raises an error if the domain does not have a display expression:

```
ALTER DOMAIN day_of_week
      DROP DISPLAY;
```

The following statement an display expression to the domain day_of_week. It raises an error if the domain already has a display expression:

```
ALTER DOMAIN day_of_week
      ADD DISPLAY INITCAP(day_of_week);
```

The following statement changes the order expression of the domain year_of_birth. It raises an error if the domain does not have an order expression:

```
ALTER DOMAIN year_of_birth
      MODIFY ORDER MOD(year_of_birth,100);
```

The following statement removes the order expression from the domain year_of_birth. It raises an error if the domain does not have an order expression:

```
ALTER DOMAIN year_of_birth
      DROP ORDER;
```

The following statement an order expression to the domain year_of_birth. It raises an error if the domain already has an order expression:

```
ALTER DOMAIN year_of_birth
      ADD ORDER FLOOR(year_of_birth/100);
```

The following example adds the annotation `Display` with the value "`day_of_week`" to the domain. If the domain already has the `Display` annotation it raises an error:

```
ALTER DOMAIN day_of_week
      ANNOTATIONS(Display 'Day of week');
```

ALTER FLASHBACK ARCHIVE

Purpose

Use the `ALTER FLASHBACK ARCHIVE` statement for these operations:

- Designate a flashback archive as the default flashback archive for the system
- Add a tablespace for use by the flashback archive
- Change the quota of a tablespace used by the flashback archive
- Remove a tablespace from use by the flashback archive
- Change the retention period of the flashback archive
- Purge the flashback archive of old data that is no longer needed

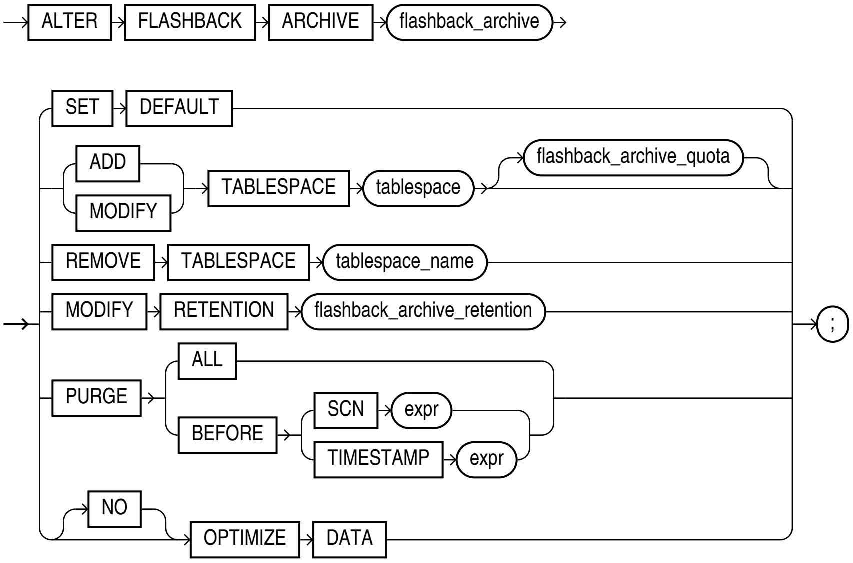
> **See Also:**
>
> *Oracle Database Development Guide* and CREATE FLASHBACK ARCHIVE for more information on using Flashback Time Travel

Prerequisites

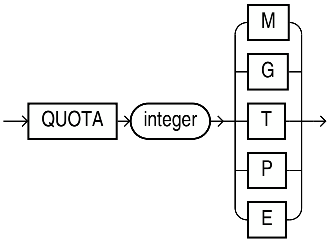
You must have the `FLASHBACK ARCHIVE ADMINISTER` system privilege to alter a flashback archive in any way. You must also have appropriate privileges on the affected tablespaces to add, modify, or remove a flashback archive tablespace.

Syntax

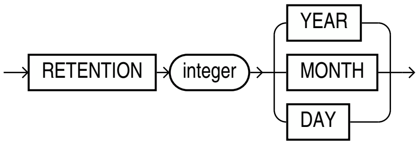
alter_flashback_archive::=



flashback_archive_quota::=



flashback_archive_retention::=



Semantics

flashback_archive

Specify the name of an existing flashback archive.

SET DEFAULT

You must be logged in as `SYSDBA` to specify this clause. Use this clause to designate this flashback archive as the default flashback archive for the system. When a `CREATE TABLE` or

ALTER TABLE statement specifies the *flashback_archive_clause* without specifying a flashback archive name, the database uses the default flashback archive to store data from that table.

This statement overrides any previous designation of a different flashback archive as the default.

> **See Also:**
>
> The CREATE TABLE *flashback_archive_clause* for more information

ADD TABLESPACE

Use this clause to add a tablespace to the flashback archive. You can use the *flashback_archive_quota* clause to specify the amount of space that can be used by the flashback archive in the new tablespace. If you omit that clause, then the flashback archive has unlimited space in the newly added tablespace.

MODIFY TABLESPACE

Use this clause to change the tablespace quota of a tablespace already used by the flashback archive.

REMOVE TABLESPACE

Use this clause to remove a tablespace from use by the flashback archive. You cannot remove the last remaining tablespace used by the flashback archive.

If the tablespace to be removed contains any data within the retention period of the flashback archive, then that data will be dropped as well. Therefore, you should move your data to another tablespace before removing the tablespace with this clause.

MODIFY RETENTION

Use this clause to change the retention period of the flashback archive.

PURGE

Use this clause to purge data from the flashback archive.

- Specify PURGE ALL to remove all data from the flashback archive. This historical information can be retrieved using a flashback query only if the SCN or timestamp specified in the flashback query is within the undo retention duration.
- Specify PURGE BEFORE SCN to remove all data from the flashback archive before the specified system change number.
- Specify PURGE BEFORE TIMESTAMP to remove all data from the flashback archive before the specified timestamp.

[NO] OPTIMIZE DATA

This clause has the same semantics as the [NO] OPTIMIZE DATA clause of CREATE FLASHBACK ARCHIVE.

See Also:

CREATE FLASHBACK ARCHIVE for information on creating flashback archives and for some simple examples of using flashback archives

ALTER FUNCTION

Purpose

Functions 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 FUNCTION` statement to recompile an invalid standalone stored function. Explicit recompilation eliminates the need for implicit run-time recompilation and prevents associated run-time compilation errors and performance overhead.

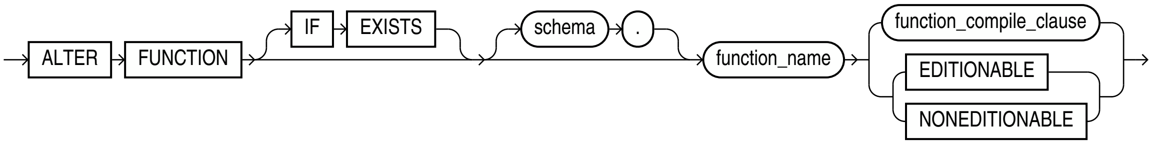
This statement does not change the declaration or definition of an existing function. To redeclare or redefine a function, use the `CREATE FUNCTION` statement with the `OR REPLACE` clause. See CREATE FUNCTION .

Prerequisites

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

Syntax

alter_function::=



(*`function_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 function. If you omit *`schema`*, then Oracle Database assumes the function is in your own schema.

function_name

Specify the name of the function to be recompiled.

function_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 functions.

EDITIONABLE | NONEDITIONABLE

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

ALTER HIERARCHY

Purpose

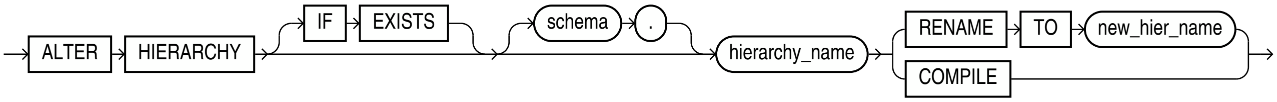
Use the `ALTER HIERARCHY` statement to rename or compile a hierarchy. For other alterations, use `CREATE OR REPLACE HIERARCHY`.

Prerequisites

To alter a hierarchy in your own schema, you must have the `ALTER HIERARCHY` system privilege. To alter a hierarchy in another user's schema, you must have the `ALTER ANY HIERARCHY` system privilege or have been granted `ALTER` directly on the hierarchy.

Syntax

***alter_hierarchy*::=**



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 in which the hierarchy exists. If you do not specify a schema, then Oracle Database looks for the hierarchy in your own schema.

hierarchy_name

Specify the name of the hierarchy.

RENAME TO

Specify RENAME TO to change the name of the hierarchy.

COMPILE

Specify COMPILE to compile the hierarchy.

new_hier_name

Specify a new name for the hierarchy.

Example

The following statement changes the name of a hierarchy:

```
ALTER HIERARCHY product_hier RENAME TO myproduct_hier;
```

ALTER INDEX

Purpose

Use the ALTER INDEX statement to change or rebuild an existing index.

> **See Also:**
>
> CREATE INDEX for information on creating an index

Prerequisites

The index must be in your own schema or you must have the ALTER ANY INDEX system privilege.

To execute the MONITORING USAGE clause, the index must be in your own schema.

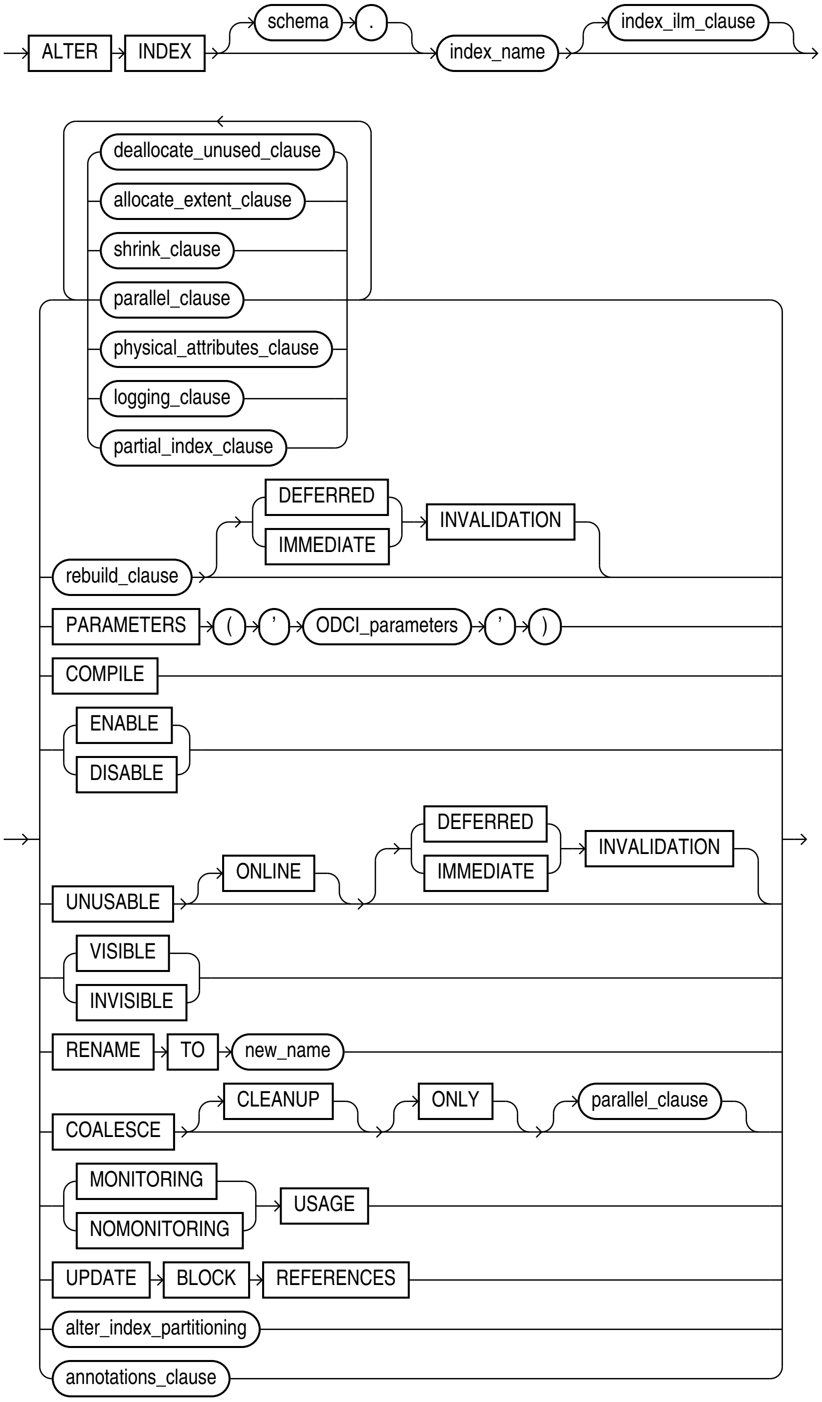
To modify a domain index, you must have EXECUTE object privilege on the indextype of the index.

Object privileges are granted on the parent index, not on individual index partitions or subpartitions.

You must have tablespace quota to modify, rebuild, or split an index partition or to modify or rebuild an index subpartition.

Syntax

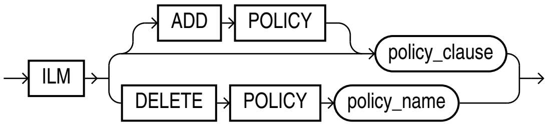
alter_index::=



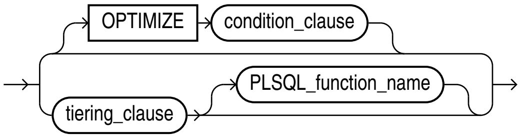
(*deallocate_unused_clause*::=, *allocate_extent_clause*::=, *shrink_clause*::=, *parallel_clause*::=, *physical_attributes_clause*::=, *logging_clause*::=, *partial_index_clause*::=, *rebuild_clause*::=, *alter_index_partitioning*::=)

(The `ODCI_parameters` are documented in Oracle Database Data Cartridge Developer's Guide.)

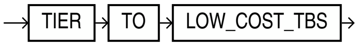
***index_ilm_clause*::=**



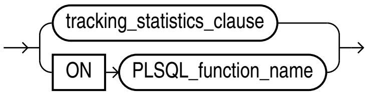
***policy_clause*::=**



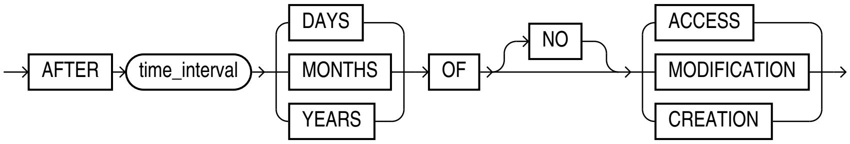
***tiering_clause*::=**



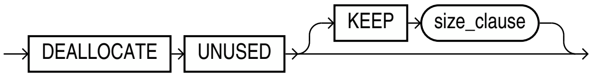
***condition_clause*::=**



***tracking_statistics_clause*::=**

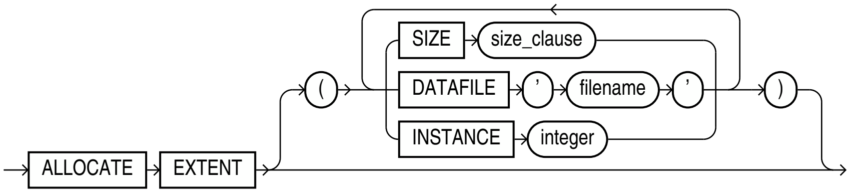


***deallocate_unused_clause*::=**



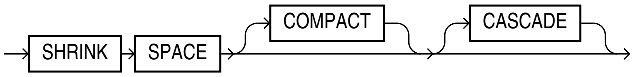
(*size_clause*::=)

***allocate_extent_clause*::=**

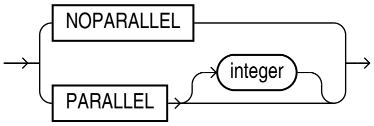


(*size_clause*::=)

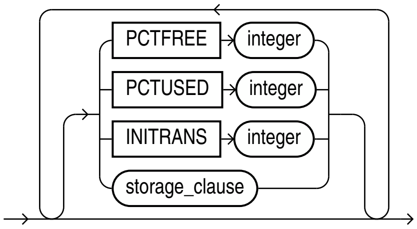
***shrink_clause*::=**



***parallel_clause*::=**

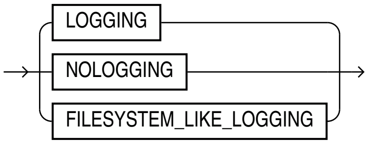


***physical_attributes_clause*::=**

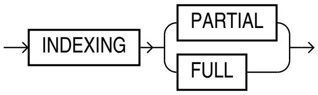


(*storage_clause*::=)

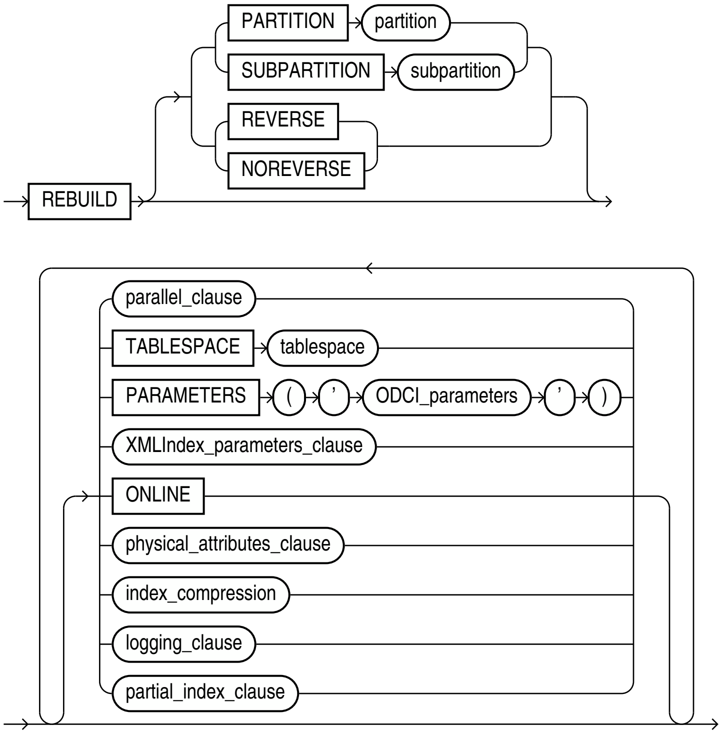
***logging_clause*::=**



***partial_index_clause*::=**



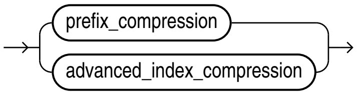
***rebuild_clause*::=**



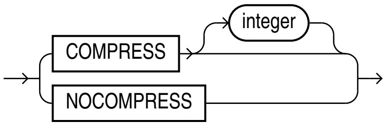
(*parallel_clause*::=, *physical_attributes_clause*::=, *index_compression*::=, *logging_clause*::=, *partial_index_clause*::=)

(The `ODCI_parameters` are documented in Oracle Database Data Cartridge Developer's Guide. The `XMLIndex_parameters_clause` is documented in *Oracle XML DB Developer's Guide*.

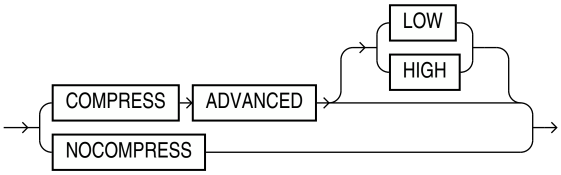
***index_compression*::=**



***prefix_compression*::=**



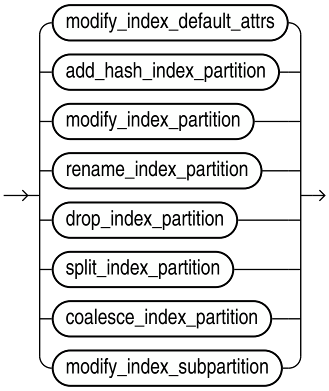
advanced_index_compression::=



annotations_clause::=

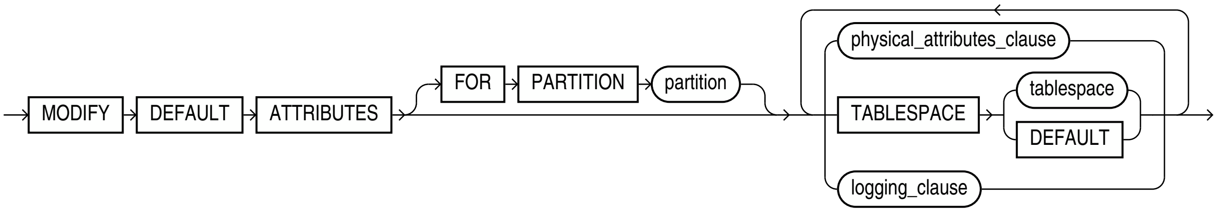
For the full syntax and semantics of the `annotations_clause` see *annotations_clause*.

alter_index_partitioning::=



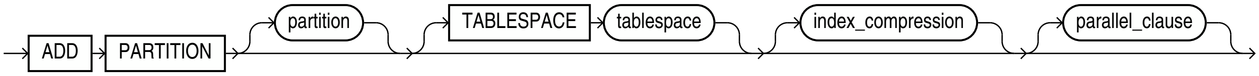
(*modify_index_default_attrs*::=, *add_hash_index_partition*::=, *modify_index_partition*::=, *rename_index_partition*::=, *drop_index_partition*::=, *split_index_partition*::=, *coalesce_index_partition*::=, *modify_index_subpartition*::=)

modify_index_default_attrs::=



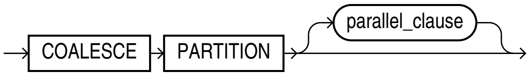
(*physical_attributes_clause*::=, *logging_clause*::=)

***add_hash_index_partition*::=**



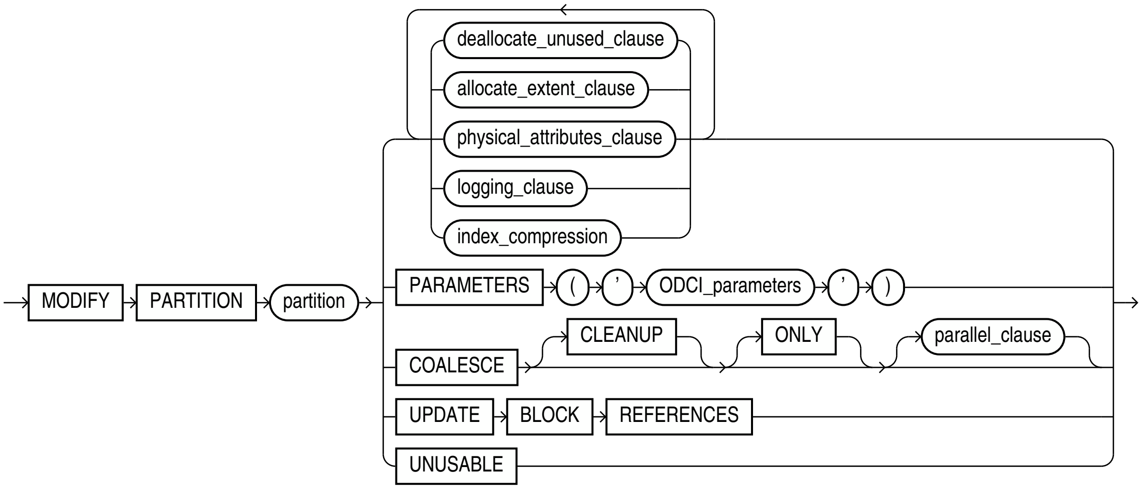
(*index_compression*::=, *parallel_clause*::=)

***coalesce_index_partition*::=**



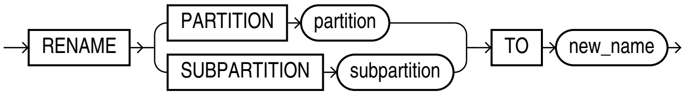
(*parallel_clause*::=)

***modify_index_partition*::=**

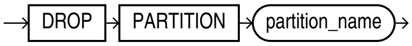


(*deallocate_unused_clause*::=, *allocate_extent_clause*::=, *physical_attributes_clause*::=, *logging_clause*::=, *index_compression*::=)

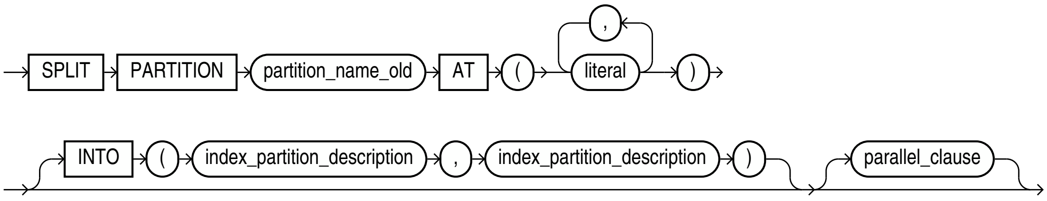
***rename_index_partition*::=**



***drop_index_partition*::=**

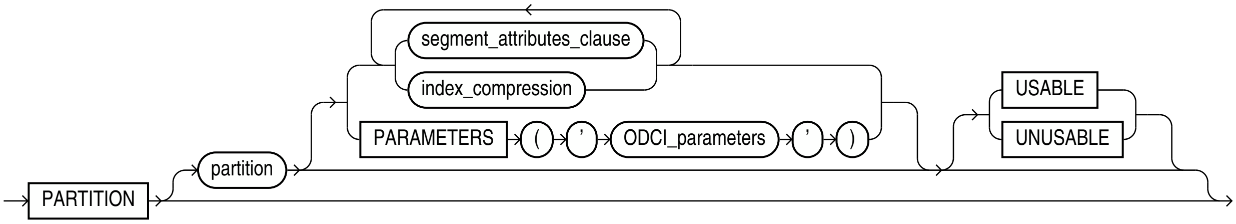


***split_index_partition*::=**



(*parallel_clause::=*)

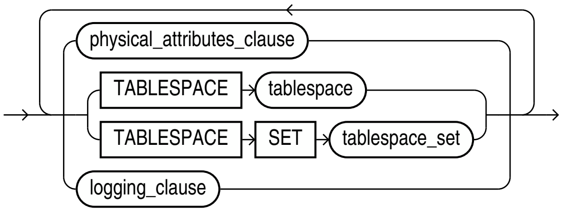
***index_partition_description*::=**



(*segment_attributes_clause::=*, *index_compression::=*)

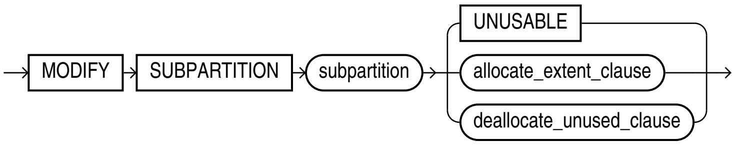
> **Note:**
>
> The `USABLE` and `UNUSABLE` keywords are not supported when *`index_partition_description`* is specified for the *`split_index_partition`* clause.

***segment_attributes_clause*::=**



(*physical_attributes_clause::=*, `TABLESPACE SET`: not supported with `ALTER INDEX`, *logging_clause::=*)

***modify_index_subpartition*::=**



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

Semantics

IF EXISTS

Specify IF EXISTS to alter an existing index.

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

schema

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

index_name

Specify the name of the index to be altered.

Restrictions on Modifying Indexes

The modification of indexes is subject to the following restrictions:

- If *index* is a domain index, then you can specify only the PARAMETERS clause, the RENAME clause, the *rebuild_clause* (with or without the PARAMETERS clause), the *parallel_clause*, or the UNUSABLE clause. No other clauses are valid.
- You cannot alter or rename a domain index that is marked LOADING or FAILED. If an index is marked FAILED, then the only clause you can specify is REBUILD.

> **See Also:**
>
> *Oracle Database Data Cartridge Developer's Guide* for information on the LOADING and FAILED states of domain indexes

index_ilm_clause

Please refer to *index_ilm_clause* in CREATE INDEX for full semantics.

deallocate_unused_clause

Use the *deallocate_unused_clause* to explicitly deallocate unused space at the end of the index and make the freed space available for other segments in the tablespace.

If *index* is range-partitioned or hash-partitioned, then Oracle Database deallocates unused space from each index partition. If *index* is a local index on a composite-partitioned table, then Oracle Database deallocates unused space from each index subpartition.

Restrictions on Deallocating Space

Deallocation of space is subject to the following restrictions:

- You cannot specify this clause for an index on a temporary table.
- You cannot specify this clause and also specify the *rebuild_clause*.

Refer to *deallocate_unused_clause* for a full description of this clause.

KEEP *integer*

The KEEP clause lets you specify the number of bytes above the high water mark that the index will have after deallocation. If the number of remaining extents is less than MINEXTENTS, then MINEXTENTS is set to the current number of extents. If the initial extent becomes smaller than INITIAL, then INITIAL is set to the value of the current initial extent. If you omit KEEP, then all unused space is freed.

Refer to ALTER TABLE for a complete description of this clause.

allocate_extent_clause

The *allocate_extent_clause* lets you explicitly allocate a new extent for the index. For a local index on a hash-partitioned table, Oracle Database allocates a new extent for each partition of the index.

Restriction on Allocating Extents

You cannot specify this clause for an index on a temporary table or for a range-partitioned or composite-partitioned index.

Refer to *allocate_extent_clause* for a full description of this clause.

shrink_clause

Use this clause to compact the index segments. Specifying ALTER INDEX ... SHRINK SPACE COMPACT is equivalent to specifying ALTER INDEX ... COALESCE.

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

Restriction on Shrinking Index Segments

You cannot specify this clause for a bitmap join index or for a function-based index.

parallel_clause

Use the PARALLEL clause to change the default degree of parallelism for queries and DML on the index.

Restriction on Parallelizing Indexes

You cannot specify this clause for an index on a temporary table.

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

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

physical_attributes_clause

Use the *physical_attributes_clause* to change the values of parameters for a nonpartitioned index, all partitions and subpartitions of a partitioned index, a specified partition, or all subpartitions of a specified partition.

> **See Also:**
>
> - the physical attributes parameters in CREATE TABLE
> - "Modifying Real Index Attributes: Example" and "Changing MAXEXTENTS: Example"

Restrictions on Index Physical Attributes

Index physical attributes are subject to the following restrictions:

- You cannot specify this clause for an index on a temporary table.
- You cannot specify the `PCTUSED` parameter at all when altering an index.
- You can specify the `PCTFREE` parameter only as part of the *`rebuild_clause`*, the *`modify_index_default_attrs`* clause, or the *`split_index_partition`* clause.

storage_clause

Use the *`storage_clause`* to change the storage parameters for a nonpartitioned index, index partition, or all partitions of a partitioned index, or default values of these parameters for a partitioned index. Refer to storage_clause for complete information on this clause.

logging_clause

Use the *`logging_clause`* to change the logging attribute of the index. If you also specify the `REBUILD` clause, then this new setting affects the rebuild operation. If you specify a different value for logging in the `REBUILD` clause, then Oracle Database uses the last logging value specified as the logging attribute of the index and of the rebuild operation.

An index segment can have logging attributes different from those of the base table and different from those of other index segments for the same base table.

Restriction on Index Logging

You cannot specify this clause for an index on a temporary table.

> **See Also:**
>
> - *logging_clause* for a full description of this clause
> - *Oracle Database VLDB and Partitioning Guide* for more information about parallel DML

partial_index_clause

Use the *`partial_index_clause`* to change the index to a full index or a partial index. Specify `INDEXING FULL` to change the index to a full index. Specify `INDEXING PARTIAL` to change the index to a partial index. This clause is valid only for indexes on partitioned tables. Refer to the *partial_index_clause* of `CREATE INDEX` for the full semantics of this clause.

RECOVERABLE | UNRECOVERABLE

These keywords are deprecated and have been replaced with `LOGGING` and `NOLOGGING`, respectively. Although `RECOVERABLE` and `UNRECOVERABLE` are supported for backward compatibility, Oracle strongly recommends that you use the `LOGGING` and `NOLOGGING` keywords.

`RECOVERABLE` is not a valid keyword for creating partitioned tables or LOB storage characteristics. `UNRECOVERABLE` is not a valid keyword for creating partitioned or index-organized tables. Also, it can be specified only with the `AS` subquery clause of `CREATE INDEX`.

rebuild_clause

Use the *`rebuild_clause`* to re-create an existing index or one of its partitions or subpartitions. If index is marked `UNUSABLE`, then a successful rebuild will mark it `USABLE`. For a function-based index, this clause also enables the index. If the function on which the index is based does not exist, then the rebuild statement will fail.

> **Note:**
>
> When you rebuild the secondary index of an index-organized table, Oracle Database preserves the primary key columns contained in the logical rowid when the index was created. Therefore, if the index was created with the `COMPATIBLE` initialization parameter set to less than 10.0.0, the rebuilt index will contain the index key and any of the primary key columns of the table that are not also in the index key. If the index was created with the `COMPATIBLE` initialization parameter set to 10.0.0 or greater, then the rebuilt index will contain the index key and all the primary key columns of the table, including those also in the index key.

Restrictions on Rebuilding Indexes

The rebuilding of indexes is subject to the following restrictions:

- You cannot rebuild an index on a temporary table.
- You cannot rebuild a bitmap index that is marked `INVALID`. Instead, you must drop and then re-create it.
- You cannot rebuild an entire partitioned index. You must rebuild each partition or subpartition, as described for the `PARTITION` clause.
- You cannot specify the *`deallocate_unused_clause`* in the same statement as the *`rebuild_clause`*.
- You cannot change the value of the `PCTFREE` parameter for the index as a whole (`ALTER INDEX`) or for a partition (`ALTER INDEX` ... `MODIFY PARTITION`). You can specify `PCTFREE` in all other forms of the `ALTER INDEX` statement.
- For a domain index:
  - You can specify only the `PARAMETERS` clause (either for the index or for a partition of the index) or the *`parallel_clause`*. No other rebuild clauses are valid.
  - You can rebuild an index only if the index is not marked `IN_PROGRESS`.
  - You can rebuild an index partition only if the index is not marked `IN_PROGRESS` or `FAILED` and the partition is not marked `IN_PROGRESS`.

- You cannot rebuild a local index, but you can rebuild a partition of a local index (`ALTER INDEX` ... `REBUILD PARTITION`).
- For a local index on a hash partition or subpartition, the only parameter you can specify is `TABLESPACE`.
- You cannot rebuild an online index that is used to enforce a deferrable unique constraint.

PARTITION Clause

Use the `PARTITION` clause to rebuild one partition of an index. You can also use this clause to move an index partition to another tablespace or to change a create-time physical attribute.

The storage of partitioned database entities in tablespaces of different block sizes is subject to several restrictions. Refer to *Oracle Database VLDB and Partitioning Guide* for a discussion of these restrictions.

Restriction on Rebuilding Partitions

You cannot specify this clause for a local index on a composite-partitioned table. Instead, use the `REBUILD SUBPARTITION` clause.

> **See Also:**
>
> *Oracle Database VLDB and Partitioning Guide* for more information about partition maintenance operations and "Rebuilding Unusable Index Partitions: Example"

SUBPARTITION Clause

Use the `SUBPARTITION` clause to rebuild one subpartition of an index. You can also use this clause to move an index subpartition to another tablespace. If you do not specify `TABLESPACE`, then the subpartition is rebuilt in the same tablespace.

The storage of partitioned database entities in tablespaces of different block sizes is subject to several restrictions. Refer to *Oracle Database VLDB and Partitioning Guide* for a discussion of these restrictions.

Restriction on Modifying Index Subpartitions

The only parameters you can specify for a subpartition are `TABLESPACE`, `ONLINE`, and the *`parallel_clause`*.

REVERSE | NOREVERSE

Indicate whether the bytes of the index block are stored in reverse order:

- `REVERSE` stores the bytes of the index block in reverse order and excludes the rowid when the index is rebuilt.
- `NOREVERSE` stores the bytes of the index block without reversing the order when the index is rebuilt. Rebuilding a `REVERSE` index without the `NOREVERSE` keyword produces a rebuilt, reverse-keyed index.

Restrictions on Reverse Indexes

Reverse indexes are subject to the following restrictions:

- You cannot reverse a bitmap index or an index-organized table.

- You cannot specify REVERSE or NOREVERSE for a partition or subpartition.

> **See Also:**
>
> "Storing Index Blocks in Reverse Order: Example"

parallel_clause

Use the *parallel_clause* to parallelize the rebuilding of the index and to change the degree of parallelism for the index itself. All subsequent operations on the index will be executed with the degree of parallelism specified by this clause, unless overridden by a subsequent data definition language (DDL) statement with the *parallel_clause*. The following exceptions apply:

- If ALTER SESSION DISABLE PARALLEL DDL was specified before rebuilding the index, then the index will be rebuilt serially and the degree of parallelism for the index will be changed to 1.
- If ALTER SESSION FORCE PARALLEL DDL was specified before rebuilding the index, then the index will be rebuilt in parallel and the degree of parallelism for the index will be changed to the value that was specified in the ALTER SESSION statement, or DEFAULT if no value was specified.

> **See Also:**
>
> "Rebuilding an Index in Parallel: Example"

TABLESPACE Clause

Specify the tablespace where the rebuilt index, index partition, or index subpartition will be stored. The default is the default tablespace where the index or partition resided before you rebuilt it.

index_compression

Use the *index_compression* clauses to enable or disable index compression for the index. Specify the *prefix_compression* clause to enable or disable prefix compression for the index. Specify the *advanced_index_compression* clause to enable or disable advanced index compression for the index.

The *index_compression* clauses have the same semantics for CREATE INDEX and ALTER INDEX. For full information on these clauses, refer to *index_compression* in the documentation on CREATE INDEX.

ONLINE Clause

Specify ONLINE to allow DML operations on the table or partition during rebuilding of the index.

Restrictions on Online Indexes

Online indexes are subject to the following restrictions:

- Parallel DML is not supported during online index building. If you specify ONLINE and subsequently issue parallel DML statements, then Oracle Database returns an error.

- You cannot specify ONLINE for a bitmap join index or a cluster index.
- For a nonunique secondary index on an index-organized table, the number of index key columns plus the number of primary key columns that are included in the logical rowid in the index-organized table cannot exceed 32. The logical rowid excludes columns that are part of the index key.

logging_clause

Specify whether the ALTER INDEX ... REBUILD operation will be logged.

Refer to the *logging_clause* for a full description of this clause.

PARAMETERS Clause

This clause is valid only for domain indexes in a top-level ALTER INDEX statement and in the *rebuild_clause*. This clause specifies the parameter string that is passed uninterpreted to the appropriate ODCI indextype routine.

The maximum length of the parameter string is 1000 characters.

If you are altering or rebuilding an entire index, then the string must refer to index-level parameters. If you are rebuilding a partition of the index, then the string must refer to partition-level parameters.

If *index* is marked UNUSABLE, then modifying the parameters alone does not make it USABLE. You must also rebuild the UNUSABLE index to make it usable.

If you have installed Oracle Text, then you can rebuild your Oracle Text domain indexes using parameters specific to that product. For more information on those parameters, refer to *Oracle Text Reference*.

Restriction on the PARAMETERS Clause

You can modify index partitions only if *index* is not marked IN_PROGRESS or FAILED, no index partitions are marked IN_PROGRESS, and the partition being modified is not marked FAILED.

> **See Also:**
>
> - *Oracle Database Data Cartridge Developer's Guide* for more information on indextype routines for domain indexes
> - CREATE INDEX for more information on domain indexes

XMLIndex_parameters_clause

This clause is valid only for XMLIndex indexes. This clause specifies the parameter string that defines the XMLIndex implementation.

The maximum length of the parameter string is 1000 characters.

If you are altering or rebuilding an entire index, then the string must refer to index-level parameters. If you are rebuilding a partition of the index, then the string must refer to partition-level parameters.

If *index* is marked UNUSABLE, then modifying the parameters alone does not make it USABLE. You must also rebuild the UNUSABLE index to make it usable.

> **See Also:**
>
> *Oracle XML DB Developer's Guide* for more information on `XMLIndex`, including the syntax and semantics of the *`XMLIndex_parameters_clause`*

Restriction on the *XMLIndex_parameters_clause*

You can modify index partitions only if *`index`* is not marked `IN_PROGRESS` or `FAILED`, no index partitions are marked `IN_PROGRESS`, and the partition being modified is not marked `FAILED`.

{ DEFERRED | IMMEDIATE } INVALIDATION

This clause lets you control when the database invalidates dependent cursors while rebuilding an index or while marking an index `UNUSABLE`.

- If you specify `DEFERRED INVALIDATION`, then the database avoids or defers invalidating dependent cursors, when possible.
- If you specify `IMMEDIATE INVALIDATION`, then the database immediately invalidates dependent cursors, as it did in Oracle Database 12*c* Release 1 (12.1) and prior releases. This is the default.

If you omit this clause, then the value of the `CURSOR_INVALIDATION` initialization parameter determines when cursors are invalidated.

> **See Also:**
>
> - *Oracle Database SQL Tuning Guide* for more information on cursor invalidation
> - *Oracle Database Reference* for more information in the `CURSOR_INVALIDATION` initialization parameter

COMPILE Clause

Use this clause to recompile an invalid index explicitly. For domain indexes, this clause is useful when the underlying indextype has been altered to support system-managed domain indexes, so that the existing domain index has been marked `INVALID`. In this situation, this `ALTER INDEX` statement migrates the domain index from a user-managed domain index to a system-managed domain index. For all types of indexes, this clause is useful when an index has been marked `INVALID` by an `ALTER TABLE` statement. In this situation, this `ALTER INDEX` statement revalidates the index without rebuilding it.

> **See Also:**
>
> The `CREATE INDEXTYPE` *storage_table_clause* and *Oracle Database Data Cartridge Developer's Guide* for information on creating system-managed domain indexes

ENABLE Clause

ENABLE applies only to a function-based index that has been disabled, either by an ALTER INDEX ... DISABLE statement, or because a user-defined function used by the index was dropped or replaced. This clause enables such an index if these conditions are true:

- The function is currently valid.
- The signature of the current function matches the signature of the function when the index was created.
- The function is currently marked as DETERMINISTIC.

Restrictions on Enabling Function-based Indexes

The ENABLE clause is subject to the following restrictions:

- You cannot specify any other clauses of ALTER INDEX in the same statement with ENABLE.
- You cannot specify this clause for an index on a temporary table. Instead, you must drop and recreate the index. You can retrieve the creation DDL for the index using the DBMS_METADATA package.

DISABLE Clause

DISABLE applies only to a function-based index. This clause lets you disable the use of a function-based index. You might want to do so, for example, while working on the body of the function. Afterward you can either rebuild the index or specify another ALTER INDEX statement with the ENABLE keyword.

USABLE | UNUSABLE

Specify UNUSABLE to mark the index or index partition(s) or index subpartition(s) UNUSABLE. The space allocated for an index or index partition or subpartition is freed immediately when the object is marked UNUSABLE. An unusable index must be rebuilt, or dropped and re-created, before it can be used. While one partition is marked UNUSABLE, the other partitions of the index are still valid. You can execute statements that require the index if the statements do not access the unusable partition. You can also split or rename the unusable partition before rebuilding it. Refer to CREATE INDEX ... USABLE | UNUSABLE for more information.

ONLINE

Specify ONLINE to indicate that DML operations on the table or partition will be allowed while marking the index UNUSABLE. If you specify this clause, then the database will not drop the index segments.

Restrictions on Marking Indexes Unusable

The following restrictions apply to marking indexes unusable:

- You cannot specify UNUSABLE for an index on a temporary table.
- When a global index is marked UNUSABLE during a partition maintenance operation, the database does not drop the unusable index segments.

VISIBLE | INVISIBLE

Use this clause to specify whether the index is visible or invisible to the optimizer. Refer to "VISIBLE | INVISIBLE" in CREATE INDEX for a full description of this clause.

RENAME Clause

Use this clause to rename an index. The *new_index_name* is a single identifier and does not include the schema name.

Restriction on Renaming Indexes

For a domain index, neither *index* nor any partitions of *index* should be in IN_PROGRESS or FAILED state.

> **See Also:**
>
> - *Building Domain Indexes* of the *Data Cartridge Developer's Guide*.
> - *Extensible Indexing Interface* of the *Data Cartridge Developer's Guide*.
> - Renaming an Index: Example

COALESCE Clause

Specify COALESCE to instruct Oracle Database to merge the contents of index blocks where possible to free blocks for reuse.

CLEANUP

Specify CLEANUP to remove orphaned index entries for records that were previously dropped or truncated by a table partition maintenance operation.

To determine whether an index contains orphaned index entries, you can query the ORPHANED_ENTRIES column of the USER_, DBA_, ALL_INDEXES data dictionary views. Refer to *Oracle Database Reference* for more information.

ONLY

Specify ONLY when you want to clean up the index without coalescing the index blocks.

parallel_clause

Use the *parallel_clause* to specify whether to parallelize the coalesce operation.

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

Restrictions on Coalescing Index Blocks

Coalescing of index blocks is subject to the following restrictions:

- You cannot specify this clause for an index on a temporary table.
- Do not specify this clause for the primary key index of an index-organized table. Instead use the COALESCE clause of ALTER TABLE.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information on space management and coalescing indexes
> - COALESCE Clause for information on coalescing the space of an index-organized table
> - *shrink_clause* for an alternative method of compacting index segments

MONITORING USAGE | NOMONITORING USAGE

Use this clause to determine whether Oracle Database should monitor index use.

- Specify MONITORING USAGE to begin monitoring the index. Oracle Database first clears existing information on index use, and then monitors the index for use until a subsequent ALTER INDEX ... NOMONITORING USAGE statement is executed.
- To terminate monitoring of the index, specify NOMONITORING USAGE.

To see whether the index has been used since this ALTER INDEX ... NOMONITORING USAGE statement was issued, query the USED column of the USER_OBJECT_USAGE data dictionary view.

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

UPDATE BLOCK REFERENCES Clause

The UPDATE BLOCK REFERENCES clause is valid only for normal and domain indexes on index-organized tables. Specify this clause to update all the stale guess data block addresses stored as part of the index row with the correct database address for the corresponding block identified by the primary key.

For a domain index, Oracle Database executes the ODCIIndexAlter routine with the alter_option parameter set to AlterIndexUpdBlockRefs. This routine enables the cartridge code to update the stale guess data block addresses in the index.

Restriction on UPDATE BLOCK REFERENCES

You cannot combine this clause with any other clause of ALTER INDEX.

annotations_clause

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

alter_index_partitioning

The partitioning clauses of the ALTER INDEX statement are valid only for partitioned indexes.

The storage of partitioned database entities in tablespaces of different block sizes is subject to several restrictions. Refer to *Oracle Database VLDB and Partitioning Guide* for a discussion of these restrictions.

Restrictions on Modifying Index Partitions

Modifying index partitions is subject to the following restrictions:

- You cannot specify any of these clauses for an index on a temporary table.
- You can combine several operations on the base index into one `ALTER INDEX` statement (except `RENAME` and `REBUILD`), but you cannot combine partition operations with other partition operations or with operations on the base index.

modify_index_default_attrs

Specify new values for the default attributes of a partitioned index.

Restriction on Modifying Partition Default Attributes

The only attribute you can specify for a hash-partitioned global index or for an index on a hash-partitioned table is `TABLESPACE`.

TABLESPACE

Specify the default tablespace for new partitions of an index or subpartitions of an index partition.

logging_clause

Specify the default logging attribute of a partitioned index or an index partition.

Refer to *logging_clause* for a full description of this clause.

FOR PARTITION

Use the `FOR PARTITION` clause to specify the default attributes for the subpartitions of a partition of a local index on a composite-partitioned table.

Restriction on FOR PARTITION

You cannot specify `FOR PARTITION` for a list partition.

> **See Also:**
>
> "Modifying Default Attributes: Example"

add_hash_index_partition

Use this clause to add a partition to a global hash-partitioned index. Oracle Database adds hash partitions and populates them with index entries rehashed from an existing hash partition of the index, as determined by the hash function. If you omit the partition name, then Oracle Database assigns a name of the form `SYS_P`*`n`*. If you omit the `TABLESPACE` clause, then Oracle Database places the partition in the tablespace specified for the index. If no tablespace is specified for the index, then Oracle Database places the partition in the default tablespace of the user, if one has been specified, or in the system default tablespace.

modify_index_partition

Use the *`modify_index_partition`* clause to modify the real physical attributes, logging attribute, or storage characteristics of index partition *`partition`* or its subpartitions. For a hash-partitioned global index, the only subclause of this clause you can specify is `UNUSABLE`.

COALESCE

Specify this clause to merge the contents of index partition blocks where possible to free blocks for reuse.

CLEANUP

Specify CLEANUP to remove orphaned index entries for records that were previously dropped or truncated by a table partition maintenance operation.

To determine whether an index partition contains orphaned index entries, you can query the ORPHANED_ENTRIES column of the USER_, DBA_, ALL_PART_INDEXES data dictionary views. Refer to *Oracle Database Reference* for more information.

UPDATE BLOCK REFERENCES

The UPDATE BLOCK REFERENCES clause is valid only for normal indexes on index-organized tables. Use this clause to update all stale guess data block addresses stored in the secondary index partition.

Restrictions on UPDATE BLOCK REFERENCES

This clause is subject to the following restrictions:

- You cannot specify the *physical_attributes_clause* for an index on a hash-partitioned table.
- You cannot specify UPDATE BLOCK REFERENCES with any other clause in ALTER INDEX.

> **Note:**
>
> If the index is a local index on a composite-partitioned table, then the changes you specify here will override any attributes specified earlier for the subpartitions of index, as well as establish default values of attributes for future subpartitions of that partition. To change the default attributes of the partition without overriding the attributes of subpartitions, use ALTER TABLE ... MODIFY DEFAULT ATTRIBUTES FOR PARTITION.

> **See Also:**
>
> "Marking an Index Unusable: Examples"

UNUSABLE Clause

This clause has the same function for index partitions that it has for the index as a whole. Refer to "USABLE | UNUSABLE".

index_compression

This clause is relevant for composite-partitioned indexes. Use this clause to change the compression attribute for the partition and every subpartition in that partition. Oracle Database marks each index subpartition in the partition UNUSABLE and you must then rebuild these subpartitions. Prefix compression must already have been specified for the index before you can specify the *prefix_compression* clause for a partition, or advanced index compression must have already been specified for the index before you can specify the *advanced_index_compression* clause for a partition. You can specify this clause only at the partition level. You cannot change the compression attribute for an individual subpartition.

You can use this clause for noncomposite index partitions. However, it is more efficient to use the *rebuild_clause* for noncomposite partitions, which lets you rebuild and set the compression attribute in one step.

rename_index_partition

Use the *rename_index_partition* clauses to rename index *partition* or *subpartition* to *new_name*.

Restrictions on Renaming Index Partitions

Renaming index partitions is subject to the following restrictions:

- You cannot rename the subpartition of a list partition.
- For a partition of a domain index, *index* cannot be marked IN_PROGRESS or FAILED, none of the partitions can be marked IN_PROGRESS, and the partition you are renaming cannot be marked FAILED.

> **See Also:**
>
> "Renaming an Index Partition: Example"

drop_index_partition

Use the *drop_index_partition* clause to remove a partition and the data in it from a partitioned global index. When you drop a partition of a global index, Oracle Database marks the next index partition UNUSABLE. You cannot drop the highest partition of a global index.

> **See Also:**
>
> "Dropping an Index Partition: Example"

split_index_partition

Use the *split_index_partition* clause to split a partition of a global range-partitioned index into two partitions, adding a new partition to the index. This clause is not valid for hash-partitioned global indexes. Instead, use the *add_hash_index_partition* clause.

Splitting a partition marked UNUSABLE results in two partitions, both marked UNUSABLE. You must rebuild the partitions before you can use them.

Splitting a partition marked USABLE results in two partitions populated with index data. Both new partitions are marked USABLE.

AT Clause

Specify the new noninclusive upper bound for *split_partition_1*. The *value_list* must evaluate to less than the presplit partition bound for *partition_name_old* and greater than the partition bound for the next lowest partition (if there is one).

INTO Clause

Specify (optionally) the name and physical attributes of each of the two partitions resulting from the split.

> **See Also:**
>
> "Splitting a Partition: Example"

coalesce_index_partition

This clause is valid only for hash-partitioned global indexes. Oracle Database reduces by one the number of index partitions. Oracle Database selects the partition to coalesce based on the requirements of the hash function. Use this clause if you want to distribute index entries of a selected partition into one of the remaining partitions and then remove the selected partition.

modify_index_subpartition

Use the *modify_index_subpartition* clause to mark UNUSABLE or allocate or deallocate storage for a subpartition of a local index on a composite-partitioned table. All other attributes of such a subpartition are inherited from partition-level default attributes.

Examples

Storing Index Blocks in Reverse Order: Example

The following statement rebuilds index ord_customer_ix (created in "Creating an Index: Example") so that the bytes of the index block are stored in reverse order:

```
ALTER INDEX ord_customer_ix REBUILD REVERSE;
```

Rebuilding an Index in Parallel: Example

The following statement causes the index to be rebuilt from the existing index by using parallel execution processes to scan the old and to build the new index:

```
ALTER INDEX ord_customer_ix REBUILD PARALLEL;
```

Modifying Real Index Attributes: Example

The following statement alters the oe.cust_lname_ix index so that future data blocks within this index use 5 initial transaction entries:

```
ALTER INDEX oe.cust_lname_ix
    INITRANS 5;
```

If the oe.cust_lname_ix index were partitioned, then this statement would also alter the default attributes of future partitions of the index. Partitions added in the future would then use 5 initial transaction entries and an incremental extent of 100K.

Enabling Parallel Queries: Example

The following statement sets the parallel attributes for index upper_ix (created in "Creating a Function-Based Index: Example") so that scans on the index will be parallelized:

```
ALTER INDEX upper_ix PARALLEL;
```

Renaming an Index: Example

The following statement renames an index:

```
ALTER INDEX upper_ix RENAME TO upper_name_ix;
```

Marking an Index Unusable: Examples

The following statements use the cost_ix index, which was created in "Creating a Range-Partitioned Global Index: Example". Partition p1 of that index was dropped in "Dropping an Index Partition: Example". The first statement marks index partition p2 as UNUSABLE:

```
ALTER INDEX cost_ix
   MODIFY PARTITION p2 UNUSABLE;
```

The next statement marks the entire index cost_ix as UNUSABLE:

```
ALTER INDEX cost_ix UNUSABLE;
```

Rebuilding Unusable Index Partitions: Example

The following statements rebuild partitions p2 and p3 of the cost_ix index, making the index once more usable: The rebuilding of partition p3 will not be logged:

```
ALTER INDEX cost_ix
   REBUILD PARTITION p2;
ALTER INDEX cost_ix
   REBUILD PARTITION p3 NOLOGGING;
```

Changing MAXEXTENTS: Example

The following statement changes the maximum number of extents for partition p3 and changes the logging attribute:

```
/* This example will fail if the tablespace in which partition p3
   resides is locally managed.
*/
ALTER INDEX cost_ix MODIFY PARTITION p3
   STORAGE(MAXEXTENTS 30) LOGGING;
```

Renaming an Index Partition: Example

The following statement renames an index partition of the cost_ix index (created in "Creating a Range-Partitioned Global Index: Example"):

```
ALTER INDEX cost_ix
   RENAME PARTITION p3 TO p3_Q3;
```

Splitting a Partition: Example

The following statement splits partition p2 of index cost_ix (created in "Creating a Range-Partitioned Global Index: Example") into p2a and p2b:

```
ALTER INDEX cost_ix
   SPLIT PARTITION p2 AT (1500)
   INTO ( PARTITION p2a TABLESPACE tbs_01 LOGGING,
          PARTITION p2b TABLESPACE tbs_02);
```

Dropping an Index Partition: Example

The following statement drops index partition p1 from the cost_ix index:

```
ALTER INDEX cost_ix
   DROP PARTITION p1;
```

Modifying Default Attributes: Example

The following statement alters the default attributes of local partitioned index `prod_idx`, which was created in "Creating an Index on a Hash-Partitioned Table: Example". Partitions added in the future will use 5 initial transaction entries:

```
ALTER INDEX prod_idx
      MODIFY DEFAULT ATTRIBUTES INITRANS 5;
```

ALTER INDEXTYPE

Purpose

Use the `ALTER INDEXTYPE` statement to add or drop an operator of the indextype or to modify the implementation type or change the properties of the indextype.

Prerequisites

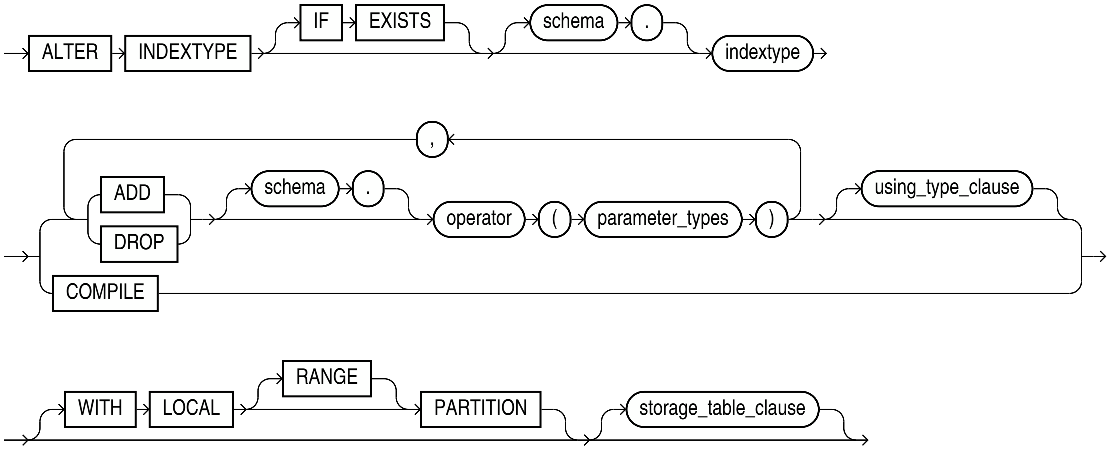
The indextype must be in your own schema or you must have the `ALTER ANY INDEXTYPE` system privilege.

To add a new operator, you must have the `EXECUTE` object privilege on the operator.

To change the implementation type, you must have the `EXECUTE` object privilege on the new implementation type.

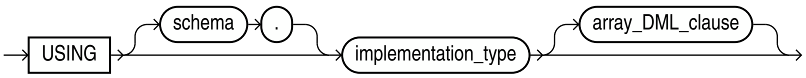
Syntax

***alter_indextype*::=**



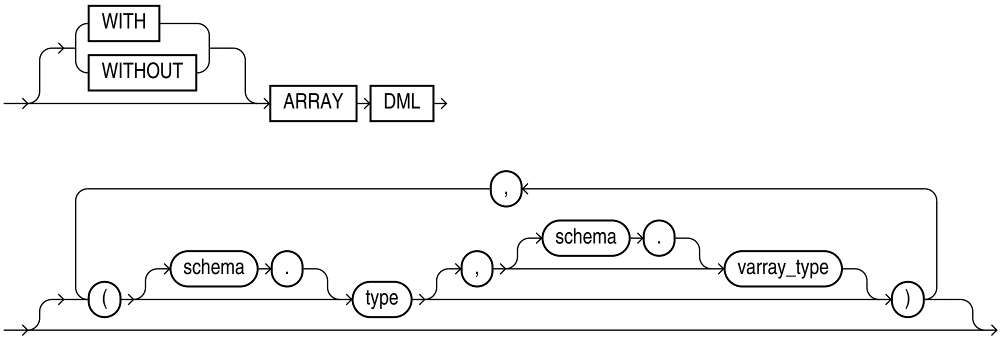
(*using_type_clause*::=, *storage_table_clause*)

***using_type_clause*::=**

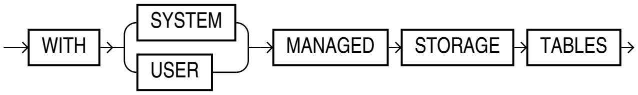


(*array_DML_clause*)

array_DML_clause



storage_table_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 name of the schema in which the indextype resides. If you omit *`schema`*, then Oracle Database assumes the indextype is in your own schema.

indextype

Specify the name of the indextype to be modified.

ADD | DROP

Use the `ADD` or `DROP` clause to add or drop an operator.

No special privilege needed to drop.

- For *`schema`*, specify the schema containing the operator. If you omit *`schema`*, then Oracle assumes the operator is in your own schema.
- For *`operator`*, specify the name of the operator supported by the indextype.

  All the operators listed in this clause must be valid operators.
- For *`parameter_type`*, list the types of parameters to the operator.

using_type_clause

The `USING` clause lets you specify a new type to provide the implementation for the indextype.

array_DML_clause

Use this clause to modify the indextype to support the array interface for the `ODCIIndexInsert` method.

type and *varray_type*

If the data type of the column to be indexed is a user-defined object type, then you must specify this clause to identify the varray *`varray_type`* that Oracle should use to hold column values of *`type`*. If the indextype supports a list of types, then you can specify a corresponding list of varray types. If you omit *`schema`* for either *`type`* or *`varray_type`*, then Oracle assumes the type is in your own schema.

If the data type of the column to be indexed is a built-in system type, then any varray type specified for the indextype takes precedence over the ODCI types defined by the system.

COMPILE

Use this clause to recompile the indextype explicitly. This clause is required only after some upgrade operations, because Oracle Database normally recompiles the indextype automatically.

storage_table_clause

This clause has the same behavior when altering an indextype that it has when you are creating an indextype. Refer to the `CREATE INDEXTYPE` *storage_table_clause* for more information.

WITH LOCAL PARTITION

This clause has the same behavior when altering an indextype that it has when you create an indextype. Refer to the `CREATE INDEXTYPE` clause WITH LOCAL PARTITION for more information.

Examples

Altering an Indextype: Example

The following example compiles the `position_indextype` indextype created in "Creating an Indextype: Example".

```
ALTER INDEXTYPE position_indextype COMPILE;
```

ALTER INMEMORY JOIN GROUP

Purpose

Use the `ALTER INMEMORY JOIN GROUP` statement to add a table column to a join group or remove a table column from a join group.

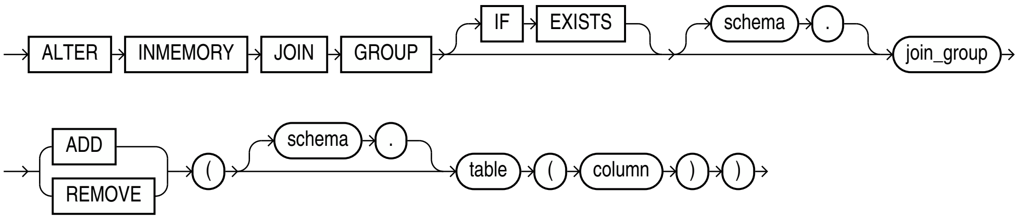
> **See Also:**
>
> - CREATE INMEMORY JOIN GROUP and DROP INMEMORY JOIN GROUP
> - *Oracle Database In-Memory Guide* for more information on join groups

Prerequisites

If the join group is not in your own schema, or if the column you want to add to or remove from the join group is in a table that is not in your own schema, then you must have the `ALTER ANY TABLE` system privilege.

Syntax

***alter_inmemory_join_group*::=**



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

join_group

Specify the name of the join group to be modified.

You can view existing join groups by querying the `DBA_JOINGROUPS` or `USER_JOINGROUPS` data dictionary view. Refer to *Oracle Database Reference* for more information on these views.

ADD

Specify `ADD` to add a table column to the join group. A join group can contain a maximum of 255 columns.

REMOVE

Specify `REMOVE` to remove a table column from the join group. A join group must contain at least 2 columns.

schema

Specify the schema of the table that contains the column to be added to or removed from the join group. If you omit *`schema`*, then Oracle Database assumes the table is in your own schema.

table

Specify the name of the table that contains the column to be added to or removed from the join group.

column

Specify the name of the column to be added to or removed from the join group.

Examples

The following example adds a column to the `prod_id1` join group created in Examples in the documentation on `CREATE INMEMORY JOIN GROUP`:

```
ALTER INMEMORY JOIN GROUP prod_id1
  ADD(product_descriptions(product_id));
```

The following example removes a column from the `prod_id1` join group:

```
ALTER INMEMORY JOIN GROUP prod_id1
  REMOVE(product_descriptions(product_id));
```

ALTER JAVA

Purpose

Use the `ALTER JAVA` statement to force the resolution of a Java class schema object or compilation of a Java source schema object. (You cannot call the methods of a Java class before all its external references to Java names are associated with other classes.)

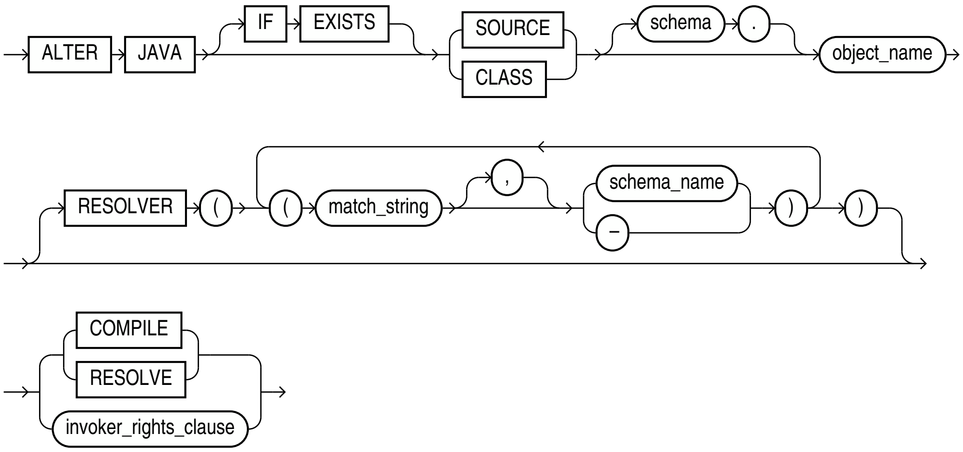
> **See Also:**
>
> *Oracle Database Java Developer's Guide* for more information on resolving Java classes and compiling Java sources

Prerequisites

The Java source or class must be in your own schema, or you must have the `ALTER ANY PROCEDURE` system privilege. You must also have the `EXECUTE` object privilege on Java classes.

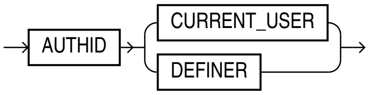
Syntax

***alter_java*::=**



(*invoker_rights_clause::=*)

***invoker_rights_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`.

JAVA SOURCE

Use `ALTER JAVA SOURCE` to compile a Java source schema object.

JAVA CLASS

Use `ALTER JAVA CLASS` to resolve a Java class schema object.

object_name

Specify a previously created Java class or source schema object. Use double quotation marks to preserve lower- or mixed-case names.

RESOLVER

The `RESOLVER` clause lets you specify how schemas are searched for referenced fully specified Java names, using the mapping pairs specified when the Java class or source was created.

See Also:

CREATE JAVA and "Resolving a Java Class: Example"

RESOLVE | COMPILE

RESOLVE and COMPILE are synonymous keywords. They let you specify that Oracle Database should attempt to resolve the primary Java class schema object.

- When applied to a class, resolution of referenced names to other class schema objects occurs.
- When applied to a source, source compilation occurs.

invoker_rights_clause

The *invoker_rights_clause* lets you specify whether the methods of the class execute with the privileges and in the schema of the user who defined it or with the privileges and in the schema of CURRENT_USER.

This clause also determines how Oracle Database resolves external names in queries, DML operations, and dynamic SQL statements in the member functions and procedures of the type.

AUTHID CURRENT_USER

Specify CURRENT_USER if you want the methods of the class to execute with the privileges of CURRENT_USER. This clause is the default and creates an **invoker-rights class**.

This clause also specifies that external names in queries, DML operations, and dynamic SQL statements resolve in the schema of CURRENT_USER. External names in all other statements resolve in the schema in which the methods reside.

AUTHID DEFINER

Specify DEFINER if you want the methods of the class to execute with the privileges of the user who defined the class.

This clause also specifies that external names resolve in the schema where the methods reside.

See Also:

Oracle Database PL/SQL Language Reference for information on how CURRENT_USER is determined

Examples

Resolving a Java Class: Example

The following statement forces the resolution of a Java class:

```
ALTER JAVA CLASS "Agent"
   RESOLVER (("/usr/bin/bfile_dir/*" pm)(* public))
   RESOLVE;
```

ALTER JSON RELATIONAL DUALITY VIEW

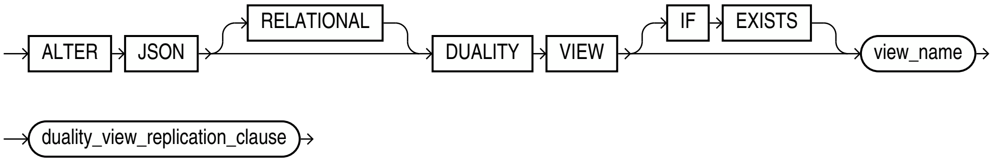
Use ALTER JSON RELATIONAL DUALITY VIEW to alter various options for a duality view like logical replication.

Prerequisites

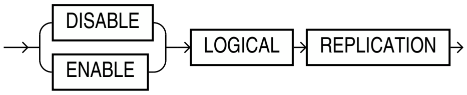
You must have one of the following privileges to use this statement:

- The view must be in your own schema
- You must have the ALTER ANY TABLE system privilege
- You must have the OGG_CAPTURE role

Syntax



duality_view_replication_clause



Semantics

duality_view_replication_clause

Steps to Enable Duality View Replication

- You can enable logical replication for the duality view using ALTER JSON RELATIONAL DUALITY VIEW ENABLE LOGICAL REPLICATION.

  You can also enable logical replication with the command CREATE JSON RELATIONAL DUALITY VIEW
- Minimal (or subset database replication) supplemental logging must be enabled at the database or container level using ALTER PLUGGABLE DATABASE ADD SUPPLEMENTAL LOG DATA DDL.
- Database *compatible* parameter must be 23.4 or higher
- Database parameter at the CDB level *enable_goldengate_replication* must be TRUE

To disable logical replication on a duality view use ALTER JSON RELATIONAL DUALITY VIEW DISABLE LOGICAL REPLICATION

Note:

On a multi instance RAC database, you must run the `ALTER SYSTEM ENABLE RAC TWO_STAGE ROLLING UPDATES ALL` DDL, before you can enable or disable logical replication.

After you run `ALTER SYSTEM ENABLE RAC TWO_STAGE ROLLING UPDATES ALL` you cannot perform an online downgrade (unpatch) of your RAC database to DBRU23.5 or lower. You must take a downtime.

On a single instance database, you do not need to run `ALTER SYSTEM ENABLE RAC TWO_STAGE ROLLING UPDATES ALL`.