SQL (Structured Query Language)

SQL (Structured Query Language) is the set of commands used to access data within the ORACLE database. The intended use of this help manual is as a quick reference guide as it is not fully inclusive of all elements of the SQL language. Please refer to the Oracle7 Server SQL Language Reference Manual for more information.

The SQL Language Quick Reference manual is organized in the following manner:

SQL Commands and Clauses

- Commands (SQL)
- ARCHIVE LOG clause
- CONSTRAINT clause
- DISABLE clause
- DROP clause
- ENABLE clause
- Filespec
- RECOVER clause
- STORAGE clause

Elements of SQL

- Literals (SQL)
- Text
- Number
- Datatypes (SQL)
- Nulls
- Pseudocolumns
- Comments (SQL)

Operators, Functions, Expressions, Conditions

- Operators
- Functions (SQL)
- Format Models
- Expr
- Condition

SQL Commands

The SQL commands listed below are divided into these categories:

* Data Definition Language commands
* Data Manipulation Language commands
* Transaction Control commands
* Session Control commands
* System Control commands

SQL Data Definition Language commands include the following:

ALTER CLUSTER
ALTER DATABASE
ALTER FUNCTION
ALTER INDEX
ALTER PACKAGE
ALTER PROCEDURE
ALTER PROFILE
ALTER RESOURCE COST
ALTER ROLE
ALTER ROLLBACK SEGMENT
ALTER SEQUENCE
ALTER SNAPSHOT
ALTER SNAPSHOT LOG
ALTER TABLE
ALTER TABLESPACE
ALTER TRIGGER
ALTER USER
ALTER VIEW
ANALYZE
AUDIT
COMMENT
CREATE CLUSTER
CREATE CONTROLFILE
CREATE DATABASE
CREATE DATABASE LINK
CREATE FUNCTION
CREATE INDEX
CREATE PACKAGE
CREATE PACKAGE BODY
CREATE PROCEDURE
CREATE PROFILE
CREATE ROLE
CREATE ROLLBACK SEGMENT
CREATE SCHEMA
CREATE SEQUENCE
CREATE SNAPSHOT
CREATE SNAPSHOT LOG
CREATE SYNONYM
CREATE TABLE
CREATE TABLESPACE
CREATE TRIGGER
CREATE USER
CREATE VIEW
DROP CLUSTER
DROP DATABASE LINK
DROP FUNCTION
DROP INDEX
DROP PACKAGE
DROP PROCEDURE
DROP PROFILE
DROP ROLE
DROP ROLLBACK SEGMENT
DROP SEQUENCE
DROP SNAPSHOT
DROP SNAPSHOT LOG
DROP SYNONYM
DROP TABLE
DROP TABLESPACE
DROP TRIGGER
DROP USER
DROP VIEW
GRANT
NOAUDIT
RENAME
REVOKE
TRUNCATE
UPDATE

SQL Data Manipulation Language commands include the following:

DELETE
EXPLAIN PLAN
INSERT
LOCK TABLE
SELECT

SQL Transaction Control commands include the following:

COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

SQL Session Control commands include the following:

ALTER SESSION
SET ROLE

SQL System Control command (only one command):

ALTER SYSTEM

ALTER CLUSTER command

PURPOSE:

To redefine future storage allocations or to allocate an extent for a cluster.

SYNTAX:

```
ALTER CLUSTER [schema.]cluster
    [PCTUSED integer] [PCTFREE integer]
    [SIZE integer [K|M] ]
    [INITRANS integer] [MAXTRANS integer]
    [STORAGE storage_clause]
    [  PARALLEL ( [ DEGREE { integer | DEFAULT } ]
                  [ INSTANCES { integer | DEFAULT } ]
                )
     | NOPARALLEL ]
    [  CACHE | NOCACHE  ]
    [ALLOCATE EXTENT [( [SIZE integer [K|M] ]
                        [DATAFILE 'filename']
                        [INSTANCE integer] )] ]
```

where:

schema
is the schema containing the cluster. If you omit schema, Oracle assumes the cluster is in your own schema.

cluster
is the name of the cluster to be altered.

SIZE
determines how many cluster keys will be stored in data blocks allocated to the cluster. You can only change the SIZE parameter for an indexed cluster, not for a hash cluster. For a description of the SIZE parameter, see the CREATE CLUSTER command.

PCTUSED
PCTFREE
INITRANS
MAXTRANS
changes the values of these parameters for the cluster. See the PCTUSED, PCTFREE, INITRANS, and MAXTRANS parameters of the CREATE TABLE command.

STORAGE
changes the storage characteristics for the cluster. See the STORAGE clause clause.

ALLOCATE EXTENT
explicitly allocates a new extent for the cluster.
SIZE
specifies the size of the extent in bytes. You can

use K or M to specify the extent size in kilobytes or megabytes. If you omit this parameter, Oracle determines the size based on the values of the cluster's STORAGE parameters.

DATAFILE
: specifies one of the data files in the cluster's tablespace to contain the new extent. If you omit this parameter, Oracle chooses the data file.

INSTANCE
: makes the new extent available to the specified instance. An instance is identified by the value of its initialization parameter INSTANCE_NUMBER. If you omit this parameter, the extent is available to all instances. Only use this parameter if you are using Oracle with the Parallel Server option in parallel mode.

Explicitly allocating an extent with this clause does not cause Oracle to evaluate the cluster's storage parameters and determine a new size for the next extent to be allocated. You can only allocate a new extent for an indexed cluster, not a hash cluster.

PARALLEL
: DEGREE specifies the number of query server processes that can scan the cluster in parallel. Either specify a positive integer or DEFAULT which signifies to use the initialization parameter PARALLEL_DEFAULT_SCANSIZE to estimate the number of query servers to use.

INSTANCES specifies the minimum number of instances that need to be available before the cluster can be spread across all available instances of a Parallel Server. A positive integer specifies the number of instances. DEFAULT signifies that the parameter PARALLEL_MAX_PARTITIONSIZE is used to calculate whether a table is split across all instances' buffer caches.

NOPARALLEL
: specifies that queries on this cluster are not performed in parallel by default. A hint in the query still causes the query to be performed in parallel.

CACHE
: specifies that blocks of this cluster are placed on the most recently used end of the LRU list of the buffer cache when the a full table scan is performed.
This option is useful for small lookup tables.

NOCACHE
: specifies that blocks of the cluster in the buffer cache follow the standard LRU algorithm when a full table scan is performed.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the cluster's creation label or you must satisfy one of these criteria:

* If the cluster's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the cluster's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the cluster's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

CREATE CLUSTER, CREATE TABLE, STORAGE clause

ALTER DATABASE command

PURPOSE:

To alter an existing database in one of these ways:

* mount the database
* convert an Oracle Version 6 data dictionary when migrating to Oracle7
* open the database
* choose archivelog or noarchivelog mode for redo log file groups
* perform media recovery
* add or drop a redo log file group or a member of a redo log file group
* rename a redo log file member or a data file
* backup the current control file
* create a new data file in place of an old one for recovery purposes
* take a data file online or offline
* enable or disable a thread of redo log file groups
* change the database's global name
* change the MAC mode
* equate the predefined label DBHIGH or DBLOW with an operating system label

SYNTAX:

```
ALTER DATABASE [database]
    { MOUNT [EXCLUSIVE | PARALLEL]
    | CONVERT
    | OPEN [RESETLOGS | NORESETLOGS]
    | ARCHIVELOG
    | NOARCHIVELOG
    | RECOVER recover_clause
    | ADD LOGFILE [THREAD integer] [GROUP integer] filespec
                                [, [GROUP integer] filespec] ...
    | ADD LOGFILE MEMBER 'filename' [REUSE] [, 'filename' [REUSE]] ...
        TO { GROUP integer
           | ('filename' [,'filename'] ...)
           |  'filename' }
                    [, 'filename' [REUSE] [, 'filename' [REUSE]] ...
        TO { GROUP integer
           | ('filename' [, 'filename'] ...)
           |  'filename' } ] ...
    | DROP LOGFILE {  GROUP integer
                   | ('filename' [, 'filename'] ...)
                   |  'filename' }
               [,  {  GROUP integer
                   | ('filename' [,'filename'] ...)
                   |  'filename' } ] ...
    | DROP LOGFILE MEMBER 'filename' [, 'filename'] ...
    | RENAME FILE 'filename' [, 'filename'] ...
              TO 'filename' [, 'filename'] ...
```

```
  | BACKUP CONTROLFILE TO 'filename' [REUSE]
  | CREATE DATAFILE 'filename' [, filename] ...
               [AS  filespec  [, filespec] ...
  | DATAFILE 'filename' { ONLINE | OFFLINE [DROP] }
  | ENABLE [PUBLIC] THREAD integer
  | DISABLE         THREAD integer
  | RENAME GLOBAL_NAME TO database[.domain]...
  | SET { DBMAC {ON | OFF}
        | DBHIGH = 'text'
        | DBLOW  = 'text' }
  | RESET COMPATIBILITY }
```

where:

database
: identifies the database to be altered. If you omit database, Oracle alters the database identified by the value of the initialization parameter DB_NAME. You can only alter the database whose control files are specified by the initialization parameter CONTROL_FILES. Note that the database identifier is not related to the SQL*Net database specification.

You can only use the following options when the database is not mounted by your instance:

MOUNT
: mounts the database.

  EXCLUSIVE
  : mounts the database in exclusive mode. This mode allows the database to be mounted by only one instance at a time. You cannot use this option if another instance has already mounted the database.

  PARALLEL
  : mounts the database in parallel mode. This mode allows the database to be mounted by multiple instances concurrently. You can only use this option if you are using Oracle with the Parallel Server option. You cannot use this option if another option has mounted the database in exclusive mode.

  The default is EXCLUSIVE.

CONVERT
: completes the conversion of the Oracle Server Version 6 data dictionary. After you use this option, the Version 6 data dictionary no longer exists in the Oracle7 database. Only use this option when you are migrating to Oracle7. For more information on using this option, see the Oracle7 Server Migration Guide.

You can only use the following options when the database is not mounted by your instance:

MOUNT
: mounts the database.

  EXCLUSIVE

mounts the database in exclusive mode. This mode allows the database to be mounted by only one instance at a time. You cannot use this option if another instance has already mounted the database.

PARALLEL
mounts the database in parallel mode. This mode allows the database to be mounted by multiple instances concurrently. You can only use this option if you are using Oracle with the Parallel Server option. You cannot use this option if another option has mounted the database in exclusive mode.

The default is EXCLUSIVE.

CONVERT
completes the conversion of the Oracle Version 6 data dictionary. After you use this option, the Version 6 data dictionary no longer exists in the Oracle7 database. Only use this option when you are migrating to Oracle7. For more information on using this option, see the Oracle7 Server Migration Guide.

You can only use the following option when your instance has the database mounted, but not open:

OPEN
opens the database, making it available for normal use. You must mount the database before you can open it.

RESETLOGS
resets the current log sequence number to 1 and invalidates all redo entries in the online and archived redo log files. You must use this option to open the database after performing media recovery with a backup controlfile. After opening the database with this option, you should perform a complete database backup.

NORESETLOGS
leaves the log sequence number and redo log files in their current state.

You can only specify these options after performing incomplete media recovery. In any other case, Oracle uses the NORESETLOGS automatically.

You can only use the following options when your instance has the database mounted in exclusive mode, but not open:

ARCHIVELOG
establishes archivelog mode for redo log file groups. In this mode, the contents of a redo log file group must be archived before the group can be reused. This option prepares for the possibility of media recovery. You can only use this option after shutting down your instance normally or immediately with no errors and then restarting it, mounting the database in exclusive mode.

NOARCHIVELOG

establishes noarchivelog mode for redo log files. In this mode, the contents of a redo log file group need not be archived so that the group can be reused. This mode does not prepare for recovery after media failure.

You can only use the following option when your instance has the database mounted in exclusive mode:

RECOVER
performs media recovery. You only recover the entire database when the database is closed. You can recover tablespaces or data files when the database is open or closed, provided the tablespaces or data files to be recovered are not being used. You cannot perform media recovery if you are connected to Oracle through the multi-threaded server architecture. You can also perform media recovery with the RECOVER SQL*DBA command.

You can use any of the following options when your instance has the database mounted, open or closed, and the files involved are not in use:

ADD LOGFILE
adds one or more redo log file groups to the specified thread, making them available to the instance assigned the thread. If you omit the THREAD parameter, the redo log file group is added to the thread assigned to your instance. You need only use the THREAD parameter if you are using Oracle with the Parallel Server option in parallel mode.

Each filespec specifies a redo log file group containing one or more members, or copies.

You can choose the value of the GROUP parameter for each redo log file group. Each value uniquely identifies the redo log file group among all groups in all threads and can range from 1 to the MAXLOGFILES value. You cannot add multiple redo log file groups having the same GROUP value. If you omit this parameter, Oracle generates its value automatically. You can examine the GROUP value for a redo log file group through the dynamic performance table.

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

You can specify an existing redo log file group in one of these ways:

GROUP parameter
You can specify the value of the GROUP parameter that identifies the redo log file group.

list of filenames
You can list all members of the redo log file group. You must fully specify each filename according to the

conventions for your operating system.

DROP LOGFILE
drops all members of a redo log file group. You can specify a redo log file group in the same manners as the ADD LOGFILE MEMBER clause. You cannot drop a redo log file group if all of its members have been lost through media failure.

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

You cannot use this clause to drop all members of a redo log file group that contain valid data. To perform this operation, use the DROP LOGFILE clause.

RENAME FILE
renames data files or redo log file members. This clause only renames files in the control file, it does not actually rename them on your operating system. You must specify each filename using the conventions for filenames on your operating system.

BACKUP CONTROLFILE
backs up the current control file to the specified 'filename'. If the backup file already exists, you must specify the REUSE option.

CREATE DATAFILE
creates a new data file in place of an old one. You can use this option to recreate a data file that was lost with no backup. The 'filename' must identify a file that is or was once part of the database. The filespec specifies the name and size of the new data file. If you omit the AS clause, Oracle creates the new file with the same name and size as the file specified by 'filename'.

Oracle 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.

You cannot create a new file based on the first data file of the SYSTEM tablespace unless the database was created in archivelog mode.

DATAFILE
takes a data file online or offline. If any other instance has the database open, your instance must also have the database open:

ONLINE
brings the data file online.

OFFLINE
takes the data file offline.

DROP
takes a data file offline when the database is in noarchivelog mode.

You can only use the following options when your instance has the

database open:

ENABLE
enables the specified thread of redo log file groups. The thread must have at least two redo log file groups before you can enable it.

PUBLIC
makes the enabled thread available to any instance that does not explicitly request a specific thread with the initialization parameter THREAD.

If you omit the PUBLIC option, the thread is only available to the instance that explicitly requests it with the initialization parameter THREAD.

DISABLE
disables the specified thread, making it unavailable to all instances. You cannot disable a thread if an instance using it has the database mounted.

RENAME GLOBAL_NAME
changes the global name of the database. The database is the new database name and can be as long as eight bytes. The optional domains specifies where the database is effectively located in the network hierarchy. Renaming your database automatically clears all data from the shared pool in the SGA. However, 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.

SET
changes one of the following for your database:

DBMAC
changes the mode in which Trusted Oracle is configured:

ON
configures Trusted Oracle in DBMS MAC mode.

OFF
configures Trusted Oracle in OS MAC mode.

DBHIGH
equates the predefined label DBHIGH to the operating system label specified by 'text'.

DBLOW
equates the predefined label DBLOW to the operating system label specified by 'text'.

You must specify labels in the default label format for your session. Changes made by this option take effect when you next start your instance. You can only use this clause if you are using Trusted Oracle.

RESET COMPATIBILITY

Issue the ALTER DATABASE RESET COMPATIBILITY command when restarting the database with the COMPATIBILE initialization parameter set to an earlier release.

PREREQUISITES:

You must have ALTER DATABASE system privilege.

SEE:

CREATE DATABASE, RECOVER clause

ALTER FUNCTION command

PURPOSE:

To recompile a stand-alone stored function.

SYNTAX:

```
ALTER FUNCTION [schema.]function
     COMPILE
```

where:

schema
is the schema containing the function. If you omit schema, Oracle assumes the function is in your own schema.

function
is the name of the function to be recompiled.

COMPILE
causes Oracle to recompile the function. The COMPILE keyword is required.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the function's creation label or you must satisfy one of these criteria:

* If the function's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the function's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the function's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
ALTER PROCEDURE, CREATE FUNCTION

ALTER INDEX command

PURPOSE:

To change future storage allocation for data blocks in an index.

SYNTAX:

```
ALTER INDEX [schema.]index
    [INITRANS integer] [MAXTRANS integer]
    [STORAGE storage_clause]
```

where:

schema
is the schema containing the index. If you omit schema, Oracle assumes the index is in your own schema.

index
is the name of the index to be altered.

INITRANS
MAXTRANS
changes the values of these parameters for the index. See the INITRANS and MAXTRANS parameters of the CREATE TABLE command.

STORAGE
changes the storage parameters for the index. See the STORAGE clause.

PREREQUISITES:

The index must be in your own schema or you must have ALTER ANY INDEX system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the index's creation label or you must satisfy one of these criteria:

* If the index's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the index's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the index's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE INDEX, CREATE TABLE, STORAGE clause

ALTER PACKAGE command

PURPOSE:

To recompile a stored package.

SYNTAX:

```
ALTER PACKAGE [schema.]package
    COMPILE [PACKAGE | BODY]
```

where:

schema
is the schema containing the package. If you omit schema, Oracle assumes the package is in your own schema.

package
is the name of the package to be recompiled.

COMPILE
recompiles the package specification or body. The COMPILE keyword is required.

PACKAGE
recompiles the package body and specification.

BODY
recompiles only the package body.

The default option is PACKAGE.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the package's creation label or you must satisfy one of these criteria:

* If the package's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the package's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the package's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE PACKAGE, CREATE PACKAGE BODY

ALTER PROCEDURE command

PURPOSE:

To recompile a stand-alone stored procedure.

SYNTAX:

```
ALTER PROCEDURE [schema.]procedure
    COMPILE
```

where:

schema
is the schema containing the procedure. If you omit schema, Oracle assumes the procedure is in your own schema.

procedure
is the name of the procedure to be recompiled.

COMPILE
causes Oracle to recompile the procedure. The COMPILE keyword is required.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the procedure's creation label or you must satisfy one of these criteria:

* If the procedure's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the procedure's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the procedure's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
ALTER FUNCTION, ALTER PACKAGE, CREATE PROCEDURE

ALTER PROFILE command

PURPOSE:

To add, modify, or remove a resource limit in a profile.

SYNTAX:

```
ALTER PROFILE profile
    LIMIT   [SESSIONS_PER_USER          {integer | UNLIMITED | DEFAULT}]
            [CPU_PER_SESSION            {integer | UNLIMITED | DEFAULT}]
            [CPU_PER_CALL               {integer | UNLIMITED | DEFAULT}]
            [CONNECT_TIME               {integer | UNLIMITED | DEFAULT}]
            [IDLE_TIME                  {integer | UNLIMITED | DEFAULT}]
            [LOGICAL_READS_PER_SESSION  {integer | UNLIMITED | DEFAULT}]
            [LOGICAL_READS_PER_CALL     {integer | UNLIMITED | DEFAULT}]
            [COMPOSITE_LIMIT            {integer | UNLIMITED | DEFAULT}]
            [PRIVATE_SGA          {integer [K|M] | UNLIMITED | DEFAULT}]
```

where:

profile
is the name of the profile to be altered.

integer
defines a new limit for a resource in this profile. For information on resource limits, see the CREATE PROFILE command.

UNLIMITED
specifies that this profile allows unlimited use of the resource.

DEFAULT
removes a resource limit from the profile. Any user assigned the profile is subject to the limit on the resource defined in the DEFAULT profile in their subsequent sessions.

PREREQUISITES:

You must have ALTER PROFILE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the profile's creation label or you must satisfy one of these criteria:

* If the profile's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the profile's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the profile's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE PROFILE

ALTER RESOURCE COST command

PURPOSE:

To specify a formula to calculate the total resource cost used in a session. For any session, this cost is limited by the value of the COMPOSITE_LIMIT parameter in the user's profile.

SYNTAX:

```
ALTER RESOURCE
    COST    [CPU_PER_SESSION              integer]
            [CONNECT_TIME                 integer]
            [LOGICAL_READS_PER_SESSION    integer]
            [PRIVATE_SGA                  integer]
```

where:

integer
is the weight of each resource.

CPU_PER_SESSION
is the amount of CPU time used by a session measured in hundredths of seconds.

CONNECT_TIME
is the elapsed time of a session measured in minutes.

LOGICAL_READS_PER_SESSION
is the number of data blocks read during a session, including blocks read from both memory and disk.

PRIVATE_SGA
is the number of bytes of private space in the System Global Area (SGA) used by a session. This limit only applies if you are using the multi-threaded server architecture and allocating private space in the SGA for your session.

PREREQUISITES:

You must have ALTER RESOURCE COST system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match DBLOW or you must have WRITEDOWN system privileges.

SEE:
CREATE PROFILE

ALTER ROLE command

PURPOSE:

To change the authorization needed to enable a role.

SYNTAX:

```
ALTER ROLE role
    { NOT IDENTIFIED
    | IDENTIFIED {BY password | EXTERNALLY }
```

where:

role
is the name of the role to be created. Oracle Corporation recommends that the role contain at least one single-byte character regardless of whether the database character set also contains multi-byte characters.

NOT IDENTIFIED
indicates that a user granted the role need not be verified when enabling it.

IDENTIFIED
indicates that a user granted the role must be verified when enabling it with the SET ROLE command:

BY password
The user must specify the password to Oracle when enabling the role. The password can only contain single-byte characters from your database character set regardless of whether this character set also contains multi-byte characters.

EXTERNALLY
The operating system verifies the user enabling to the role. Depending on the operating system, the user may have to specify a password to the operating system when enabling the role.

If you omit both the NOT IDENTIFIED option and the IDENTIFIED clause, the role defaults to NOT IDENTIFIED.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the role's creation label or you must satisfy one of these criteria:

* If the role's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the role's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.

* If the role's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

CREATE ROLE, SET ROLE

ALTER ROLLBACK SEGMENT command

PURPOSE:

To alter a rollback segment in one of these ways:

* by bringing it online
* by taking it offline
* by changing its storage characteristics

SYNTAX:

```
ALTER ROLLBACK SEGMENT rollback_segment
    { ONLINE
    | OFFLINE
    | STORAGE storage_clause }
```

where:

rollback_segment
specifies the name of an existing rollback segment.

ONLINE
brings the rollback segment online.

OFFLINE
takes the rollback segment offline.

STORAGE
changes the rollback segment's storage characteristics.

PREREQUISITES:

You must have ALTER ROLLBACK SEGMENT system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the rollback segment's creation label or you must satisfy one of these criteria:

* If the rollback segment's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the rollback segment's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the rollback segment's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE ROLLBACK SEGMENT, CREATE TABLESPACE, STORAGE clause

ALTER SEQUENCE command

PURPOSE:

To change the sequence in one of these ways:

* changing the increment between future sequence values
* setting or eliminating the minimum or maximum value
* changing the number of cached sequence numbers
* specifying whether or not sequence numbers must be ordered

SYNTAX:

```
ALTER SEQUENCE [schema.]sequence
    [INCREMENT BY integer]
    [MAXVALUE integer | NOMAXVALUE]
    [MINVALUE integer | NOMINVALUE]
    [CYCLE | NOCYCLE]
    [CACHE integer | NOCACHE]
    [ORDER | NOORDER]
```

where:

schema
is the schema to contain the sequence. If you omit schema, Oracle creates the sequence in your own schema.

sequence
is the name of the sequence to be created.

INCREMENT BY
specifies the interval between sequence numbers. This value can be any positive or negative Oracle integer, but it cannot be 0. If this value is negative, then the sequence descends. If the increment is positive, then the sequence ascends. If you omit this clause, the interval defaults to 1.

MINVALUE
specifies the sequence's minimum value.

NOMINVALUE
specifies a minimum value of 1 for an ascending sequence or -10 for a descending sequence.

The default is NOMINVALUE.

MAXVALUE
specifies the maximum value the sequence can generate.

NOMAXVALUE
specifies a maximum value of 10 for a descending sequence.

The default is NOMAXVALUE.

START WITH
specifies the first sequence number to be generated. You can use this option to start an ascending sequence at a value greater than its minimum or to start a descending sequence at a value less than its maximum. For ascending sequences, the default value is the sequence's minimum value. For descending sequences, the default value is the sequence's maximum value.

CYCLE
specifies that the sequence continues to generate values after reaching either its maximum or minimum value. After an ascending sequence reaches its maximum value, it generates its minimum value. After a descending sequence reaches its minimum, it generates its maximum.

NOCYCLE
specifies that the sequence cannot generate more values after reaching its maximum or minimum value.

The default is NOCYCLE.

CACHE
specifies how many values of the sequence Oracle preallocates and keeps in memory for faster access. The minimum value for this parameter is 2. For sequences that cycle, this value must be less than the number of values in the cycle.

NOCACHE
specifies that values of the sequence are not preallocated.

If you omit both the CACHE parameter and the NOCACHE option, Oracle caches 20 sequence numbers by default. However, if you are using Oracle with the Parallel Server option in parallel mode and you specify the ORDER option, sequence values are never cached, regardless of whether you specify the CACHE parameter or the NOCACHE option.

ORDER
guarantees that sequence numbers are generated in order of request. You may want to use this option if you are using the sequence numbers as timestamps. Guaranteeing order is usually not important for sequences used to generate primary keys.

NOORDER
does not guarantee sequence numbers are generated in order of request.

If you omit both the ORDER and NOORDER options, Oracle chooses NOORDER by default. Note that the ORDER option is only necessary to guarantee ordered generation if you are using Oracle with the Parallel Server option in parallel mode. If you are using exclusive mode, sequence numbers are always generated in order.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the sequence's creation label or you must satisfy one of these criteria:

* If the sequence's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the sequence's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the sequence's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

CREATE SEQUENCE, DROP SEQUENCE

ALTER SESSION command

PURPOSE:

To alter your current session in one of these ways:

* to enable or disable the SQL trace facility
* to change the values of NLS parameters
* to change your DBMS session label in Trusted Oracle
* to change the default label format for your session
* to close a database link
* to send advice to remote databases for forcing an in-doubt distributed transaction
* to permit or prohibit procedures and stored functions from issuing COMMIT and ROLLBACK statements
* to change the goal of the cost-based optimization approach

SYNTAX:

```
ALTER SESSION
   { SET { SQL_TRACE              = { TRUE | FALSE }
         | GLOBAL_NAMES           = { TRUE | FALSE }
         | NLS_LANGUAGE           =   language
         | NLS_TERRITORY          =   territory
         | NLS_DATE_FORMAT        =  'fmt'
         | NLS_DATE_LANGUAGE      =   language
         | NLS_NUMERIC_CHARACTERS =  'text'
         | NLS_ISO_CURRENCY       =   territory
         | NLS_CURRENCY           =  'text'
         | NLS_SORT               = { sort  | BINARY }
         | LABEL                  = {'text' | DBHIGH | DBLOW | OSLABEL }
         | MLS_LABEL_FORMAT       =  'fmt'
         | OPTIMIZER_GOAL         = { RULE | ALL_ROWS | FIRST_ROWS | CHOOSE }
         | FLAGGER                = { ENTRY | INTERMEDIATE | FULL | OFF }
         | CLOSE_CACHED_OPEN_CURSORS = { TRUE | FALSE }
   } ...
   | CLOSE DATABASE LINK dblink
   | ADVISE {COMMIT | ROLLBACK | NOTHING}
   | {ENABLE | DISABLE} COMMIT IN PROCEDURE }
```

where:

SQL_TRACE
controls the SQL trace facility for your session:
TRUE
enables the SQL trace facility.
FALSE
disables the SQL trace facility.

GLOBAL_NAMES
controls the enforcement of global name resolution for your session:
TRUE
enables the enforcement of global name resolution.
FALSE

disables the enforcement of global name resolution.

For information on enabling and disabling global name resolution with this parameter, see the ALTER SYSTEM command.

NLS_LANGUAGE
changes the language in which Oracle returns errors and other messages. This parameter also implicitly specifies new values for these items:

* language for day and month names and abbreviations and spelled values of other date format elements
* sort sequence

NLS_TERRITORY
implicitly specifies new values for these items:

* default date format
* decimal character and group separator
* local currency symbol
* ISO currency symbol
* first day of the week for D date format element

NLS_DATE_FORMAT
explicitly specifies a new default date format.

NLS_DATE_LANGUAGE
explicitly changes the language for day and month names and abbreviations and spelled values of other date format elements.

NLS_NUMERIC_CHARACTERS
explicitly specifies a new decimal character and group separator. The 'text' value must have this form:

'dg'

where:

d
is the new decimal character.

g
is the new group separator.

The decimal character and the group separator must be different and can only be single-byte characters.

NLS_ISO_CURRENCY
explicitly specifies the territory whose ISO currency symbol should be used.

NLS_CURRENCY
explicitly specifies a new local currency symbol.

NLS_SORT
changes the sequence into which Oracle sorts character values.

sort
specifies the name of a linguistic sort sequence.

BINARY

specifies a binary sort.

LABEL
changes your DBMS session label to either:

* the label specified by 'text' in your session's default label format
* the label equivalent to DBHIGH
* the label equivalent to DBLOW
* your operating system label using OSLABEL

MLS_LABEL_FORMAT
changes the default label format for your session.

OPTIMIZER_GOAL
specifies the approach and goal of the optimizer for your session:

RULE
specifies the rule-based approach.

ALL_ROWS
specifies the cost-based approach and optimizes for best throughput.

FIRST_ROWS
specifies the cost-based approach and optimizes for best response time.

CHOOSE
causes the optimizer to choose an optimization approach based on the presence of statistics in the data dictionary.

FLAGGER
specifies that non-SQL92 compliant syntax should be flagged for the session. Oracle flags any non-standard constructs as errors and displays the violating syntax.
Currently there is no difference between entry, intermediate, and full level flagging. These options will become significant as Oracle conforms to SQL92 intermediate and full level standards.
OFF disables FIPS flagging.

CLOSE_CACHED_OPEN_CURSORS
controls whether cursors opened and cached in memory by PL/SQL are automatically closed at each COMMIT. A value of FALSE signifies that cursors opened by PL/SQL are held open so that subsequent executions need not open a new cursor. A value of TRUE causes open cursors to be closed at each COMMIT or ROLLBACK.

CLOSE DATABASE LINK
closes the database link dblink, eliminating your session's connection to the remote database. The database link cannot be currently in use by an active transaction or an open cursor.

ADVISE
sends advice for forcing a distributed transaction to a remote database. This advice appears on the remote database in the ADVICE column of the DBA_2PC_PENDING data dictionary view in the event the distributed transaction becomes in-doubt. The following are advice options:

COMMIT
places the value 'C' in DBA_2PC_PENDING.ADVICE.
ROLLBACK
places the value 'R' in DBA_2PC_PENDING.ADVICE.
NOTHING
places the value ' ' in DBA_2PC_PENDING.ADVICE.

COMMIT IN PROCEDURE
specifies whether procedures and stored functions can issue COMMIT and ROLLBACK statements:
ENABLE
permits procedures and stored functions to issue these statements.
DISABLE
prohibits procedures and stored functions from issuing these statements.

PREREQUISITES:

To enable and disable the SQL trace facility or to change the default label format, you must have ALTER SESSION system privilege.

To raise your session label, you must have WRITEUP and READUP system privileges. To lower your session label, you must have WRITEDOWN system privilege. To change your session label laterally, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

To perform the other operations of this command, you do not need any privileges.

SEE:
Tuning SQL Statements and Appendix B of the Oracle Server Application Developer's Guide.

ALTER SNAPSHOT command

PURPOSE:

To alter a snapshot in one of these ways:

* changing its storage characteristics
* changing its automatic refresh mode and times

SYNTAX:

```
ALTER SNAPSHOT [schema.]snapshot
    [ PCTFREE  integer | PCTUSED  integer
    | INITRANS integer | MAXTRANS integer
    | STORAGE storage_clause ] ...
    [ USING INDEX [   INITTRANS integer | MAXTRANS integer
                    | STORAGE storage_clause] ...
    [REFRESH [FAST | COMPLETE | FORCE] [START WITH date] [NEXT date]]
```

where:

schema
: is the schema containing the snapshot. If you omit schema, Oracle assumes the snapshot is in your own schema.

snapshot
: is the name of the snapshot to be altered.

PCTFREE
PCTUSED
INITRANS
MAXTRANS
: change the values of these parameters for the internal table that Oracle uses to maintain the snapshot's data. See the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters of the CREATE TABLE command.

STORAGE
: changes the storage characteristics of the internal table that Oracle uses to maintain the snapshot's data.

REFRESH
: changes the mode and times for automatic refreshes:

    FAST
    : specifies a fast refresh, or a refresh using the snapshot log associated with the master table.

    COMPLETE
    : specifies a complete refresh, or a refresh that reexecutes the snapshot's query.

    FORCE
    : specifies a fast refresh if one is possible or complete refresh if a fast refresh is not possible. Oracle decides whether a fast refresh is possible at refresh time.
    If you omit the FAST, COMPLETE, and FORCE options,

Oracle uses FORCE by default.

START WITH
specifies a date expression for the next automatic refresh time.

NEXT
specifies a new date expression for calculating the interval between automatic refreshes.

START WITH and NEXT values must evaluate to times in the future.

USING INDEX
alters the storage characteristics for the index on a simple snapshot.

PREREQUISITES:

The snapshot must be in your own schema or you must have ALTER ANY SNAPSHOT system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the snapshot's creation label or you must satisfy one of these criteria:

* If the snapshot's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the snapshot's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the snapshot's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

To change the storage characteristics of the internal table that Oracle uses to maintain the snapshot's data, you must also have the privileges to alter that table. For information on these privileges, see the ALTER TABLE command.

SEE:
CREATE SNAPSHOT, DROP SNAPSHOT

ALTER SNAPSHOT LOG command

PURPOSE:

Changes the storage characteristics of a snapshot log.

SYNTAX:

```
ALTER SNAPSHOT LOG ON [schema.]table
    [PCTFREE  integer]    [PCTUSED  integer]
    [INITRANS integer]    [MAXTRANS integer]
    [STORAGE storage_clause]
```

where:

schema
is the schema containing the snapshot log and its master table. If you omit schema, Oracle assumes the snapshot log is in your own schema.

table
is the name of the master table associated with the snapshot log to be altered.

PCTFREE
PCTUSED
INITRANS
MAXTRANS
change the values of these parameters for the snapshot log. See the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters of the CREATE TABLE command.

STORAGE
changes the storage characteristics of the snapshot log.

PREREQUISITES:

Since a snapshot log is simply a table, the privileges that authorize operations on it are the same as those for a table. To change its storage characteristics, you must have the privileges listed for the ALTER TABLE command.

SEE:
CREATE SNAPSHOT, CREATE SNAPSHOT LOG, DROP SNAPSHOT LOG

ALTER SYSTEM command

PURPOSE:

To dynamically alter your Oracle instance in one of these ways:

* to enable or disable resource limits
* to manage shared server processes or dispatcher processes for the multi-threaded server architecture
* to explicitly switch redo log file groups
* to explicitly perform a checkpoint
* to verify access to data files
* to restrict logons to Oracle to only those users with RESTRICTED SESSION system privilege
* to enable distributed recovery in a single-process environment
* to disable distributed recovery
* to manually archive redo log file groups or to enable or disable automatic archiving
* to clear all data from the shared pool in the System Global Area (SGA)
* to terminate a session

SYNTAX:

```
ALTER SYSTEM
    { {ENABLE | DISABLE} RESTRICTED SESSION
    | FLUSH SHARED_POOL
    | {CHECKPOINT | CHECK DATAFILES} [GLOBAL | LOCAL]
    | SET { RESOURCE_LIMIT           = { TRUE | FALSE }
          | GLOBAL_NAMES             = { TRUE | FALSE }
          | MTS_DISPATCHERS          = 'protocol, integer'
          | MTS_SERVERS              = integer
          | LICENSE_MAX_SESSIONS     = integer
          | LICENSE_SESSIONS_WARNING = integer
          | LICENSE_MAX_USERS        = integer
          | SESSION_CACHED_CURSORS   = integer } ...
    | SWITCH LOGFILE
    | {ENABLE | DISABLE} DISTRIBUTED RECOVERY
    | ARCHIVE LOG archive_log_clause
    | KILL SESSION 'integer1, integer2' }
```

where:

You can use these options regardless of whether your instance has the database dismounted or mounted, open or closed:

ENABLE RESTRICTED SESSION
: allows only users with RESTRICTED SESSION system privilege to logon to Oracle.

DISABLE RESTRICTED SESSION
: reverses the effect of the ENABLE RESTRICTED SESSION option, allowing all users with CREATE SESSION system privilege to logon to Oracle.

FLUSH SHARED_POOL
clears all data from the shared pool in the System Global Area (SGA).

You can use these options when your instance has the database mounted, open or closed:

CHECKPOINT
performs a checkpoint.
GLOBAL
performs a checkpoint for all instances that have opened the database.
LOCAL
performs a checkpoint only for the thread of redo log file groups for your instance. You can only use this option when your instance has the database open.

If you omit both the GLOBAL and LOCAL options, Oracle performs a global checkpoint.

CHECK DATAFILES
verifies access to online data files.
GLOBAL
verifies that all instances that have opened the database can access all online data files.
LOCAL
verifies that your instance can access all online data files.

If you omit both the GLOBAL and LOCAL options, Oracle uses GLOBAL by default.

You can only use these parameters and options when your instance has the database open:

RESOURCE_LIMIT
controls resource limits.
TRUE
enables resource limits.
FALSE
disables resource limits.

GLOBAL_NAMES
controls the enforcement of global naming:
TRUE
enables the enforcement of global names.
FALSE
disables the enforcement of global names.

MTS_SERVERS
specifies a new minimum number of shared server processes.

MTS_DISPATCHERS
specifies a new number of dispatcher processes:

protocol
is the network protocol of the dispatcher processes.
integer
is the new number of dispatcher processes of the specified protocol.

You can specify multiple MTS_DISPATCHERS parameters in a single command for multiple network protocols.

LICENSE_MAX_SESSIONS
limits the number of sessions on your instance. A value of 0 disables the limit.

LICENSE_SESSIONS_WARNING
establishes a threshold of sessions over which Oracle writes warning messages to the ALERT file for subsequent sessions. A value of 0 disables the warning threshold.

LICENSE_MAX_USERS
limits the number of users on your database. A value of 0 disables the limit.

SESSION_CACHED_CURSORS
specify a positive integer for the maximum number of session cursors kept in the cursor cache.

SWITCH LOGFILE
switches redo log file groups.

ENABLE DISTRIBUTED RECOVERY
enables distributed recovery. In a single-process environment, you must use this option to initiate distributed recovery.

DISABLE DISTRIBUTED RECOVERY
disables distributed recovery.

ARCHIVE LOG
manually archives redo log files or enables or disables automatic archiving.

KILL SESSION
terminates a session. You must identify the session with both of these values from the V$SESSION:
integer1
is the value of the SID column.
integer2
is the value of the SERIAL# column.

PREREQUISITES:

You must have ALTER SYSTEM system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must be the equivalent of DBHIGH.

SEE:

ALTER SESSION, CREATE PROFILE, CREATE USER

ALTER TABLE command

PURPOSE:

To alter the definition of a table in one of these ways:

* to add a column
* to add an integrity constraint
* to redefine a column (datatype, size, default value)
* to modify storage characteristics or other parameters
* to enable, disable, or drop an integrity constraint or trigger
* to explicitly allocate an extent

SYNTAX:

```
ALTER TABLE [schema.]table
    [ADD {    { column datatype [DEFAULT expr] [column_constraint] ...
              | table_constraint}
         |  ( { column datatype [DEFAULT expr] [column_constraint] ...
              | table_constraint}
           [, { column datatype [DEFAULT expr] [column_constraint] ...
              | table_constraint} ] ... ) } ]
    [MODIFY {   column [datatype] [DEFAULT expr] [column_constraint] ...
            |  (column [datatype] [DEFAULT expr] [column_constraint] ...
 [, column datatype [DEFAULT expr] [column_constraint] ...] ...) } ]
    [PCTFREE  integer] [PCTUSED  integer]
    [INITRANS integer] [MAXTRANS integer]
    [STORAGE storage_clause]
    [DROP drop_clause] ...
    [ALLOCATE EXTENT [( [SIZE integer [K|M] ]
                        [DATAFILE 'filename']
                        [INSTANCE integer] )]
    [  PARALLEL ( [ DEGREE { integer | DEFAULT } ]
                  [ INSTANCES { integer | DEFAULT } ]
                )
     | NOPARALLEL ]
    [  CACHE | NOCACHE  ]
    [ ENABLE   enable_clause
    | DISABLE disable_clause ] ...
```

where:

schema
: is the schema containing the table. If you omit schema, Oracle assumes the table is in your own schema.

table
: is the name of the table to be altered.

ADD
: adds a column or integrity constraint.

MODIFY
: modifies a the definition of an existing column. If you omit any of

the optional parts of the column definition (datatype, default value, or column constraint), these parts remain unchanged.

column
is the name of the column to be added or modified.

datatype
specifies a datatype for a new column or a new datatype for an existing column.

DEFAULT
specifies a default value for a new column or a new default for an existing column. Oracle assigns this value to the column if a subsequent INSERT statement omits a value for the column. The datatype of the default value must match the datatype specified for the column. A DEFAULT expression cannot contain references to other columns, the pseudocolumns CURRVAL, NEXTVAL, LEVEL, and ROWNUM, or date constants that are not fully specified.

column_constraint
adds or removes a NOT NULL constraint to or from and existing column.

table_constraint
adds an integrity constraint to the table.

PCTFREE
PCTUSED
INITRANS
MAXTRANS
changes the value of one of these parameters for the table. See the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters of the CREATE TABLE command.

STORAGE
changes the storage characteristics of the table.

DROP
drops an integrity constraint.

ALLOCATE EXTENT
explicitly allocates a new extent for the table.

SIZE
specifies the size of the extent in bytes. You can use K or M to specify the extent size in kilobytes or megabytes. If you omit this parameter, Oracle determines the size based on the values of the table's STORAGE parameters.

DATAFILE
specifies one of the data files in the table's tablespace to contain the new extent. If you omit this parameter, Oracle chooses the data file.

INSTANCE
makes the new extent available to the specified instance. An instance is identified by the value of

its initialization parameter INSTANCE_NUMBER. If you omit this parameter, the extent is available to all instances. Only use this parameter if you are using Oracle with the Parallel Server option in parallel mode.

Explicitly allocating an extent with this clause does affect the size for the next extent to be allocated as specified by the NEXT and PCTINCREASE storage parameters.

PARALLEL
DEGREE specifies the number of query server processes that can scan the table in parallel. Either specify a positive integer or DEFAULT which signifies to use the initialization parameter PARALLEL_DEFAULT_SCANSIZE to estimate the number of query servers to use.

INSTANCES specifies the minimum number of instances that need to be available before the table can be spread across all available instances of a Parallel Server. A positive integer specifies the number of instances. DEFAULT signifies that the parameter PARALLEL_MAX_PARTITIONSIZE is used to calculate whether a table is split across all instances' buffer caches.

NOPARALLEL
specifies that queries on this table are not performed in parallel by default. A hint in the query still causes the query to be performed in parallel.

CACHE
specifies that blocks of this table are placed on the most recently used end of the LRU list of the buffer cache when the a full table scan is performed.
This option is useful for small lookup tables.

NOCACHE
specifies that blocks of the table in the buffer cache follow the standard LRU algorithm when a full table scan is performed.

ENABLE
enables a single integrity constraint or all triggers associated with the table.

DISABLE
disables a single integrity constraint or all triggers associated with the table.

Integrity constraints specified in these clauses must be defined in the ALTER TABLE statement or in a previously issued statement. You can also enable and disable integrity constraints with the ENABLE and DISABLE keywords of the CONSTRAINT clause. If you define an integrity constraint but do not explicitly enable or disable it, Oracle enables it by default.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the table's creation label or you must satisfy one of these criteria:

* If the table's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the table's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the table's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

CONSTRAINT clause, CREATE TABLE, DISABLE clause, DROP clause, ENABLE clause, STORAGE clause

ALTER TABLESPACE command

PURPOSE:

To alter an existing tablespace in one of these ways:

* to add or rename data file(s)
* to change default storage parameters
* to take the tablespace online or offline
* to begin or end a backup

SYNTAX:

```
ALTER TABLESPACE tablespace
    { ADD DATAFILE filespec [, filespec] ...
    | RENAME DATAFILE 'filename' [,'filename'] ...
                  TO 'filename' [,'filename'] ...
    | DEFAULT STORAGE storage_clause
    | ONLINE
    | OFFLINE [NORMAL | TEMPORARY | IMMEDIATE]
    | READ ONLY
    | READ WRITE
    | {BEGIN | END} BACKUP}
```

where:

tablespace
is the name of the tablespace to be altered.

ADD DATAFILE
adds the data file specified by filespec to the tablespace. You can add a data file while the tablespace is online or offline. Be sure that the data file is not already in use by another database.

RENAME DATAFILE
renames one or more of the tablespace's data files. Take the tablespace offline before renaming the data file. Each 'filename' must fully specify a data file using the conventions for filenames on your operating system.

This clause only associates the tablespace with the new file rather than the old one. This clause does not actually change the name of the operating system file. You must change the name of the file through your operating system.

DEFAULT STORAGE
specifies the new default storage parameters for objects subsequently created in the tablespace.

ONLINE
brings the tablespace online.

OFFLINE
takes the tablespace offline and prevents further access to its

segments.

NORMAL
performs a checkpoint for all data files in the tablespace. All of these data files must be online. You need not perform media recovery on this tablespace before bringing it back online. You must use this option if the database is in noarchivelog mode.

TEMPORARY
performs a checkpoint for all online data files in the tablespace but does not ensure that all files can be written. Any offline files may require media recovery before you bring the tablespace back online.

IMMEDIATE
does not ensure that tablespace files are available and does not perform a checkpoint. You must perform media recovery on the tablespace before bringing it back online.

The default is NORMAL.

Before taking a tablespace offline for a long period of time, you may want to alter any users who have been assigned the tablespace as either a default or temporary tablespace. When the tablespace is offline, these users cannot allocate space for objects or sort areas in the tablespace. You can reassign users new default and temporary tablespaces with the ALTER USER command.

READ ONLY
specifies that write operations are not permitted on the tablespace.
Before using this option, the tablespace must meet these prerequisites:

* the tablespace must be online
* there must not be any active transactions in the entire database
* the tablespace must not contain any active rollback segments
* the tablespace must not be involved in an online backup
* the COMPATIBLE initialization parameter must be set to 7.1.0 or greater

READ WRITE
specifies that a read-only tablespace be made writeable. To issue this command, all of the datafiles in the tablespace must be online.

BEGIN BACKUP
signifies that an online backup is to be performed on the data files that comprise this tablespace. This option does not prevent users from accessing the tablespace. This option is used for control file and redo log record keeping. You must use this option before beginning an online backup.

While the backup is in progress, you cannot perform any of these operations:

* take the tablespace offline normally
* shutdown the instance
* begin another backup of the tablespace

END BACKUP

signifies that an online backup of the tablespace is complete. Use this option as soon as possible after completing an online backup.

PREREQUISITES:

If you have ALTER TABLESPACE system privilege, you can perform any of this command's operations. If you have MANAGE TABLESPACE system privilege, you can only perform these operations:

* to take the tablespace online or offline
* to begin or end a backup

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the tablespace's creation label or you must satisfy one of these criteria:

* If the tablespace's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the tablespace's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the tablespace's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

If you are using Trusted Oracle in DBMS MAC mode, to add a data file, your operating system process label must be the equivalent of DBHIGH.

SEE:

CREATE DATABASE, CREATE TABLESPACE, DROP TABLESPACE, STORAGE clause

ALTER TRIGGER command

PURPOSE:

To perform one of these operations on a database trigger:

* enable
* disable

SYNTAX:

```
ALTER TRIGGER [schema.]trigger
    { ENABLE
    | DISABLE }
```

where:

schema
is the schema containing the trigger. If you omit schema, Oracle assumes the trigger is in your own schema.

trigger
is the name of the trigger to be altered.

ENABLE
enables the trigger.

DISABLE
disables the trigger.

PREREQUISITES:

The trigger must be in your own schema or you must have ALTER ANY TRIGGER system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the trigger's creation label or you must satisfy one of these criteria:

* If the trigger's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the trigger's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the trigger's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE TRIGGER, DISABLE clause, DROP TRIGGER, ENABLE clause

ALTER USER command

PURPOSE:

To change any of these characteristics of a database user:

* password
* default tablespace for object creation
* tablespace for temporary segments created for the user
* tablespace access and tablespace quotas
* limits on database resources
* default roles

SYNTAX:

```
ALTER USER user
    [IDENTIFIED {BY password | EXTERNALLY}]
    [DEFAULT TABLESPACE tablespace]
    [TEMPORARY TABLESPACE tablespace]
    [QUOTA {integer [K|M] | UNLIMITED} ON tablespace] ...
    [PROFILE profile]
    [DEFAULT ROLE { role [, role] ...
                  | ALL [EXCEPT role [, role] ...]
                  | NONE}]
```

where:

user
: is the user to be altered.

IDENTIFIED
: indicates how Oracle permits user access.

  BY
  : specifies a new password for the user. The password does not appear in quotes and is not case-sensitive. The password can only contain single-byte characters from your database character set regardless of whether this character set also contains multi-byte characters.

  EXTERNALLY
  : indicates that Oracle verifies user access with the operating system, rather than with a password. See the CREATE USER command.

Although you do not need privileges to change your own password, you must have ALTER USER system privilege to change from BY password to EXTERNALLY or vice-versa.

DEFAULT TABLESPACE
: specifies the default tablespace for object creation.

TEMPORARY TABLESPACE
: specifies the tablespace for the creation of temporary segments for operations such as sorting that require more space than is available

in memory.

QUOTA
establishes a space quota of integer bytes on the tablespace for the user. This quota is the maximum space in tablespace that can be allocated for objects in the user's schema. You can use K or M to specify the quota in kilobytes or megabytes. You need not have quota on the tablespace to establish a quota on the tablespace for another user. See the CREATE USER command.

If you reduce an existing quota to a value below the space allocated for existing objects in the user's schema in the tablespace, no more space in the tablespace can be allocated to objects in the schema.

Note that an ALTER USER statement can contain multiple QUOTA clauses for multiple tablespaces.

UNLIMITED
places no limit on the space in the tablespace allocated to objects in the user's schema.

PROFILE
changes the user's profile to profile. In subsequent sessions, the user is subject to the limits defined in the new profile.

To assign the default limits to the user, assign the user the DEFAULT profile.

DEFAULT ROLE
establishes default roles for the user. Oracle enables the user's default roles at logon. By default, all roles granted to the user are default roles.

ALL
makes all the roles granted to the user default roles, except those listed in the EXCEPT clause.

NONE
makes none of the roles granted to the user default roles.

PREREQUISITES:

You must have ALTER USER privilege. However, you can change your own password without this privilege.

SEE:
CREATE PROFILE, CREATE ROLE, CREATE TABLESPACE, CREATE USER

ALTER VIEW command

PURPOSE:

To recompile a view.

SYNTAX:

```
ALTER VIEW [schema.]view
    COMPILE
```

where:

schema
is the schema containing the view. If you omit schema, Oracle assumes the view is in your own schema.

view
is the name of the view to be recompiled.

COMPILE
causes Oracle to recompile the view. The COMPILE keyword is required.

PREREQUISITES:

The view must be in your own schema or you must have ALTER ANY TABLE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the view's creation label or you must satisfy one of these criteria:

* If the view's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the view's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the view's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE VIEW

ANALYZE command

PURPOSE:

To perform one of these functions on an index, table, or cluster:

* to collect statistics about the object used by the optimizer and store them in the data dictionary
* to delete statistics about the object from the data dictionary
* to validate the structure of the object
* to identify migrated and chained rows of the table or cluster

SYNTAX:

```
ANALYZE
    { INDEX [schema.]index
            { { COMPUTE STATISTICS
              | ESTIMATE STATISTICS [SAMPLE integer {ROWS | PERCENT}]
              | DELETE STATISTICS }
            | VALIDATE STRUCTURE }
    | {TABLE [schema.]table | CLUSTER [schema.]cluster}
            { { COMPUTE
              | ESTIMATE [SAMPLE integer {ROWS | PERCENT}]
              | DELETE } STATISTICS
            | VALIDATE STRUCTURE [CASCADE]
            | LIST CHAINED ROWS [INTO [schema.]table] } }
```

where:

INDEX
identifies an index to be analyzed. If you omit schema, Oracle assumes the index is in your own schema.

TABLE
identifies a table to be analyzed. If you omit schema, Oracle assumes the table is in your own schema. When you collect statistics for a table, Oracle also automatically collects the statistics for each of the table's indexes.

CLUSTER
identifies a cluster to be analyzed. If you omit schema, Oracle assumes the cluster is in your own schema. When you collect statistics for a cluster, Oracle also automatically collects the statistics for all the cluster's tables and all their indexes, including the cluster index.

COMPUTE STATISTICS
computes exact statistics about the analyzed object and stores them in the data dictionary.

ESTIMATE STATISTICS
estimates statistics about the analyzed object and stores them in the data dictionary.

SAMPLE

specifies the amount of data from the analyzed object Oracle samples to estimate statistics. If you omit this parameter, Oracle samples 1064 rows. If you specify more than half of the data, Oracle reads all the data and computes the statistics.

ROWS
causes Oracle to sample integer rows of the table or cluster or integer entries from the index. The integer must be at least 1.

PERCENT
causes Oracle to sample integer percent of the rows from the table or cluster or integer percent of the index entries. The integer can range from 1 to 99.

DELETE STATISTICS
deletes any statistics about the analyzed object that are currently stored in the data dictionary.

VALIDATE STRUCTURE
validates the structure of the analyzed object. If you use this option when analyzing a cluster, Oracle automatically validates the structure of the cluster's tables.

CASCADE
validates the structure of the indexes associated with the table or cluster. If you use this option when validating a table, Oracle also validates the table's indexes. If you use this option when validating a cluster, Oracle also validates all the clustered tables' indexes, including the cluster index.

LIST CHAINED ROWS
identifies migrated and chained rows of the analyzed table or cluster. You cannot use this option when analyzing an index.

INTO
specifies a table into which Oracle lists the migrated and chained rows. If you omit schema, Oracle assumes the list table is in your own schema. If you omit this clause altogether, Oracle assumes that the table is named CHAINED_ROWS. The list table must be on your local database.

PREREQUISITES:

The object to be analyzed must be in your own schema or you must have the ANALYZE ANY system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the creation label of the object to be analyzed or you must satisfy one of these criteria:

* If the object's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the object's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the object's creation label and your DBMS label are

noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

If you want to list chained rows of a table or cluster into a list table, the list table must be in your own schema or you must have INSERT privilege on the list table or you must have INSERT ANY TABLE system privilege. If you are using Trusted Oracle in DBMS MAC mode, the list table must also meet the criteria for the analyzed object described above.

SEE:

Chapter 13, The Optimizer, in the Oracle7 Server Concepts Manual, and Chapter 22, Tuning I/O, in the Oracle7 Server Administrator's Guide.

ARCHIVE LOG clause

PURPOSE:

To manually archive redo log file groups or to enable or disable automatic archiving.

SYNTAX:

```
ARCHIVE LOG [THREAD integer]
    { { SEQ integer
      | CHANGE integer
      | CURRENT
      | GROUP integer
      | LOGFILE 'filename'
      | NEXT
      | ALL
      | START }
      [TO 'location']
    | STOP }
```

where:

THREAD
: specifies thread containing the redo log file group to be archived. You only need to specify this parameter if you are using Oracle with the Parallel Server option in parallel mode.

SEQ
: manually archives the online redo log file group identified by the log sequence number integer in the specified thread. If you omit the THREAD parameter, Oracle archives the specified group from the thread assigned to your instance.

CHANGE
: manually archives the online redo log file group containing the redo log entry with the system change number (SCN) specified by integer in the specified thread. If the SCN is in the current redo log file group, Oracle performs a log switch. If you omit the THREAD parameter, Oracle archives the groups containing this SCN from all enabled threads. You can only use this option when your instance has the database open.

CURRENT
: manually archives the current redo log file group of the specified thread, forcing a log switch. If you omit the THREAD parameter, Oracle archives the current redo log file groups from all enabled threads. You can only use this option when your instance has the database open.

GROUP
: manually archives the online redo log file group with the specified GROUP value. You can determine the GROUP value for a redo log file group by examining the data dictionary view DBA_LOG_FILES. If you

specify both the THREAD and GROUP parameters, the specified redo log file group must be in the specified thread.

LOGFILE
manually archives the online redo log file group containing the redo log file member identified by 'filename'. If you specify both the THREAD and LOGFILE parameters, the specified redo log file group must be in the specified thread.

NEXT
manually archives the next online redo log file group from the specified thread that is full but has not yet been archived. If you omit the THREAD parameter, Oracle archives the earliest unarchived redo log file group from any enabled thread.

ALL
manually archives all online redo log file groups from the specified thread that are full but have not been archived. If you omit the THREAD parameter, Oracle archives all full unarchived redo log file groups from all enabled threads.

START
enables automatic archiving of redo log file groups. You can only enable automatic archiving for the thread assigned to your instance.

TO
specifies the location to which the redo log file group is archived. The value of this parameter must be a fully-specified file location following the conventions of your operating system. If you omit this parameter, Oracle archives the redo log file group to the location specified by the initialization parameter LOG_ARCHIVE_DEST.

STOP
disables automatic archiving of redo log file groups. You can only disable automatic archiving for the thread assigned to your instance.

PREREQUISITES:

The ARCHIVE LOG clause must appear in an ALTER SYSTEM command. You must have the privileges necessary to issue this statement. For information on these privileges, see the ALTER SYSTEM command.

You must also have the OSDBA or OSOPER role enabled.

You can use most of the options of this clause when your instance has the database mounted, open or closed. Options that require your instance to have the database open are noted.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must be the equivalent of DBHIGH.

SEE:
ALTER SYSTEM

AUDIT command (SQL Statements)

PURPOSE:

To choose specific SQL statements for auditing in subsequent user sessions. To choose particular schema objects for auditing, use the AUDIT command (Schema Objects).

SYNTAX:

```
AUDIT {statement_opt | system_priv}
   [, {statement_opt | system_priv} ] ...
    [BY user [, user] ...]
    [BY {SESSION | ACCESS}]
    [WHENEVER [NOT] SUCCESSFUL]
```

where:

statement_opt
chooses specific SQL statements for auditing.

system_priv
chooses SQL statements that are authorized by the specified system privilege for auditing.

BY user
chooses only SQL statements issued by specified users for auditing. If you omit this clause, Oracle audits all users' statements.

BY SESSION
causes Oracle to write a single record for all SQL statements of the same type issued in the same session.

BY ACCESS
causes Oracle to write one record for each audited statement.

If you specify statement options or system privileges that audit Data Definition Language statements, Oracle automatically audits by access regardless of whether you specify the BY SESSION or BY ACCESS option.

For statement options and system privileges that audit other types of SQL statements, you can specify either the BY SESSION or BY ACCESS option. BY SESSION is the default.

WHENEVER SUCCESSFUL
chooses auditing only for SQL statements that complete successfully.
NOT
chooses auditing only for statements that fail, or result in errors.

If you omit the WHENEVER clause, Oracle audits SQL statements regardless of success or failure.

PREREQUISITES:

You must have AUDIT SYSTEM system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the creation label of the users whose SQL statements you are auditing.

SEE:

AUDIT (Schema Objects), NOAUDIT (SQL Statements)

AUDIT command (Schema Objects)

PURPOSE:

To choose a specific schema object for auditing. To choose particular SQL commands for auditing, use the AUDIT command (SQL Statements).

SYNTAX:

```
AUDIT object_opt [, object_opt] ...
    ON { [schema.]object | DEFAULT }
    [BY {SESSION | ACCESS}]
    [WHENEVER [NOT] SUCCESSFUL]
```

where:

object_opt
specifies a particular operation for auditing.

schema
is the schema containing the object chosen for auditing. If you omit schema, Oracle assumes the object is in your own schema.

object
identifies the object chosen for auditing. The object must be one of these types:

* table
* view
* sequence
* stored procedure, function, or package
* snapshot

You can also specify a synonym for a table, view, sequence, procedure, stored function, package, or snapshot.

DEFAULT
establishes the specified object options as default object options for subsequently created objects.

BY SESSION
means that Oracle writes a single record for all operations of the same type on the same object issued in the same session.

BY ACCESS
means that Oracle writes one record for each audited operation.

If you omit both of these options, Oracle audits by session.

WHENEVER SUCCESSFUL
chooses auditing only for SQL statements that complete successfully.

NOT
chooses auditing only for statements that fail, or result in errors.

If you omit the WHENEVER clause entirely, Oracle audits all SQL statements, regardless of success or failure.

PREREQUISITES:

The object you choose for auditing must be in your own schema or you must have AUDIT ANY system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the object's creation label or you must satisfy one of these criteria:

* If the object's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the object's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.

If the object's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

AUDIT (SQL Statements), NOAUDIT (Schema Objects)

COMMENT command

PURPOSE:

To add a comment about a table, view, snapshot, or column into the data dictionary.

SYNTAX:

```
COMMENT ON {  TABLE [schema.]{table | view | snapshot}
           | COLUMN [schema.]{table | view | snapshot}.column }
    IS 'text'
```

where:

TABLE
specifies the schema and name of the table, view, or snapshot to be commented.

COLUMN
specifies the name of the column of a table, view, or snapshot to be commented.

If you omit schema, Oracle assumes the table, view, or snapshot is in your own schema.

IS 'text'
is the text of the comment.

PREREQUISITES:

The table, view, or snapshot must be in your own schema or you must have COMMENT ANY TABLE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the creation label of the table, view, snapshot, or column.

SEE:
Comments

COMMIT command

PURPOSE:

To end your current transaction and make permanent all changes performed in the transaction. This command also erases all savepoints in the transaction and releases the transaction's locks. You can also use this command to manually commit an in-doubt distributed transaction.

SYNTAX:

```
COMMIT [WORK]
    [ COMMENT 'text'
    | FORCE 'text' [, integer] ]
```

where:

WORK
is supported only for compliance with standard SQL. The statements COMMIT and COMMIT WORK are equivalent.

COMMENT
specifies a comment to be associated with the current transaction. The 'text' is a quoted literal of up to 50 characters that Oracle stores in the data dictionary view DBA_2PC_PENDING along with the transaction ID if the transaction becomes in-doubt.

FORCE
manually commits an in-doubt distributed transaction. The transaction is identified by the 'text' containing its local or global transaction ID. To find the IDs of such transactions, query the data dictionary view DBA_2PC_PENDING. You can also use the integer to specifically assign the transaction a system change number (SCN). If you omit the integer, the transaction is committed using the current SCN.

COMMIT statements using the FORCE clause are not supported in PL/SQL.

PREREQUISITES:

You need no privileges to commit your current transaction.

To manually commit a distributed in-doubt transaction that you originally committed, you must have FORCE TRANSACTION system privilege. To manually commit a distributed in-doubt transaction that was originally committed by another user, you must have FORCE ANY TRANSACTION system privilege.

SEE:
ROLLBACK, SAVEPOINT, SET TRANSACTION

CONSTRAINT clause

PURPOSE:

To define an integrity constraint. An integrity constraint is a rule that restricts the values for one or more columns in a table.

SYNTAX:

Column constraint:

```
[CONSTRAINT constraint]
{ [NOT] NULL
| {UNIQUE | PRIMARY KEY}
|  REFERENCES [schema.]table [(column)]
       [ON DELETE CASCADE]
|  CHECK (condition) }
{ [ USING INDEX [PCTFREE integer]
                [INITRANS integer] [MAXTRANS integer]
                [TABLESPACE tablespace]
                [STORAGE storage_clause]
                [PARALLEL [integer] | NOPARALLEL] ]
  [ EXCEPTIONS INTO [schema.]table
| DISABLE }
```

Table constraint:

```
[CONSTRAINT constraint]
{ {UNIQUE | PRIMARY KEY} (column [,column] ...)
|  FOREIGN KEY (column [,column] ...)
        REFERENCES [schema.]table [(column [,column] ...)]
       [ON DELETE CASCADE]
| CHECK (condition) }
{ [ USING INDEX [PCTFREE integer]
                [INITRANS integer] [MAXTRANS integer]
                [TABLESPACE tablespace]
                [STORAGE storage_clause]
                [PARALLEL [integer] | NOPARALLEL] ]
  [ EXCEPTIONS INTO [schema.]table[@dblink]
| DISABLE }
```

where:

CONSTRAINT
identifies the integrity constraint by the name constraint. Oracle stores this name in the data dictionary along with the definition of the integrity constraint. If you omit this identifier, Oracle generates a name with this form:

SYS_Cn

where

n
is an integer that makes the name unique within the database.

For the names and definitions of integrity constraints, query the data dictionary.

NULL
specifies that a column can contain null values.

NOT NULL
specifies that a column cannot contain null values.

If you do not specify NULL or NOT NULL in a column definition, NULL is the default.

UNIQUE
designates a column or combination of columns as a unique key.

PRIMARY KEY
designates a column or combination of columns as the table's primary key.

FOREIGN KEY
designates a column or combination of columns as the foreign key in a referential integrity constraint.

REFERENCES
identifies the primary or unique key that is referenced by a foreign key in a referential integrity constraint.

ON DELETE CASCADE
specifies that Oracle maintains referential integrity by automatically removing dependent foreign key values if you remove a referenced primary or unique key value.

CHECK
specifies a condition that each row in the table must satisfy.

USING INDEX
specifies parameters for the index Oracle uses to enforce a UNIQUE or PRIMARY KEY constraint. The name of the index is the same as the name of the constraint. You can choose the values of the INITRANS, MAXTRANS, TABLESPACE, STORAGE, and PCTFREE parameters for the index. For information on these parameters, see the CREATE TABLE command.

Only use this clause when enabling UNIQUE and PRIMARY KEY constraints.

PARALLEL
specifies the number of processes that create the index in parallel. You can only specify positive integer values greater than 1. If you do not specify an integer, the degree of parallelism is based on the parallelism specified in the table's definition.

NOPARALLEL
specifies that the index should not be created in parallel.

EXCEPTIONS INTO

identifies a table into which Oracle places information about rows that violate an enabled integrity constraint. This table must exist before you use this option. If you omit schema, Oracle assumes the exception table is in your own schema. The exception table must be on your local database.

DISABLE

disables the integrity constraint. If an integrity constraint is disabled, Oracle does not enforce it.

If you do not specify this option, Oracle automatically enables the integrity constraint.

You can also enable and disable integrity constraints with the ENABLE and DISABLE clauses of the CREATE TABLE and ALTER TABLE commands.

PREREQUISITES:

CONSTRAINT clauses can appear in either CREATE TABLE or ALTER TABLE commands. To define an integrity constraint, you must have the privileges necessary to issue one of these commands. See the CREATE TABLE and ALTER TABLE commands.

Defining a constraint may also require additional privileges or preconditions that depend on the type of constraint.

SEE:

ALTER TABLE, CREATE TABLE, DISABLE clause, ENABLE clause

CREATE CLUSTER command

PURPOSE:

To create a cluster. A cluster is a schema object that contains one or more tables that all have one or more columns in common.

SYNTAX:

```
CREATE CLUSTER [schema.]cluster
    (column datatype [,column datatype] ... )
    [PCTUSED integer] [PCTFREE integer]
    [SIZE integer [K|M] ]
    [INITRANS integer] [MAXTRANS integer]
    [TABLESPACE tablespace]
    [STORAGE storage_clause]
    [  PARALLEL ( [ DEGREE { integer | DEFAULT } ]
                  [ INSTANCES { integer | DEFAULT } ]
                )
     | NOPARALLEL ]
    [  CACHE | NOCACHE  ]
    [INDEX
    | [HASH IS column] HASHKEYS integer]
```

where:

schema
: is the schema to contain the cluster. If you omit schema, Oracle creates the cluster in your current schema.

cluster
: is the name of the cluster to be created.

column
: is the name of a column of the cluster key.

datatype
: is the datatype of a cluster key column. A cluster key column can have any internal datatype except LONG or LONG RAW.

PCTUSED
: specifies the limit that Oracle uses to determine when additional rows can be added to a cluster's data block. The value of this parameter is expressed as a whole number and interpreted as a percentage.

PCTFREE
: specifies the space reserved in each of the cluster's data blocks for future expansion. The value of the parameter is expressed as a whole number and interpreted as a percentage.

INITRANS
: specifies the initial number of concurrent update transactions allocated for data blocks of the cluster. The value of this

parameter for a cluster cannot be less than 2 or more than the value of the MAXTRANS parameter. The default value is the greater of the INITRANS value for the cluster's tablespace and 2.

MAXTRANS
specifies the maximum number of concurrent update transactions for any given data block belonging to the cluster. The value of this parameter cannot be less than the value of the INITRANS parameter. The maximum value of this parameter is 255. The default value is the MAXTRANS value for the tablespace to contain the cluster.

For a complete description of the PCTUSED, PCTFREE, INITRANS, and MAXTRANS parameters, see the CREATE TABLE.

SIZE
specifies the amount of space in bytes to store all rows with the same cluster key value or the same hash value. You can use K or M to specify this space in kilobytes or megabytes. The value of this parameter cannot exceed the size of a data block. If you omit this parameter, Oracle reserves one data block for each cluster key value or hash value.

TABLESPACE
specifies the tablespace in which the cluster is created.

STORAGE
specifies how data blocks are allocated to the cluster.

PARALLEL
DEGREE specifies the number of query server processes that can scan the cluster in parallel. Either specify a positive integer or DEFAULT which signifies to use the initialization parameter PARALLEL_DEFAULT_SCANSIZE to estimate the number of query servers to use.

INSTANCES specifies the minimum number of instances that need to be available before the cluster can be spread across all available instances of a Parallel Server. A positive integer specifies the number of instances. DEFAULT signifies that the parameter PARALLEL_MAX_PARTITIONSIZE is used to calculate whether a table is split across all instances' buffer caches.

NOPARALLEL
specifies that queries on this cluster are not performed in parallel by default. A hint in the query still causes the query to be performed in parallel.

CACHE
specifies that blocks of this cluster are placed on the most recently used end of the LRU list of the buffer cache when the a full table scan is performed.
This option is useful for small lookup tables.

NOCACHE
specifies that blocks of the cluster in the buffer cache follow the standard LRU algorithm when a full table scan is performed.

PARALLEL
specifies the number of processes that can scan the tables in the in parallel.
You can only specify positive integer values greater than 1. If you do not specify an integer, the degree of parallelism is based on an estimate of the size of the table and the value of the PARALLEL_DEFAULT_SCANSIZE initialization parameter.

NOPARALLEL
specifies that queries on this cluster are not performed in parallel by default. A hint in the query still causes the query to be performed in parallel.

CACHE
specifies that the entire table is to be placed in the buffer cache. This option is useful for small lookup tables.

NOCACHE
specifies that blocks of the table in the buffer cache follow the standard LRU algorithm.

CACHE PARTITIONS
specifies the cluster is to be partitioned and cached on all instances of a parallel server available for parallel query processing, if at least the specified number of instances is available. If
PARALLEL
DEGREE specifies the number of query server processes that can scan the cluster in parallel. Either specify a positive integer or DEFAULT which signifies to use the initialization parameter PARALLEL_DEFAULT_SCANSIZE to estimate the number of query servers to use.

INSTANCES specifies the minimum number of instances that need to be available before the cluster can be spread across all available instances of a Parallel Server. A positive integer specifies the number of instances. DEFAULT signifies that the parameter PARALLEL_MAX_PARTITIONSIZE is used to calculate whether a table is split across all instances' buffer caches.

NOPARALLEL
specifies that queries on this cluster are not performed in parallel by default. A hint in the query still causes the query to be performed in parallel.

CACHE
specifies that blocks of this cluster are placed on the most recently used end of the LRU list of the buffer cache when the a full table scan is performed.
This option is useful for small lookup tables.

NOCACHE
specifies that blocks of the cluster in the buffer cache follow the standard LRU algorithm when a full table scan is performed.

INDEX

creates an indexed cluster. In an indexed cluster, rows are stored together based on their cluster key values

.

HASH IS

specifies a column to be used as the hash function for a hash cluster. In a hash cluster, rows are stored together based on their hash values. The hash function specifies the hash value for each row in the cluster. The value that you specify must be the only column of the cluster key and have a datatype of NUMBER with a scale of 0. Each value in the column must be a non-negative integer.

If you omit this parameter, Oracle uses an internal hash function for the hash cluster. The cluster key of a hash column can have one or more columns of any datatype. Hash clusters with composite cluster keys or cluster keys made up of non-integer columns must use the internal hash function.

HASHKEYS

creates a hash cluster and specifies the number of hash values for a hash cluster. Oracle rounds the HASHKEYS value up to the nearest prime number to obtain the actual number of hash values. The minimum value for this parameter is 2.

If you omit both the INDEX option and the HASHKEYS parameter, Oracle creates an indexed cluster by default.

PREREQUISITES:

To create a cluster in your own schema, you must have CREATE CLUSTER system privilege. To create a cluster in another user's schema, you must have CREATE ANY CLUSTER system privilege. Also, the owner of the schema to contain the cluster must have either space quota on the tablespace containing the cluster or UNLIMITED TABLESPACE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the label of the tablespace to contain the cluster. To create a cluster in another user's schema, your DBMS label must dominate the creation label of the owner of the schema.

SEE:

CREATE INDEX, CREATE TABLE, STORAGE clause

CREATE CONTROLFILE command

PURPOSE:

To recreate a control file in one of these cases:

* All copies of your existing control files have been lost through media failure.
* You want to change the name of the database.
* You want to change the maximum number of redo log file groups, redo log file members, archived redo log files, data files, or instances that can concurrently have the database mounted and open.

Warning: Oracle Corporation recommends that you perform a full backup of all files in the database before using this command.

SYNTAX:

```
CREATE CONTROLFILE [REUSE]
    [SET] DATABASE database
    LOGFILE [GROUP integer] filespec [, [GROUP integer] filespec] ...
    {RESETLOGS | NORESETLOGS}
    DATAFILE filespec [, filespec] ...
    [MAXLOGFILES integer]
    [MAXLOGMEMBERS integer]
    [MAXLOGHISTORY integer]
    [MAXDATAFILES integer]
    [MAXINSTANCES integer]
    [ARCHIVELOG | NOARCHIVELOG]
```

where:

REUSE
specifies that existing control files identified by the initialization parameter CONTROL_FILES can be reused, thus ignoring and overwriting any and all information they may currently contain. If you omit this option and any of these control files already exist, Oracle returns an error.

SET DATABASE
changes the name of the database. The name of a database can be as long as eight bytes.

DATABASE
specifies the name of the database. The value of this parameter must be the existing database name established by the previous CREATE DATABASE statement or CREATE CONTROLFILE statement.

LOGFILE
specifies the redo log file groups for your database. You must list all members of all redo log file groups. These files must all exist.

RESETLOGS
ignores the contents of the files listed in the LOGFILE clause. Each filespec in the LOGFILE clause must specify the SIZE parameter. Oracle assigns all redo log file groups to thread 1 and enables this thread for public use by any instance. After using this option, you must open the database using the RESETLOGS option of the ALTER DATABASE command.

NORESETLOGS
specifies that all files in the LOGFILE clause should be used as they were when the database was last open. These files must be the current redo log files rather than restored backups. Oracle reassigns the redo log file groups to the threads to which they were previously assigned and reenables the threads as they were previously enabled. If you specify GROUP values, Oracle verifies these values with the GROUP values when the database was last open.

DATAFILE
specifies the data files of the database. You must list all data files. These files must all exist, although they may be restored backups that require media recovery.

MAXLOGFILES
specifies the maximum number of redo log file groups that can ever be created for the database. Oracle uses this value to determine how much space in the control file to allocate for the names of redo log files. The default and maximum values depend on your operating system. The value that you specify should not be less than the greatest GROUP value for any redo log file group.

Note that the number of redo log file groups accessible to your instance is also limited by the initialization parameter LOG_FILES.

MAXLOGMEMBERS
specifies the maximum number of members, or copies, for a redo log file group. Oracle uses this value to determine how much space in the control file to allocate for the names of redo log files. The minimum value is 1. The maximum and default values depend on your operating system.

MAXLOGHISTORY
specifies the maximum number of archived redo log file groups for automatic media recovery of the Oracle Parallel Server. Oracle uses this value to determine how much space in the control file to allocate for the names of archived redo log files. The minimum value is 0. The default value is a multiple of the MAXINSTANCES value and varies depending on your operating system. The maximum value is limited only by the maximum size of the control file. Note that this parameter is only useful if you are using Oracle with the Parallel Server option in both parallel mode and archivelog mode.

MAXDATAFILES
specifies the maximum number of data files that can ever be created for the database. The minimum value is 1. The maximum and default values depend on your operating system. The value you specify

should not be less than the total number of data files ever in the database, including those for tablespaces that have been dropped.

Note that the number of data files accessible to your instance is also limited by the initialization parameter DB_FILES.

MAXINSTANCES
specifies the maximum number of instances that can simultaneously have the database mounted and open. This value takes precedence over the value of the initialization parameter INSTANCES. The minimum value is 1. The maximum and default values depend on your operating system.

ARCHIVELOG
establishes the mode of archiving the contents of redo log files before reusing them. This option prepares for the possibility of media recovery as well as instance recovery.

NOARCHIVELOG
establishes the initial mode of reusing redo log files without archiving their contents. This option prepares for the possibility of instance recovery but not media recovery.

If you omit both the ARCHIVELOG and NOARCHIVELOG options, Oracle chooses noarchivelog mode by default. After creating the control file, you can change between archivelog mode and noarchivelog mode with the ALTER DATABASE command.

PREREQUISITES:

You must have the OSDBA role enabled. The database must not be mounted by any instance.

If you are using Trusted Oracle in DBMS MAC mode, your operating system label must be the equivalent of DBHIGH.

SEE:
CREATE DATABASE

CREATE DATABASE command

PURPOSE:

To create a database, making it available for general use, with these options:

* to establish a maximum number of instances, data files, redo log files groups, or redo log file members
* to specify names and sizes of data files and redo log files
* to choose a mode of use for the redo log

Warning: This command prepares a database for initial use and erases any data currently in the specified files. Only use this command when you understand its ramifications.

SYNTAX:

```
CREATE DATABASE [database]
    [CONTROLFILE REUSE]
    [LOGFILE [GROUP integer] filespec [, [GROUP integer] filespec] ...]
    [MAXLOGFILES integer ]
    [MAXLOGMEMBERS integer]
    [MAXLOGHISTORY integer]
    [DATAFILE filespec [, filespec] ...]
    [MAXDATAFILES integer]
    [MAXINSTANCES integer]
    [ARCHIVELOG | NOARCHIVELOG]
    [EXCLUSIVE]
    [CHARACTER SET charset]
```

where:

database
is the name of the database to be created and can be up to eight bytes long. Oracle writes this name into the control file. If you subsequently issue an ALTER DATABASE statement and that explicitly specifies a database name, Oracle verifies that name with the name in the control file.

The database name may only contain the alphabetic characters:
* alphabetic characters (A...Z)
* numbers (0...9)
* an underscore (_)
* a dollar sign ($)
* a pound sign (#)

The database cannot be a SQL*DBA reserved word. If you omit database from a CREATE DATABASE statement, Oracle uses the name specified by the initialization parameter DB_NAME.

CONTROLFILE REUSE
reuses existing control files identified by the initialization parameter CONTROL_FILES, thus ignoring and overwriting any

information they currently contain. This option is usually used only when you are recreating a database, rather than creating one for the first time. You cannot use this option if you also specify a parameter value that requires that the control file be larger than the existing files. These parameters are MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES, and MAXINSTANCES.

If you omit this option and any of the files specified by CONTROL_FILES already exist, Oracle returns an error.

LOGFILE

specifies one or more files to be used as redo log files. Each filespec specifies a redo log file group containing one or more redo log file members, or copies. All redo log files specified in a CREATE DATABASE statement are added to redo log thread number 1.

You can also choose the value of the GROUP parameter for the redo log file group. Each value uniquely identifies a redo log file group and can range from 1 to the value of the MAXLOGFILES parameter. You cannot specify multiple redo log file groups having the same GROUP value. If you omit this parameter, Oracle generates its value automatically. You can examine the GROUP value for a redo log file group through the dynamic performance table V$LOG.

If you omit the LOGFILE clause, Oracle creates two redo log file groups by default. The names and sizes of the default files vary depending on your operating system.

MAXLOGFILES

specifies the maximum number of redo log file groups that can ever be created for the database. Oracle uses this value to determine how much space in the control file to allocate for the names of redo log files. The default, minimum, and maximum values vary depending on your operating system.

The number of redo log file groups accessible to your instance is also limited by the initialization parameter LOG_FILES.

MAXLOGMEMBERS

specifies the maximum number of members, or copies, for a redo log file group. Oracle uses this value to determine how much space in the control file to allocate for the names of redo log files. The minimum value is 1. The maximum and default values vary depending on your operating system.

MAXLOGHISTORY

specifies the maximum number of archived redo log files for automatic media recovery of Oracle with the Parallel Server option. Oracle uses this value to determine how much space in the control file to allocate for the names of archived redo log files. The minimum value is 0. The default value is a multiple of the MAXINSTANCES value and varies depending on your operating system. The maximum value is limited only by the maximum size of the control file. Note that this parameter is only useful if you are using the Oracle with the Parallel Server option in parallel mode and

archivelog mode.

DATAFILE
specifies one or more files to be used as data files. These files all become part of the SYSTEM tablespace. If you omit this clause, Oracle creates one data file by default. The name and size of this default file depends on your operating system.

MAXDATAFILES
specifies the maximum number of data files that can ever be created for the database. The minimum value is 1. The maximum and default values depend on your operating system.

The number of data files accessible to your instance is also limited by the initialization parameter DB_FILES

MAXINSTANCES
specifies the maximum number of instances that can simultaneously have this database mounted and open. This value takes precedence over the value of the initialization parameter INSTANCES. The minimum value is 1. The maximum and default values depend on your operating system.

ARCHIVELOG
establishes archivelog mode for redo log file groups. In this mode, the contents of a redo log file group must be archived before the group can be reused. This option prepares for the possibility of media recovery.

NOARCHIVELOG
establishes noarchivelog mode for redo log files groups. In this mode, the contents of a redo log file group need not be archived before the group can be reused. This option does not prepares for the possibility of media recovery.

The default is noarchivelog mode. After creating the database, you can change between archivelog mode and noarchivelog mode with the ALTER DATABASE command.

EXCLUSIVE
mounts the database in exclusive mode after it is created. This mode allows only your instance to access the database. Oracle automatically mounts the database in exclusive mode after creating it, so this keyword is entirely optional.

For multiple instances to access the database, you must first create the database, close and dismount the database, and then mount it in parallel mode. For information on closing, dismounting, and mounting the database, see the ALTER DATABASE command.

CHARACTER SET
specifies the character set the database uses to store data. You cannot change the database character set after creating the database. The supported character sets and default value of this parameter depends on your operating system.

PREREQUISITES:

You must have the OSDBA role enabled.

If you are using Trusted Oracle and you plan to use the database in DBMS MAC mode, your operating system label should be the equivalent of DBLOW.

SEE:

ALTER DATABASE, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE

CREATE DATABASE LINK command

PURPOSE:

To create a database link. A database link is an object in the local database that allows you to access objects on a remote database or to mount a secondary database in read-only mode. The remote database can be either an Oracle or a non-Oracle database.

SYNTAX:

```
CREATE [PUBLIC] DATABASE LINK dblink
    [CONNECT TO user IDENTIFIED BY password]
    [USING 'connect_string']
```

where:

PUBLIC
creates a public database link available to all users. If you omit this option, the database link is private and is available only to you.

dblink
is the complete or partial name of the database link. For guidelines for naming database links, see the section Referring to Objects in Remote Databases.

CONNECT TO user
IDENTIFIED BY password
is the username and password used to connect to the remote database. If you omit this clause, the database link uses the username and password of each user who uses the database link.

USING
specifies either:

* the database specification of a remote database
* the specification of a secondary database for a read-only mount.

For information on specifying remote databases, see the SQL*Net User's Guide for your specific SQL*Net protocol.

Read-only mounts are only available in Trusted Oracle and can only be specified for public database links.

PREREQUISITES:

To create a private database link, you must have CREATE DATABASE LINK system privilege. To create a public database link, you must have CREATE PUBLIC DATABASE LINK system privilege. Also, you must have CREATE SESSION privilege on a remote database. SQL*Net must be installed on both the local and remote databases.

SEE:

CREATE SYNONYM, SELECT

CREATE FUNCTION command

PURPOSE:

To create a stand-alone stored function. A stored function is a set of PL/SQL statements you can call by name. Stored functions are very similar to procedures, except that a function returns a value to the environment in which it is called.

SYNTAX:

```
CREATE [OR REPLACE] FUNCTION [schema.]function
    [ (argument [IN] datatype
    [, argument [IN] datatype] ...)]
    RETURN datatype
    {IS | AS} pl/sql_subprogram_body
```

where:

OR REPLACE
recreates the function if it already exists. You can use this option to change the definition of an existing function without dropping, recreating, and regranting object privileges previously granted on the function. If you redefine a function, Oracle recompiles it. For information on recompiling functions, see the ALTER FUNCTION command.

Users who had previously been granted privileges on a redefined function can still access the function without being regranted the privileges.

schema
is the schema to contain the function. If you omit schema, Oracle creates the function in your current schema.

function
is the name of the function to be created.

argument
is the name of an argument to the function. If the function does not accept arguments, you can omit the parentheses following the function name.

IN
specifies that you must supply a value for the argument when calling the function. This is always true for function arguments, so this keyword is entirely optional.

A procedure, rather than a stored function, can accept arguments for which the procedure passes a value back to the calling environment after execution.

datatype
is the datatype of an argument. An argument can have any datatype supported by PL/SQL.

The datatype cannot specify a length, precision, or scale. Oracle derives the length, precision, or scale of an argument from the environment from which the function is called.

RETURN datatype
specifies the datatype of the function's return value. Because every function must return a value, this clause is required. The return value can have any datatype supported by PL/SQL.

The datatype cannot specify a length, precision, or scale. Oracle derives the length, precision, or scale of the return value from the environment from which the function is called.

pl/sql_subprogram_body
is the definition of the function. Function definitions are written in PL/SQL.

To embed a CREATE FUNCTION statement inside an Oracle Precompiler program, you must terminate the statement with the keyword END-EXEC followed by the embedded SQL statement terminator for the specific language.

PREREQUISITES:

Before a stored function can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

To create a function in your own schema, you must have CREATE PROCEDURE system privilege. To create a function in another user's schema, you must have CREATE ANY PROCEDURE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, you can create a function in another user's schema if your DBMS label dominates the creation label of the other user.

SEE:
ALTER FUNCTION, CREATE PACKAGE, CREATE PACKAGE BODY, CREATE PROCEDURE, DROP FUNCTION

CREATE INDEX command

PURPOSE:

To create an index on one or more columns of a table or a cluster. An index is a database object that contains an entry for each value that appears in the indexed column(s) of the table or cluster and provides direct, fast access to rows.

SYNTAX:

```
CREATE INDEX [schema.]index
    ON { [schema.]table (column [ASC|DESC][, column [ASC|DESC]] ...)
       | CLUSTER [schema.]cluster }
    [INITRANS integer] [MAXTRANS integer]
    [TABLESPACE tablespace]
    [STORAGE storage_clause]
    [PARALLEL ( [DEGREE { integer | DEFAULT }]
                [INSTANCES {integer | DEFAULT }]
              )
    | NOPARALLEL ]
    [PCTFREE integer]
    [NOSORT]
```

where:

schema
is the schema to contain the index. If you omit schema, Oracle creates the index in your own schema.

index
is the name of the index to be created.

table
is the name of the table for which the index is to be created. If you do not qualify table with schema, Oracle assumes the table is contained in your own schema.

column
is the name of a column in the table. An index can have as many as 16 columns. A column of an index cannot be of datatype LONG or LONG RAW.

ASC
DESC
are allowed for DB2 syntax compatibility, although indexes are always created in ascending order. Indexes on character data are created in ascending order of the character values in the database character set.

CLUSTER
specifies the cluster for which a cluster index is to be created. If you do not qualify cluster with schema, Oracle assumes the cluster is contained in your current schema. You cannot create a

cluster index for a hash cluster.

INITRANS
MAXTRANS
establishes values for these parameters for the index. See the INITRANS and MAXTRANS parameters of the CREATE TABLE command.

TABLESPACE
is the name of the tablespace to hold the index. If you omit this option, Oracle creates the index in the default tablespace of the owner of the schema containing the index.

STORAGE
establishes the storage characteristics for the index.

PARALLEL
DEGREE specifies the number of processes that create an index in parallel. DEFAULT specifies the degree of parallelism is based on the parallelism specified in the table's definition.

NOPARALLEL
specifies that the index should not be created in parallel.

PCTFREE
is the percentage of space to leave free for updates and insertions within each of the index's data blocks.

NOSORT
indicates to Oracle that the rows are stored in the database in ascending order and therefore Oracle does not have to sort the rows when creating the index.

PREREQUISITES:

To create an index in your own schema, one of these conditions must be true:

* The table or cluster to be indexed must be in your own schema.
* You must have INDEX privilege on the table to be indexed.
* You must have CREATE ANY INDEX system privilege.

To create an index in another schema, you must have CREATE ANY INDEX system privilege.

Also, the owner of the schema to contain the index must have either space quota on the tablespace to contain the index or UNLIMITED TABLESPACE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the tablespace's label and match the table's label. If the table was created at DBHIGH or DBLOW, you must explicitly set your label to DBHIGH or DBLOW. You can create an index in another user's schema if your DBMS label dominates the creation label of the other user.

SEE:

ALTER INDEX, CONSTRAINT clause, DROP INDEX, STORAGE clause

CREATE PACKAGE command

PURPOSE:

To create the specification for a stored package. A package is an encapsulated collection of related procedures, functions, and other program objects stored together in the database. The specification declares these objects.

SYNTAX:

```
CREATE [OR REPLACE] PACKAGE [schema.]package
    {IS | AS} pl/sql_package_spec
```

where:

OR REPLACE
recreates the package specification if it already exists. You can use this option to change the specification of an existing package without dropping, recreating, and regranting object privileges previously granted on the package. If you change a package specification, Oracle recompiles it. For information on recompiling package specifications, see the ALTER PROCEDURE command.

Users who had previously been granted privileges on a redefined package can still access the package without being regranted the privileges.

schema
is the schema to contain the package. If you omit schema, Oracle creates the package in your own schema.

package
is the name of the package to be created.

pl/sql_package_spec
is the package specification. The package specification can declare program objects. Package specifications are written in PL/SQL.

To embed a CREATE PACKAGE statement inside an Oracle Precompiler program, you must terminate the statement with the keyword END-EXEC followed by the embedded SQL statement terminator for the specific language.

PREREQUISITES:

Before a package can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

To create a package in your own schema, you must have CREATE PROCEDURE system privilege. To create a package in another user's schema, you must have CREATE ANY PROCEDURE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, you can only

create a package in another user's schema if your DBMS label dominates the creation label of the other user.

To create a package, you must be using Oracle with the procedural option.

SEE:

ALTER PACKAGE, CREATE FUNCTION, CREATE PACKAGE BODY, CREATE PROCEDURE, DROP PACKAGE

CREATE PACKAGE BODY command

PURPOSE:

To create the body of a stored package. A package is an encapsulated collection of related procedures, stored functions, and other program objects stored together in the database. The body defines these objects.

SYNTAX:

```
CREATE [OR REPLACE] PACKAGE BODY [schema.]package
    {IS | AS} pl/sql_package_body
```

where:

OR REPLACE
recreates the package body if it already exists. You can use this option to change the body of an existing package without dropping, recreating, and regranting object privileges previously granted on it. If you change a package body, Oracle recompiles it. For information on recompiling package bodies, see the ALTER PACKAGE command.

Users who had previously been granted privileges on a redefined package can still access the package without being regranted the privileges.

schema
is the schema to contain the package. If you omit schema, Oracle creates the package in your current schema.

package
is the name of the package to be created.

pl/sql_package_body
is the package body. The package body can declare and define program objects. Package bodies are written in PL/SQL.

To embed a CREATE PACKAGE BODY statement inside an Oracle Precompiler program, you must terminate the statement with the keyword END-EXEC followed by the embedded SQL statement terminator for the specific language.

PREREQUISITES:

Before a package can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

To create a package in your own schema, you must have CREATE PROCEDURE system privilege. To create a package in another user's schema, you must have CREATE ANY PROCEDURE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, you can only

create a package in another user's schema if your DBMS label dominates the creation label of the other user.

To create a package, you must be using Oracle with the procedural option.

SEE:

ALTER PACKAGE, CREATE FUNCTION, CREATE PACKAGE, CREATE PROCEDURE, DROP PACKAGE

CREATE PROCEDURE command

PURPOSE:

To create a stand-alone stored procedure. A procedure is a group of PL/SQL statements that you can call by name.

SYNTAX:

```
CREATE [OR REPLACE] PROCEDURE [schema.]procedure
    [ (argument [IN | OUT | IN OUT] datatype
    [, argument [IN | OUT | IN OUT] datatype] ...)]
    {IS | AS} pl/sql_subprogram_body
```

where:

OR REPLACE
: recreates the procedure if it already exists. You can use this option to change the definition of an existing procedure without dropping, recreating, and regranting object privileges previously granted on it. If you redefine a procedure, Oracle recompiles it. For information on recompiling procedures, see the ALTER PROCEDURE command.

  Users who had previously been granted privileges on a redefined procedure can still access the procedure without being regranted the privileges.

schema
: is the schema to contain the procedure. If you omit schema, Oracle creates the procedure in your current schema.

procedure
: is the name of the procedure to be created.

argument
: is the name of an argument to the procedure. If the procedure does not accept arguments, you can omit the parentheses following the procedure name.

IN
: specifies that you must specify a value for the argument when calling the procedure.

OUT
: specifies that the procedure passes a value for this argument back to its calling environment after execution.

IN OUT
: specifies that you must specify a value for the argument when calling the procedure and that the procedure passes a value back to its calling environment after execution.

  If you omit IN, OUT, and IN OUT, the argument defaults to IN.

datatype
is the datatype of an argument. An argument can have any datatype supported by PL/SQL.

The datatype cannot specify a length, precision, or scale. Oracle derives the length, precision, or scale of an argument from the environment from which the procedure is called.

pl/sql_subprogram_body
is the definition of the procedure. Procedure definitions are written in PL/SQL.

To embed a CREATE PROCEDURE statement inside an Oracle Precompiler program, you must terminate the statement with the keyword END-EXEC followed by the embedded SQL statement terminator for the specific language.

PREREQUISITES:

Before a procedure can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

To create a procedure in your own schema, you must have CREATE PROCEDURE system privilege. To create a procedure in another schema, you must have CREATE ANY PROCEDURE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, you can only create a procedure in another user's schema if your DBMS label dominates the creation label of the other user.

To create a procedure, you must be using Oracle with the procedural option.

SEE:
ALTER PROCEDURE, CREATE FUNCTION, CREATE PACKAGE, CREATE PACKAGE BODY, DROP PROCEDURE

CREATE PROFILE command

PURPOSE:

To create a profile. A profile is a set of limits on database resources. If you assign the profile to a user, that user cannot exceed these limits.

SYNTAX:

```
CREATE PROFILE profile
    LIMIT   [SESSIONS_PER_USER          {integer | UNLIMITED | DEFAULT}]
            [CPU_PER_SESSION            {integer | UNLIMITED | DEFAULT}]
            [CPU_PER_CALL               {integer | UNLIMITED | DEFAULT}]
            [CONNECT_TIME               {integer | UNLIMITED | DEFAULT}]
            [IDLE_TIME                  {integer | UNLIMITED | DEFAULT}]
            [LOGICAL_READS_PER_SESSION  {integer | UNLIMITED | DEFAULT}]
            [LOGICAL_READS_PER_CALL     {integer | UNLIMITED | DEFAULT}]
            [COMPOSITE_LIMIT            {integer | UNLIMITED | DEFAULT}]
            [PRIVATE_SGA          {integer [K|M] | UNLIMITED | DEFAULT}]
```

where:

profile
: is the name of the profile to be created.

SESSIONS_PER_USER
: limits a user to integer concurrent sessions.

CPU_PER_SESSION
: limits the CPU time for a session. This value is expressed in hundredths of seconds.

CPU_PER_CALL
: limits the CPU time for a call (a parse, execute, or fetch). This value is expressed in hundredths of seconds.

CONNECT_TIME
: limits the total elapsed time of a session. This value is expressed in minutes.

IDLE_TIME
: limits periods of continuous inactive time during a session. This value is expressed in minutes. Long-running queries and other operations are not subject to this limit.

LOGICAL_READS_PER_SESSION
: limits the number of data blocks read in a session, including blocks read from memory and disk, to integer blocks.

LOGICAL_READS_PER_CALL
: limits the number of data blocks read for a call to process a SQL statement (a parse, execute, or fetch) to integer blocks.

PRIVATE_SGA
limits the amount of private space a session can allocate in the shared pool of the System Global Area (SGA) to integer bytes. You can also use the K or M to specify this limit in kilobytes or megabytes. This limit only applies if you are using the multi-threaded server architecture. The private space for a session in the SGA includes private SQL and PL/SQL areas, but not shared SQL and PL/SQL areas.

COMPOSITE_LIMIT
limits the total resource cost for a session. You must express the value of this parameter in service units.

Oracle calculates the total resource cost as a weighted sum of these resources:

* CPU_PER_SESSION
* CONNECT_TIME
* LOGICAL_READS_PER_SESSION
* PRIVATE_SGA

For information on how to specify the weight for each session resource see the ALTER RESOURCE COST command.

UNLIMITED
indicates that a user assigned this profile can use an unlimited amount of this resource.

DEFAULT
omits a limit for this resource in this profile. A user assigned this profile is subject to the limit for this resource specified in the DEFAULT profile.

PREREQUISITES:

You must have CREATE PROFILE system privilege.

SEE:
ALTER PROFILE, ALTER RESOURCE COST, ALTER SYSTEM, ALTER USER, DROP PROFILE

CREATE ROLE command

PURPOSE:

To create a role. A role is a set of privileges that can be granted to users or to other roles.

SYNTAX:

```
CREATE ROLE role
    [ NOT IDENTIFIED
    | IDENTIFIED {BY password | EXTERNALLY} ]
```

where:

role
: is the name of the role to be created. Oracle Corporation recommends that the role contain at least one single-byte character regardless of whether the database character set also contains multi-byte characters.

NOT IDENTIFIED
: indicates that a user granted the role need not be verified when enabling it.

IDENTIFIED
: indicates that a user granted the role must be verified when enabling it with the SET ROLE command:

    BY password
    : The user must specify the password to Oracle when enabling the role. The password can only contain single-byte characters from your database character set regardless of whether this character set also contains multi-byte characters.

    EXTERNALLY
    : The operating system verifies the user enabling to the role. Depending on the operating system, the user may have to specify a password to the operating system when enabling the role.

If you omit both the NOT IDENTIFIED option and the IDENTIFIED clause, the role defaults to NOT IDENTIFIED.

PREREQUISITES:

You must have CREATE ROLE system privilege.

SEE:
ALTER ROLE, DROP ROLE, GRANT (System Privileges and Roles), REVOKE (System Privileges and Roles), SET ROLE

CREATE ROLLBACK SEGMENT command

PURPOSE:

To create a rollback segment. A rollback segment is an object that is used by Oracle to store data necessary to reverse, or undo, changes made by transactions.

SYNTAX:

```
CREATE [PUBLIC] ROLLBACK SEGMENT rollback_segment
    [TABLESPACE tablespace]
    [STORAGE storage_clause]
```

where:

PUBLIC
specifies that the rollback segment is public and is available to any instance. If you omit this option, the rollback segment is private and is only available to the instance naming it in its initialization parameter ROLLBACK_SEGMENTS.

rollback_segment
is the name of the rollback segment to be created.

TABLESPACE
identifies the tablespace in which the rollback segment is created. If you omit this option, Oracle creates the rollback segment in the SYSTEM tablespace.

STORAGE
specifies the characteristics for the rollback segment.

PREREQUISITES:

You must have CREATE ROLLBACK SEGMENT system privilege. Also, you must have either space quota on the tablespace to contain the rollback segment or UNLIMITED TABLESPACE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the tablespace's label.

SEE:
ALTER ROLLBACK SEGMENT, CREATE DATABASE, CREATE TABLESPACE, DROP ROLLBACK SEGMENT, STORAGE clause

CREATE SCHEMA command

PURPOSE:

To create multiple tables and views and perform multiple grants in a single transaction.

SYNTAX:

```
CREATE SCHEMA AUTHORIZATION schema
    { CREATE TABLE command
    | CREATE VIEW command
    | GRANT command } ...
```

where:

schema
: is the name of the schema. The schema name must be the same as your Oracle username.

CREATE TABLE command
: is a CREATE TABLE statement to be issued as part of this CREATE SCHEMA statement.

CREATE VIEW command
: is a CREATE VIEW statement to be issued as part of this CREATE SCHEMA statement.

GRANT command
: is a GRANT statement (Objects Privileges) to be issued as part of this CREATE SCHEMA statement.

The CREATE SCHEMA statement only supports the syntax of these commands as defined by standard SQL, rather than the complete syntax supported by Oracle.

PREREQUISITES:

The CREATE SCHEMA statement can include CREATE TABLE, CREATE VIEW, and GRANT statements. To issue a CREATE SCHEMA statement, you must have the privileges necessary to issue the included statements.

SEE:
CREATE TABLE, CREATE VIEW, GRANT

CREATE SEQUENCE command

PURPOSE:

To create a sequence. A sequence is a database object from which multiple users may generate unique integers. You can use sequences to automatically generate primary key values.

SYNTAX:

```
CREATE SEQUENCE [schema.]sequence
    [INCREMENT BY integer]
    [START WITH integer]
    [MAXVALUE integer | NOMAXVALUE]
    [MINVALUE integer | NOMINVALUE]
    [CYCLE | NOCYCLE]
    [CACHE integer | NOCACHE]
    [ORDER | NOORDER]
```

where:

schema
is the schema to contain the sequence. If you omit schema, Oracle creates the sequence in your own schema.

sequence
is the name of the sequence to be created.

INCREMENT BY
specifies the interval between sequence numbers. This value can be any positive or negative Oracle integer, but it cannot be 0. If this value is negative, then the sequence descends. If the increment is positive, then the sequence ascends. If you omit this clause, the interval defaults to 1.

MINVALUE
specifies the sequence's minimum value.

NOMINVALUE
specifies a minimum value of 1 for an ascending sequence or -10 for a descending sequence.

The default is NOMINVALUE.

MAXVALUE
specifies the maximum value the sequence can generate.

NOMAXVALUE
specifies a maximum value of 10 for a descending sequence.

The default is NOMAXVALUE.

START WITH

specifies the first sequence number to be generated. You can use this option to start an ascending sequence at a value greater than its minimum or to start a descending sequence at a value less than its maximum. For ascending sequences, the default value is the sequence's minimum value. For descending sequences, the default value is the sequence's maximum value.

CYCLE
specifies that the sequence continues to generate values after reaching either its maximum or minimum value. After an ascending sequence reaches its maximum value, it generates its minimum value. After a descending sequence reaches its minimum, it generates its maximum.

NOCYCLE
specifies that the sequence cannot generate more values after reaching its maximum or minimum value.

The default is NOCYCLE.

CACHE
specifies how many values of the sequence Oracle preallocates and keeps in memory for faster access. The minimum value for this parameter is 2. For sequences that cycle, this value must be less than the number of values in the cycle.

NOCACHE
specifies that values of the sequence are not preallocated.

If you omit both the CACHE parameter and the NOCACHE option, Oracle caches 20 sequence numbers by default. However, if you are using Oracle with the Parallel Server option in parallel mode and you specify the ORDER option, sequence values are never cached, regardless of whether you specify the CACHE parameter or the NOCACHE option.

ORDER
guarantees that sequence numbers are generated in order of request. You may want to use this option if you are using the sequence numbers as timestamps. Guaranteeing order is usually not important for sequences used to generate primary keys.

NOORDER
does not guarantee sequence numbers are generated in order of request.

If you omit both the ORDER and NOORDER options, Oracle chooses NOORDER by default. Note that the ORDER option is only necessary to guarantee ordered generation if you are using Oracle with the Parallel Server option in parallel mode. If you are using exclusive mode, sequence numbers are always generated in order.

PREREQUISITES:

To create a sequence in your own schema, you must have CREATE

SEQUENCE privilege.

To create a sequence in another user's schema, you must have CREATE ANY SEQUENCE privilege. If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the creation label of the owner of the schema to contain the sequence.

SEE:

ALTER SEQUENCE, DROP SEQUENCE

CREATE SNAPSHOT command

PURPOSE:

To create a snapshot. A snapshot is a table that contains the results of a query of one or more tables or views, often located on a remote database.

SYNTAX:

```
CREATE SNAPSHOT [schema.]snapshot
    [ [PCTFREE  integer] [PCTUSED  integer]
      [INITRANS integer] [MAXTRANS integer]
      [TABLESPACE tablespace]
      [STORAGE storage_clause]
    [ USING INDEX [  PCTFREE integer | TABLESPACE tablespace
                   | INITTRANS integer | MAXTRANS integer
                   | STORAGE storage_clause ] ...
    | [CLUSTER cluster (column [, column]...)] ]
    [ REFRESH [FAST | COMPLETE | FORCE] [START WITH date] [NEXT date]]
    AS subquery
```

where:

schema
: is the schema to contain the snapshot. If you omit schema, Oracle creates the snapshot in your schema.

snapshot
: is the name of the snapshot to be created.

  Oracle chooses names for the table, views, and index used to maintain the snapshot by prefixing the snapshot name. To limit these names to 30 bytes and allow them to contain the entire snapshot name, Oracle Corporation recommends that you limit your snapshot names to 23 bytes.

PCTFREE
PCTUSED
INITRANS
MAXTRANS
: establishes values for these parameters for the internal table Oracle uses to maintain the snapshot's data.

TABLESPACE
: specifies the tablespace in which the snapshot is to be created. If you omit this option, Oracle creates the snapshot in the default tablespace of the owner of the snapshot's schema.

STORAGE
: establishes storage characteristics for the table Oracle uses to maintain the snapshot's data.

USING INDEX

specifies the storage characteristics for the index on a simple snapshot. If the USING INDEX clause not specified, the index is create with the same tablespace and storage parameters as the snapshot.

CLUSTER

creates the snapshot as part of the specified cluster. Since a clustered snapshot uses the cluster's space allocation, do not use the PCTFREE, PCTUSED, INITRANS, or MAXTRANS parameters, the TABLESPACE option, or the STORAGE clause in conjunction with the CLUSTER option.

REFRESH

specifies how and when Oracle automatically refreshes the snapshot:

FAST

specifies a fast refresh, or a refresh using only the updated data stored in the snapshot log associated with the master table.

COMPLETE

specifies a complete refresh, or a refresh that re-executes the snapshot's query.

FORCE

specifies a fast refresh if one is possible or complete refresh if a fast refresh is not possible. Oracle decides whether a fast refresh is possible at refresh time.
If you omit the FAST, COMPLETE, and FORCE options, Oracle uses FORCE by default.

START WITH

specifies a date expression for the first automatic refresh time.

NEXT

specifies a date expression for calculating the interval between automatic refreshes.

Both the START WITH and NEXT values must evaluate to a time in the future. If you omit the START WITH value, Oracle determines the first automatic refresh time by evaluating the NEXT expression when you create the snapshot. If you specify a START WITH value but omit the NEXT value, Oracle refreshes the snapshot only once. If you omit both the START WITH and NEXT values or if you omit the REFRESH clause entirely, Oracle does not automatically refresh the snapshot.

AS subquery

specifies the snapshot query. When you create the snapshot, Oracle executes this query and places the results in the snapshot. The select list can contain up to 253 expressions. A snapshot query is subject to the same restrictions as a view query.

PREREQUISITES:

To create a snapshot in your own schema, you must have CREATE SNAPSHOT system privilege. To create a snapshot in another user's schema, you must have CREATE ANY SNAPSHOT system privilege.

Before a snapshot can be created, the user SYS must run the SQL script DBMSSNAP.SQL on both the database to contain the snapshot and the database(s) containing the tables and views of the snapshot's query. This script creates the package SNAPSHOT which contains both public and private stored procedures used for refreshing the snapshot and purging the snapshot log. The exact name and location of this script may vary depending on your operating system.

When you create a snapshot, Oracle creates a table, two views, and an index in the schema of the snapshot. Oracle uses these objects to maintain the snapshot's data. You must have the privileges necessary to create these objects. For information on these privileges, see the CREATE TABLE, CREATE VIEW, and CREATE INDEX commands.

The owner of the schema containing the snapshot must have either space quota on the tablespace to contain the snapshot or UNLIMITED TABLESPACE system privilege. Also, both you (the creator) and the owner must also have the privileges necessary to issue the snapshot's query.

To create a snapshot, you must be using Oracle with the procedural option. To create a snapshot on a remote table or view, you must also be using the distributed option.

SEE:
ALTER SNAPHSOT, CREATE SNAPSHOT LOG, DROP SNAPSHOT

CREATE SNAPSHOT LOG command

PURPOSE:

To create a snapshot log. A snapshot log is a table associated with the master table of a snapshot. Oracle stores changes to the master table's data in the snapshot log and then uses the snapshot log to refresh the master table's snapshots.

SYNTAX:

```
CREATE SNAPSHOT LOG ON [schema.]table
    [PCTFREE  integer]    [PCTUSED  integer]
    [INITRANS integer]    [MAXTRANS integer]
    [TABLESPACE tablespace]
    [STORAGE storage_clause]
```

where:

schema
: is the schema containing the snapshot log's master table. If you omit schema, Oracle assumes the master table is contained in your own schema. Oracle creates the snapshot log in the schema of its master table. You cannot create a snapshot log for a table in the schema of the user SYS.

table
: is the name of the master table for which the snapshot log is to be created. You cannot create a snapshot log for a view.

  Oracle chooses names for the table and trigger used to maintain the snapshot log by prefixing the master table name. To limit these names to 30 bytes and allow them to contain the entire master table name, Oracle Corporation recommends that you limit master table names to 24 bytes.

PCTFREE
PCTUSED
INITRANS
MAXTRANS
: establishes values for these parameters for the snapshot log.

TABLESPACE
: specifies the tablespace in which the snapshot log is to be created. If you omit this option, Oracle creates the snapshot log in the default tablespace the owner of the snapshot log's schema.

STORAGE
: establishes storage characteristics for the snapshot log.

PREREQUISITES:

You must have the privileges necessary to create a table in the schema of the master table. For information on these privileges, see the CREATE TABLE command.

Before a snapshot log can be created, the user SYS must run the SQL script DBMSSNAP.SQL on the database containing the master table. This script creates the package SNAPSHOT which contains both public and private stored procedures used for refreshing the snapshot and urging the snapshot log. The exact name and location of this script may vary depending on your operating system.

You must also have the privileges to create a trigger on the master table. For information on these privileges, see the CREATE TRIGGER command.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the label of the tablespace in which the snapshot log is to be stored.

SEE:

ALTER SNAPSHOT LOG, CREATE SNAPSHOT, DROP SNAPSHOT LOG

CREATE SYNONYM command

PURPOSE:

To create a synonym. A synonym is an alternative name for a table, view, sequence, procedure, stored function, package, snapshot, or another synonym.

SYNTAX:

```
CREATE [PUBLIC] SYNONYM [schema.]synonym
    FOR [schema.]object[@dblink]
```

where:

PUBLIC
creates a public synonym. Public synonyms are accessible to all users. If you omit this option, the synonym is private and is accessible only within its schema.

schema
is the schema to contain the synonym. If you omit schema, Oracle creates the synonym in your own schema.

synonym
is the name of the synonym to be created.

FOR
identifies the object for which the synonym is created. If you do not qualify object with schema, Oracle assumes that the object is in your own schema. The object can be of these types:

* table
* view
* sequence
* stored procedure, function, or package
* snapshot
* synonym

The object cannot be contained in a package. Note that the object need not currently exist and you need not have privileges to access the object.

You can use a complete or partial dblink to create a synonym for an object on a remote database where the object is located. If you specify dblink and omit schema, the synonym refers to an object in the schema specified by the database link. Oracle Corporation recommends that you specify the schema containing the object in the remote database.

If you omit dblink, Oracle assumes the object is located on the local database.

PREREQUISITES:

To create a private synonym in your own schema, you must have CREATE SYNONYM system privilege.

To create a private synonym in another user's schema, you must have CREATE ANY SYNONYM system privilege. If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the creation label of the owner of schema to contain the synonym.

To create a PUBLIC synonym, you must have CREATE PUBLIC SYNONYM system privilege.

SEE:

CREATE DATABASE LINK, CREATE TABLE, CREATE VIEW

CREATE TABLE command

PURPOSE:

To create a table, the basic structure to hold user data, specifying this information:

* column definitions
* integrity constraints
* the table's tablespace
* storage characteristics
* an optional cluster
* data from an arbitrary query

SYNTAX:

```
CREATE TABLE [schema.]table
     ( { column datatype [DEFAULT expr] [column_constraint] ...
       | table_constraint}
    [, { column datatype [DEFAULT expr] [column_constraint] ...
       | table_constraint} ]...)
    [ [PCTFREE  integer] [PCTUSED  integer]
      [INITRANS integer] [MAXTRANS integer]
      [TABLESPACE tablespace]
      [STORAGE storage_clause]
    [  PARALLEL ( [ DEGREE { integer | DEFAULT } ]
                  [ INSTANCES { integer | DEFAULT } ]
                )
     | NOPARALLEL ]
    [  CACHE | NOCACHE  ]
    | [CLUSTER cluster (column [, column]...)] ]
    [ ENABLE   enable_clause
    | DISABLE disable_clause ] ...
    [AS subquery]
```

where:

schema
: is the schema to contain the table. If you omit schema, Oracle creates the table in your own schema.

table
: is the name of the table to be created.

column
: specifies the name of a column of the table. The number of columns in a table can range from 1 to 254.

datatype
: is the datatype of a column.

DEFAULT
: specifies a value to be assigned to the column if a subsequent INSERT statement omits a value for the column. The datatype of the

expression must match the datatype of the column. A DEFAULT expression cannot contain references to other columns, the pseudocolumns CURRVAL, NEXTVAL, LEVEL, and ROWNUM, or date constants that are not fully specified.

column_constraint
defines an integrity constraint as part of the column definition.

table_constraint
defines an integrity constraint as part of the table definition.

PCTFREE
specifies the percentage of space in each of the table's data blocks reserved for future updates to the table's rows. The value of PCTFREE must be a positive integer from 1 to 99. A value of 0 allows the entire block to be filled by inserts of new rows. The default value is 10. This value reserves 10% of each block for updates to existing rows and allows inserts of new rows to fill a maximum of 90% of each block.

PCTFREE has the same function in the commands that create and alter clusters, indexes, snapshots, and snapshot logs. The combination of PCTFREE and PCTUSED determines whether inserted rows will go into existing data blocks or into new blocks.

PCTUSED
specifies the minimum percentage of used space that Oracle maintains for each data block of the table. A block becomes a candidate for row insertion when its used space falls below PCTUSED. PCTUSED is specified as a positive integer from 1 to 99 and defaults to 40.

PCTUSED has the same function in the commands that create and alter clusters, snapshots, and snapshot logs.

The sum of PCTFREE and PCTUSED must be less than 100. You can use PCTFREE and PCTUSED together use space within a table more efficiently.

INITRANS
specifies the initial number of transaction entries allocated within each data block allocated to the table. This value can range from 1 to 255 and defaults to 1. In general, you should not change the INITRANS value from its default.

Each transaction that updates a block requires a transaction entry in the block. The size of a transaction entry depends on your operating system.

This parameter ensures that a minimum number of concurrent transactions can update the block and helps avoid the overhead of dynamically allocating a transaction entry.

The INITRANS parameter serves the same purpose in clusters, indexes, snapshots, and snapshot logs as in tables. The minimum and default INITRANS value for a cluster or index is 2, rather than 1.

MAXTRANS
specifies the maximum number of concurrent transactions that can update a data block allocated to the table. This limit does not apply to queries. This value can range from 1 to 255 and the default is a function of the data block size. You should not change the MAXTRANS value from its default.

If the number concurrent transactions updating a block exceeds the INITRANS value, Oracle dynamically allocates transaction entries in the block until either the MAXTRANS value is exceeded or the block has no more free space.

The MAXTRANS parameter serves the same purpose in clusters, snapshots, and snapshot logs as in tables.

TABLESPACE
specifies the tablespace in which Oracle creates the table. If you omit this option, then Oracle creates the table in the default tablespace of the owner of the schema containing the table.

STORAGE
specifies the storage characteristics for the table. This clause has performance ramifications for large tables. Storage should be allocated to minimize dynamic allocation of additional space.

PARALLEL
DEGREE specifies the number of query server processes that can scan the table in parallel. Either specify a positive integer or DEFAULT which signifies to use the initialization parameter PARALLEL_DEFAULT_SCANSIZE to estimate the number of query servers to use.

INSTANCES specifies the minimum number of instances that need to be available before the table can be spread across all available instances of a Parallel Server. A positive integer specifies the number of instances. DEFAULT signifies that the parameter PARALLEL_MAX_PARTITIONSIZE is used to calculate whether a table is split across all instances' buffer caches.

NOPARALLEL
specifies that queries on this table are not performed in parallel by default. A hint in the query still causes the query to be performed in parallel.

CACHE
specifies that blocks of this table are placed on the most recently used end of the LRU list of the buffer cache when the a full table scan is performed.
This option is useful for small lookup tables.

NOCACHE
specifies that blocks of the table in the buffer cache follow the standard LRU algorithm when a full table scan is performed.

CLUSTER

specifies that the table is to be part of the cluster. The columns listed in this clause are the table columns that correspond to the cluster's columns. Generally, the cluster columns of a table are the column or columns that comprise its primary key or a portion of its primary key.

Specify one column from the table for each column in the cluster key. The columns are matched by position, not by name. Since a clustered table uses the cluster's space allocation, do not use the PCTFREE, PCTUSED, INITRANS, or MAXTRANS parameters, the TABLESPACE option, or the STORAGE clause in conjunction with the CLUSTER option.

ENABLE
enables an integrity constraint.

DISABLE
disables an integrity constraint.

Constraints specified in the ENABLE and DISABLE clauses of a CREATE TABLE statement must be defined in the statement. You can also enable and disable constraints with the ENABLE and DISABLE keywords of the CONSTRAINT clause. If you define a constraint but do not explicitly enable or disable it, Oracle enables it by default.

You cannot use the ENABLE and DISABLE clauses in a CREATE TABLE statement to enable and disable triggers.

AS subquery
inserts the rows returned by the subquery into the table upon its creation.

If you include this clause, the column definitions can only specify column names, default values, and integrity constraints, not datatypes. Oracle derives column datatypes and lengths from the subquery. Oracle also automatically defines NOT NULL constraints on columns in the new table if they existed on the corresponding columns of the selected table and the subquery does not modify the column value with a SQL function or operator. A CREATE TABLE statement cannot contain both the AS clause and a referential integrity constraint definition.

The number of columns must equal the number of expressions in the subquery. If all expressions in the subquery are columns, you can omit the columns from the table definition entirely. In this case, the names of the columns of table are the same as the columns in the subquery.

PREREQUISITES:

To create a table in your own schema, you must have CREATE TABLE system privilege. To create a table in another user's schema, you must have CREATE ANY TABLE system privilege. Also, the owner of the schema to contain the table must have either space quota on the tablespace to contain the table or UNLIMITED TABLESPACE system

privilege.

SEE:
ALTER TABLE, CONSTRAINT clause, CREATE CLUSTER, CREATE INDEX, CREATE TABLESPACE, DISABLE clause, DROP TABLE, ENABLE clause, STORAGE clause

CREATE TABLESPACE command

PURPOSE:

To create a tablespace. A tablespace is an allocation of space in the database that can contain objects.

SYNTAX:

```
CREATE TABLESPACE tablespace
    DATAFILE filespec [, filespec] ...
    [DEFAULT STORAGE storage_clause]
    [ONLINE | OFFLINE]
```

where:

tablespace
is the name of the tablespace to be created.

DATAFILE
specifies the data file or files to comprise the tablespace.

DEFAULT STORAGE
specifies the default storage parameters for all objects created in the tablespace.

ONLINE
makes the tablespace available immediately after creation to users who have been granted access to the tablespace.

OFFLINE
makes the tablespace unavailable after immediately after creation.

If you omit both the ONLINE and OFFLINE options, Oracle creates the tablespace online by default. The data dictionary view DBA_TABLESPACES indicates whether each tablespace is online or offline.

PREREQUISITES:

You must have CREATE TABLESPACE system privilege. Also, the SYSTEM tablespace must contain at least two rollback segments including the SYSTEM rollback segment.

SEE:
ALTER TABLESPACE, DROP TABLESPACE

CREATE TRIGGER command

PURPOSE:

To create and enable a database trigger. A database trigger is a stored PL/SQL block that is associated with a table. Oracle automatically executes a trigger when a specified SQL statement is issued against the table.

SYNTAX:

```
CREATE [OR REPLACE] TRIGGER [schema.]trigger
    {BEFORE | AFTER}
    {DELETE | INSERT | UPDATE [OF column [, column] ...]}
[OR {DELETE | INSERT | UPDATE [OF column [, column] ...]}] ...
    ON [schema.]table
    [ [REFERENCING { OLD [AS] old [NEW [AS] new]
                   | NEW [AS] new [OLD [AS] old] } ]
     FOR EACH ROW
     [WHEN (condition)] ]
    pl/sql_block
```

where:

OR REPLACE
recreates the trigger if it already exists. You can use this option to change the definition of an existing trigger without first dropping it.

schema
is the schema to contain the trigger. If you omit schema, Oracle creates the trigger in your own schema.

trigger
is the name of the trigger to be created.

BEFORE
indicates that Oracle fires the trigger before executing the triggering statement.

AFTER
indicates that Oracle fires the trigger after executing the triggering statement.

DELETE
indicates that Oracle fires the trigger whenever a DELETE statement removes a row from the table.

INSERT
indicates that Oracle fires the trigger whenever an INSERT statement adds a row to table.

UPDATE...OF
indicates that Oracle fires the trigger whenever an UPDATE statement

changes a value in one of the columns specified in the OF clause. If you omit the OF clause, Oracle fires the trigger whenever an UPDATE statement changes a value in any column of the table.

ON
specifies the schema and name of the table on which the trigger is to be created. If you omit schema, Oracle assumes the table is in our own schema. You cannot create a trigger on a table in the schema SYS.

REFERENCING
specifies correlation names. You can use correlation names in the PL/SQL block and WHEN clause of a row trigger to refer specifically to old and new values of the current row. The default correlation names are OLD and NEW. If your row trigger is associated with a table named OLD or NEW, you can use this clause to specify different correlation names to avoid confusion between the table name and the correlation name.

FOR EACH ROW
designates the trigger to be a row trigger. Oracle fires a row trigger once for each row that is affected by the triggering statement and meets the optional trigger constraint defined in the WHEN clause.

If you omit this clause, the trigger is a statement trigger. Oracle fires a statement trigger only once when the triggering statement is issued if the optional trigger constraint is met.

WHEN
specifies the trigger restriction. The trigger restriction contains a SQL condition that must be satisfied for Oracle to fire the trigger. This condition must contain correlation names and cannot contain a query.

You can only specify a trigger restriction for a row trigger. Oracle evaluates this condition for each row affected by the triggering statement.

pl/sql_block
is the PL/SQL block that Oracle executes to fire the trigger.

Note that the PL/SQL block of a trigger cannot contain transaction control SQL statements (COMMIT, ROLLBACK, and SAVEPOINT).

To embed a CREATE TRIGGER statement inside an Oracle Precompiler program, you must terminate the statement with the keyword END-EXEC followed by the embedded SQL statement terminator for the specific language.

PREREQUISITES:

Before a trigger can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

To issue this statement, you must have one of these system privileges:

CREATE TRIGGER
: This system privilege allows you to create a trigger in your own schema on a table in your own schema.

CREATE ANY TRIGGER
: This system privilege allows you to create a trigger in any user's schema on a table in any user's schema.

If the trigger issues SQL statements or calls procedures or functions, then the owner of the schema to contain the trigger must have the privileges necessary to perform these operations. These privileges must be granted directly to the owner, rather than acquired through roles.

To create a trigger, you must be using Oracle with the procedural option.

SEE:

ALTER TRIGGER, DISABLE clause, DROP TRIGGER, ENABLE clause

CREATE USER command

PURPOSE:

To create a database user, or an account through which you can log in to the database, and establish the means by which Oracle permits access by the user. You can optionally assign these properties to the user:

* default tablespace
* temporary tablespace
* quotas for allocating space in tablespaces
* profile containing resource limits

SYNTAX:

```
CREATE USER user
    IDENTIFIED {BY password | EXTERNALLY}
    [DEFAULT TABLESPACE tablespace]
    [TEMPORARY TABLESPACE tablespace]
    [QUOTA {integer [K|M] | UNLIMITED} ON tablespace] ...
    [PROFILE profile]
```

where:

user
: is the name of the user to be created. This name can only contain characters from your database character set. Oracle Corporation recommends that the user contain at least one single-byte character regardless of whether the database character set also contains multi-byte characters.

IDENTIFIED
: indicates how Oracle permits user access:

  BY password
  : The user must specify this password to logon. The password can only contain single-byte characters from your database character set regardless of whether this character set also contains multi-byte characters.

  EXTERNALLY
  : Oracle verifies user access through the operating system.

DEFAULT TABLESPACE
: identifies the default tablespace for objects that the user creates. If you omit this clause, objects default to the SYSTEM tablespace.

TEMPORARY TABLESPACE
: identifies the tablespace for the user's temporary segments. If you omit this clause, temporary segments default to the SYSTEM tablespace.

QUOTA
: allows the user to allocate space in the tablespace and optionally

establishes a quota of integer bytes. This quota is the maximum space in the tablespace the user can allocate. You can also use the K or M to specify the quota in kilobytes or megabytes.

UNLIMITED

allows the user to allocate space in the tablespace without bound.

PROFILE

assigns the profile named profile to the user. The profile limits the amount of database resources the user can use. If you omit this clause, Oracle assigns the DEFAULT profile to the user.

PREREQUISITES:

You must have CREATE USER system privilege.

If you are using Trusted Oracle in DBMS MAC mode, you must meet additional prerequisites to perform the optional assignments of this statement:

* To assign a default or temporary tablespace, your DBMS label must dominate the tablespace's creation label.
* To assign a profile, your DBMS label must dominate the profile's creation label.

SEE:

ALTER USER, CREATE PROFILE, CREATE TABLESPACE

CREATE VIEW command

PURPOSE:

To define a view, a logical table based on one or more tables or views.

SYNTAX:

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW [schema.]view
    [(alias [,alias]...)]
    AS subquery
    [WITH CHECK OPTION [CONSTRAINT constraint]]
```

where:

OR REPLACE
: recreates the view if it already exists. You can use this option to change the definition of an existing view without dropping, recreating, and regranting object privileges previously granted on it.

FORCE
: creates the view regardless of whether the view's base tables exist or the owner of the schema containing the view has privileges on them. Note that both of these conditions must be true before any SELECT, INSERT, UPDATE, or DELETE statements can be issued against the view.

NOFORCE
: creates the view only if the base tables exist and the owner of the schema containing the view has privileges on them.

  The default is NOFORCE.

schema
: is the schema to contain the view. If you omit schema, Oracle creates the view in your own schema.

view
: is the name of the view.

alias
: specifies names for the expressions selected by the view's query. The number of aliases must match the number of expressions selected by the view. Aliases must follow the rules for naming schema objects. Aliases must be unique within the view.

  If you omit the aliases, Oracle derives them from the columns or column aliases in the view's query. For this reason, you must use aliases if the view's query contains expressions rather than only column names.

AS subquery
: identifies columns and rows of the table(s) that the view is based

on. A view's query can be any SELECT statement without the ORDER BY or FOR UPDATE clauses. Its select list can contain up to 254 expressions.

WITH CHECK OPTION
specifies that inserts and updates performed through the view must result in rows that the view query can select. The CHECK OPTION cannot make this guarantee if there is a subquery in the query of this view or any view on which this view is based.

CONSTRAINT
is the name assigned to the CHECK OPTION constraint. If you omit this identifier, Oracle automatically assigns the constraint a name of this form:

SYS_Cn

where

n

is an integer that makes the constraint name unique within the database.

PREREQUISITES:

To create a view in your own schema, you must have CREATE VIEW system privilege. To create a view in another user's schema, you must have CREATE ANY VIEW system privilege.

The owner of the schema containing the view must have the privileges necessary to either select, insert, update, or delete rows from all the tables or views on which the view is based. For information on these privileges, see the SELECT, INSERT, UPDATE, and DELETE commands. The owner must be granted these privileges directly, rather than through a role.

SEE:
CREATE TABLE, CREATE SYNONYM, DROP VIEW, RENAME

DELETE command

PURPOSE:

To remove rows from a table or from a view's base table.

SYNTAX:

```
DELETE [FROM] [schema.]{table | view}[@dblink] [alias]
    [WHERE condition]
```

where:

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

table
view
is the name of a table from which the rows are to be deleted. If you specify view, Oracle deletes rows from the view's base table.

dblink
is the complete or partial name of a database link to a remote database where the table or view is located. You can only delete rows from a remote table or view if you are using Oracle with the distributed option.

If you omit dblink, Oracle assumes that the table or view is located on the local database.

alias
is an alias assigned to the table. Aliases are generally used in DELETE statements with correlated queries.

WHERE
deletes only rows that satisfy the condition. The condition can reference the table and can contain a subquery. If you omit this clause, Oracle deletes all rows from the table.

PREREQUISITES:

For you to delete rows from a table, the table must be in your own schema or you must have DELETE privilege on the table.

For you to delete rows from the base table of a view, the owner of the schema containing the view must have DELETE privilege on the base table. Also, if the view is in a schema other than your own, you must be granted DELETE privilege on the view.

The DELETE ANY TABLE system privilege also allows you to delete rows from any table or any view's base table.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the creation label of the table or view or you must

meet one of these criteria:

* If the creation label of the table or view is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the creation label of your table or view is noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

In addition, for each row to be deleted, your DBMS label must match the row's label or you must meet one of these criteria:

* If the row's label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the row's label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the row's label is noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

DROP TABLE, TRUNCATE

DISABLE clause

PURPOSE:

To disable an integrity constraint or all triggers associated with a table:

* If you disable an integrity constraint, Oracle does not enforce it. However, disabled integrity constraints appear in the data dictionary along with enabled integrity constraints.
* If you disable a trigger, Oracle does not fire it if its triggering condition is satisfied.

SYNTAX:

```
DISABLE {     { UNIQUE (column [, column] ...)
              | PRIMARY KEY
              | CONSTRAINT constraint }
                  [CASCADE]
         |    ALL TRIGGERS }
```

where:

UNIQUE
disables the UNIQUE constraint defined on the specified column or combination of columns.

PRIMARY KEY
disables the table's PRIMARY KEY constraint.

CONSTRAINT
disables the integrity constraint with the name constraint.

CASCADE
disables any integrity constraints that depend on the specified integrity constraint. To disable a primary or unique key that is part of a referential integrity constraint, you must specify this option.

ALL TRIGGERS
disables all triggers associated with the table. This option can only appear in a DISABLE clause in an ALTER TABLE statement, not a CREATE TABLE statement.

PREREQUISITES:

A DISABLE clause that disables an integrity constraint can appear in either a CREATE TABLE or ALTER TABLE command. To disable an integrity constraint, you must have the privileges necessary to issue one of these commands. For information on these privileges, see the CREATE TABLE and ALTER TABLE commands.

For an integrity constraint to appear in a DISABLE clause, one of these conditions must be true:

* the integrity constraint must be defined in the containing statement.
* the integrity constraint must already have been defined and enabled in previously issued statements.

A DISABLE clause that disables triggers can only appear in an ALTER TABLE statement. To disable triggers with a DISABLE clause, you must have the privileges necessary to issue this statement. For information on these privileges, see the ALTER TABLE command. Also, the triggers must be in your own schema or you must have ALTER ANY TRIGGER system privilege.

SEE:

ALTER TABLE, ALTER TRIGGER, CONSTRAINT clause, CREATE TABLE, CREATE TRIGGER, ENABLE clause

DROP clause

PURPOSE:

To remove an integrity constraint from the database.

SYNTAX:

```
DROP { PRIMARY KEY
     | UNIQUE (column [, column] ...)
     | CONSTRAINT constraint }
     [CASCADE]
```

where:

PRIMARY KEY
drops the table's PRIMARY KEY constraint.

UNIQUE
drops the UNIQUE constraint on the specified columns.

CONSTRAINT
drops the integrity constraint named constraint.

CASCADE
drops all other integrity constraints that depend on the dropped integrity constraint.

PREREQUISITES:

The DROP clause can appear in an ALTER TABLE statement. To drop an integrity constraint, you must have the privileges necessary to issue an ALTER TABLE statement. For information on these privileges, see the ALTER TABLE command earlier.

SEE:
ALTER TABLE, CONSTRAINT clause

DROP CLUSTER command

PURPOSE:

To remove a cluster from the database.

SYNTAX:

```
DROP CLUSTER [schema.]cluster
    [INCLUDING TABLES [CASCADE CONSTRAINTS] ]
```

where:

schema
is the schema containing the cluster. If you omit schema, Oracle assumes the cluster is in your own schema.

cluster
is the name of the cluster to be dropped.

INCLUDING TABLES
drops all tables that belong to the cluster. If you omit this clause, and the cluster still contains tables, Oracle returns an error and does not drop the cluster.

CASCADE CONSTRAINTS
drops all referential integrity constraints from tables outside the cluster that refer to primary and unique keys in the tables of the cluster. If you omit this option and such referential integrity constraints exist, Oracle returns an error and does not drop the cluster.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the cluster's creation label or you must satisfy one of these criteria:

* If the cluster's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the cluster's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the cluster's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
DROP TABLE

DROP DATABASE LINK command

PURPOSE:

To remove a database link from the database.

SYNTAX:

```
DROP [PUBLIC] DATABASE LINK dblink
```

where:

PUBLIC
must be specified to drop a PUBLIC database link.

dblink
specifies the database link to be dropped.

PREREQUISITES:

To drop a private database link, the database link must be in your own schema. To drop a PUBLIC database link, you must have DROP PUBLIC DATABASE LINK system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the database link's creation label or you must satisfy one of these criteria:

* If the database link's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the database link's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the database link's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE DATABASE LINK

DROP FUNCTION command

PURPOSE:

To remove a stand-alone stored function from the database.

SYNTAX:

```
DROP FUNCTION [schema.]function
```

where:

schema
is the schema containing the function. If you omit schema, Oracle assumes the function is in your own schema.

function
is the name of the function to be dropped.

PREREQUISITES:

The function must be in your own schema or you must have DROP ANY PROCEDURE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the function's creation label or you must satisfy one of these criteria:

* If the function's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the function's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the function's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE FUNCTION

DROP INDEX command

PURPOSE:

To remove an index from the database.

SYNTAX:

```
DROP INDEX [schema.]index
```

where:

schema
is the schema containing the index. If you omit schema, Oracle assumes the index is in your own schema.

index
is the name of the index to be dropped.

PREREQUISITES:

The index must be in your own schema or you must have DROP ANY INDEX system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the index's creation label or you must satisfy one of these criteria:

* If the index's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the index's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the index's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
ALTER INDEX, CREATE INDEX, CREATE TABLE

DROP PACKAGE command

PURPOSE:

To remove a stored package from the database.

SYNTAX:

```
DROP PACKAGE [BODY] [schema.]package
```

where:

BODY
drops only the body of the package. If you omit this option, Oracle drops both the body and specification of the package.

schema
is the schema containing the package. If you omit schema, Oracle assumes the package is in your own schema.

package
is the name of the package to be dropped.

PREREQUISITES:

The package must be in your own schema or you must have DROP ANY PROCEDURE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the cluster's creation label or you must satisfy one of these criteria:

* If the package's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the package's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the package's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE PACKAGE

DROP PROCEDURE command

PURPOSE:

To remove a stand-alone stored procedure from the database.

SYNTAX:

```
DROP PROCEDURE [schema.]procedure
```

where:

schema
is the schema containing the procedure. If you omit schema, Oracle assumes the procedure is in your own schema.

procedure
is the name of the procedure to be dropped.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the cluster's creation label or you must satisfy one of these criteria:

* If the procedure's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the procedure's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the procedure's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE PROCEDURE

DROP PROFILE command

PURPOSE:

To remove a profile from the database.

SYNTAX:

```
DROP PROFILE profile
    [CASCADE]
```

where:

profile
is the name of the profile to be dropped.

CASCADE
deassigns the profile from any users to whom it is assigned. Oracle automatically assigns the DEFAULT profile to such users. You must specify this option to drop a profile that is currently assigned to users.

PREREQUISITES:

You must have DROP PROFILE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the profile's creation label or you must satisfy one of these criteria:

* If the profile's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the profile's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the profile's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE PROFILE

DROP ROLE command

PURPOSE:

To remove a role from the database.

SYNTAX:

```
DROP ROLE role
```

where:

role
is the role to be dropped.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the role's creation label or you must satisfy one of these criteria:

* If the role's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the role's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.

If the role's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE ROLE, SET ROLE

DROP ROLLBACK SEGMENT command

PURPOSE:

To remove a rollback segment from the database.

SYNTAX:

```
DROP ROLLBACK SEGMENT rollback_segment
```

where:

rollback_segment
is the name the rollback segment to be dropped.

PREREQUISITES:

You must have DROP ROLLBACK SEGMENT system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the rollback segment's creation label or you must satisfy one of these criteria:

* If the rollback segment's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the rollback segment's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the rollback segment's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
ALTER ROLLBACK SEGMENT, CREATE ROLLBACK SEGMENT, CREATE TABLESPACE

DROP SEQUENCE command

PURPOSE:

To remove a sequence from the database.

SYNTAX:

```
DROP SEQUENCE [schema.]sequence
```

where:

schema
is the schema containing the sequence. If you omit schema, Oracle assumes the sequence is in your own schema.

sequence
is the name of the sequence to be dropped.

PREREQUISITES:

The sequence must be in your own schema or you must have DROP ANY SEQUENCE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the sequence's creation label or you must satisfy one of these criteria:

* If the sequence's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the sequence's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the sequence's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
ALTER SEQUENCE, CREATE SEQUENCE

DROP SNAPSHOT command

PURPOSE:

To remove a snapshot from the database.

SYNTAX:

```
DROP SNAPSHOT [schema.]snapshot
```

where:

schema
is the schema containing the snapshot. If you omit schema, Oracle assumes the snapshot is in your own schema.

snapshot
is the name of the snapshot to be dropped.

PREREQUISITES:

The snapshot must be in your own schema or you must have DROP ANY SNAPSHOT system privilege. You must also have the privileges to drop the internal table, views, and index that Oracle uses to maintain the snapshot's data. For information on these privileges, see the DROP TABLE, DROP VIEW, and DROP INDEX commands.

SEE:
CREATE SNAPSHOT

DROP SNAPSHOT LOG command

PURPOSE:

To remove a snapshot log from the database.

SYNTAX:

```
DROP SNAPSHOT LOG ON [schema.]table
```

where:

schema
: is the schema containing the snapshot log and its master table. If you omit schema, Oracle assumes the snapshot log and master table are in your own schema.

table
: is the name of the master table associated with the snapshot log to be dropped.

PREREQUISITES:

Since a snapshot log consists of a table and a trigger, the privileges that authorize operations on it are the same as for a table. To drop a snapshot log, you must have the privileges listed for the DROP TABLE command. You must also have the privileges to drop a trigger from the snapshot log's master table. For information on these privileges, see the DROP TRIGGER command.

SEE:

CREATE SNAPSHOT LOG

DROP SYNONYM command

PURPOSE:

To remove a synonym from the database.

SYNTAX:

```
DROP [PUBLIC] SYNONYM [schema.]synonym
```

where:

PUBLIC
must be specified to drop a public synonym.

schema
is the schema containing the synonym. If you omit schema, Oracle assumes the synonym is in your own schema.

synonym
is the name of the synonym to be dropped.

PREREQUISITES:

If you want to drop a private synonym, either the synonym must be in your own schema or you must have DROP ANY SYNONYM system privilege. If you want to drop a PUBLIC synonym, either the synonym must be in your own schema or you must have DROP ANY PUBLIC SYNONYM system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the synonym's creation label or you must satisfy one of these criteria:

* If the synonym's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the synonym's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the synonym's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE SYNONYM

DROP TABLE command

PURPOSE:

To remove a table and all its data from the database.

SYNTAX:

```
DROP TABLE [schema.]table
    [CASCADE CONSTRAINTS]
```

where:

schema
is the schema containing the table. If you omit schema, Oracle assumes the table is in your own schema.

table
is the name of the table to be dropped.

CASCADE CONSTRAINTS
drops all referential integrity constraints that refer to primary and unique keys in the dropped table. If you omit this option, and such referential integrity constraints exist, Oracle returns an error and does not drop the table.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the table's creation label or you must satisfy one of these criteria:

* If the table's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the table's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the table's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
ALTER TABLE, CREATE INDEX, CREATE TABLE, DROP CLUSTER

DROP TABLESPACE command

PURPOSE:

To remove a tablespace from the database.

SYNTAX:

```
DROP TABLESPACE tablespace
    [INCLUDING CONTENTS [CASCADE CONSTRAINTS]]
```

where:

tablespace
is the name of the tablespace to be dropped.

INCLUDING CONTENTS
drops all the content of the tablespace. You must specify this clause to drop a tablespace that contains any database objects. If you omit this clause, and the tablespace is not empty, Oracle returns an error and does not drop the tablespace.

CASCADE CONSTRAINTS
drops all referential integrity constraints from tables outside the tablespace that refer to primary and unique keys in the tables of the tablespace. If you omit this option and such referential integrity constraints exist, Oracle returns an error and does not drop the tablespace.

PREREQUISITES:

You must have DROP TABLESPACE system privilege. No rollback segments in the tablespace can be assigned active transactions.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the tablespace's creation label or you must satisfy one of these criteria:

* If the tablespace's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the tablespace's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the tablespace's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
ALTER TABLESPACE, CREATE DATABASE, CREATE TABLESPACE

DROP TRIGGER command

PURPOSE:

To remove a database trigger from the database.

SYNTAX:

```
DROP TRIGGER [schema.]trigger
```

where:

schema
is the schema containing the trigger. If you omit schema, Oracle assumes the trigger is in your own schema.

trigger
is the name of the trigger to be dropped.

PREREQUISITES:

The trigger must be in your own schema or you must have DROP ANY TRIGGER system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the trigger's creation label or you must satisfy one of these criteria:

* If the trigger's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the trigger's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the trigger's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE TRIGGER

DROP USER command

PURPOSE:

To remove a database user and optionally remove the user's objects.

SYNTAX:

```
DROP USER user [CASCADE]
```

where:

user
: is the user to be dropped.

CASCADE
: drops all objects in the user's schema before dropping the user. You must specify this option to drop a user whose schema contains any objects.

PREREQUISITES:

You must have DROP USER system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the user's creation label or you must satisfy one of these criteria:

* If the user's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the user's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the user's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE USER, other DROP commands

DROP VIEW command

PURPOSE:

To remove a view from the database.

SYNTAX:

```
DROP VIEW [schema.]view
```

where:

schema
is the schema containing the view. If you omit schema, Oracle assumes the view is in your own schema.

view
is the name of the view to be dropped.

PREREQUISITES:

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

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the view's creation label or you must satisfy one of these criteria:

* If the view's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the view's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the view's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE SYNONYM, CREATE TABLE, CREATE VIEW

ENABLE clause

PURPOSE:

To enable an integrity constraint or all triggers associated with a table:

* If you enable a constraint, Oracle enforces it by applying it to all data in the table. All table data must satisfy an enabled constraint.
* If you enable a trigger, Oracle fires the trigger whenever its triggering condition is satisfied.

SYNTAX:

```
ENABLE { {UNIQUE (column [, column] ...)
         |PRIMARY KEY
         |CONSTRAINT constraint}
            [USING INDEX [INITRANS integer]
                         [MAXTRANS integer]
                         [TABLESPACE tablespace]
                         [STORAGE storage_clause]
                         [PCTFREE integer]
              [PARALLEL [integer] | NOPARALLEL]
            [EXCEPTIONS INTO [schema.]table ]
       | ALL TRIGGERS }
```

where:

UNIQUE
enables the UNIQUE constraint defined on the specified column or combination of columns.

PRIMARY KEY
enables the table's PRIMARY KEY constraint.

CONSTRAINT
enables the integrity constraint named constraint.

USING INDEX
specifies parameters for the index Oracle creates to enforce a UNIQUE or PRIMARY KEY constraint. Oracle gives the index the same name as the constraint. You can choose the values of the INITRANS, MAXTRANS, TABLESPACE, STORAGE, and PCTFREE parameters for the index. For information on these parameters, see the CREATE TABLE command.

Only use these parameters when enabling UNIQUE and PRIMARY KEY constraints.

PARALLEL
specifies the number of processes that create the index in parallel. You can only specify positive integer values greater than 1. If you do not specify an integer, the degree of parallelism is based on the parallelism specified in the table's definition.

NOPARALLEL
specifies that the index should not be created in parallel.

EXCEPTIONS INTO
identifies an table into which Oracle places information about rows that violate the integrity constraint. The table must exist before you use this option. If you omit schema, Oracle assumes the exception table is in your own schema. The exception table must be on your local database.

ALL TRIGGERS
enables all triggers associated with the table. You can only use this option in an ENABLE clause in an ALTER TABLE statement, not a CREATE TABLE statement.

PREREQUISITES:

An ENABLE clause that enables an integrity constraint can appear in either a CREATE TABLE or ALTER TABLE statement. To enable a constraint in this manner, you must have the privileges necessary to issue one of these statements. For information on these privileges, see the CREATE TABLE or ALTER TABLE command.

If you enable a UNIQUE or PRIMARY KEY constraint, Oracle creates an index on the columns of the unique or primary key in the schema containing the table. To enable such a constraint, you must have the privileges necessary to create the index. For information on these privileges, see the CREATE INDEX command.

If you enable a referential integrity constraint, the referenced UNIQUE or PRIMARY KEY constraint must already be enabled.

For an integrity constraint to appear in an ENABLE clause, one of these conditions must be true:

* the integrity constraint must be defined in the containing statement
* the integrity constraint must already have been defined and disabled in a previously issued statement

An ENABLE clause that enables triggers can appear in an ALTER TABLE statement. To enable triggers with the ENABLE clause, you must have the privileges necessary to issue this statement. For information on these privileges, see the ALTER TABLE command. Also, the triggers must be in your own schema or you must have ALTER ANY TRIGGER system privilege.

SEE:
ALTER TABLE, ALTER TRIGGER, CONSTRAINT clause, CREATE TABLE, CREATE TRIGGER, DISABLE clause, STORAGE clause

EXPLAIN PLAN command

PURPOSE:

To determine the execution plan Oracle follows to execute a specified SQL statement. This command inserts a row describing each step of the execution plan into a specified table. If you are using cost-based optimization, this command also determines the cost of executing the statement.

SYNTAX:

```
EXPLAIN PLAN
    [SET STATEMENT ID = 'text']
    [INTO [schema.]table[@dblink]]
    FOR statement
```

where:

SET
specifies the value of the STATEMENT_ID column for the rows of the execution plan in the output table. If you omit this clause, the STATEMENT_ID value defaults to null.

INTO
specifies the schema, name, and database containing the output table. This table must exist before you use the EXPLAIN PLAN command. If you omit schema, Oracle assumes the table is in your own schema.

The dblink can be a complete or partial name of a database link to a remote Oracle7 database where the output table is located. You can only specify a remote output table if you are using Oracle with the distributed option. If you omit dblink, Oracle assumes the table is on your local database.

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

FOR
specifies a SELECT, INSERT, UPDATE, or DELETE statement for which the execution plan is generated.

PREREQUISITES:

To issue an EXPLAIN PLAN statement, you must have the privileges necessary to insert rows into an existing output table that you specify to hold the execution plan. For information on these privileges, see the INSERT command.

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

to access both the other view and its underlying table.

To examine the execution plan produced by an EXPLAIN PLAN statement, you must have the privileges necessary to query the output table. For more information on these privileges, see the SELECT command.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the output table's creation label or you must satisfy one of these criteria:

* If the output table's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the output table's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

Appendix B, Performance Diagnostic Tools, in the Oracle7 Server Application Developer's Guide.

Filespec statement

PURPOSE:

To either specify a file as a data file or specify a group of one or more files as a redo log file group.

SYNTAX:

Data files:

```
'filename' [SIZE integer [K|M] ] [REUSE]
```

Redo log file groups:

```
{  'filename'
| ('filename' [, 'filename'] ...)}
    [SIZE integer [K|M] ] [REUSE]
```

where:

'filename'
is the name of either a data file or a redo log file member. A redo log file group can have one or more members, or copies. Each 'filename' must be fully specified according to the conventions for your operating system.

SIZE
specifies the size of the file. If you omit this parameter, the file must already exist.

K
specifies the size in kilobytes.

M
specifies the size in megabytes.

If you omit K and M, the size is specified in bytes.

REUSE
allows Oracle to reuse an existing file. If the file already exists, Oracle verifies that its size matches the value of the SIZE parameter. If the file does not exist, Oracle creates it. If you omit this option, the file must not already exist and Oracle creates the file.

The REUSE option is only significant when used with the SIZE option. If you omit the SIZE option, Oracle expects the file to exist already. Note that whenever Oracle uses an existing file, the file's previous contents are lost.

PREREQUISITES:

A filespec can appear in either CREATE DATABASE, ALTER DATABASE, CREATE TABLESPACE, or ALTER TABLESPACE commands. You must have the privileges necessary to issue one of these commands. For information on these privileges, see the CREATE DATABASE, ALTER

DATABASE, CREATE TABLESPACE, and ALTER TABLESPACE commands.

SEE:
ALTER_DATABASE, ALTER_TABLESPACE, CREATE_DATABASE, CREATE TABLESPACE

GRANT command (System Privileges and Roles)

PURPOSE:

To grant system privileges and roles to users and roles. To grant object privileges, use the GRANT command (Object Privileges).

SYNTAX:

```
GRANT {system_priv | role} [, {system_priv | role}] ...
    TO {user | role | PUBLIC} [, {user | role | PUBLIC}] ...
    [WITH ADMIN OPTION]
```

where:

system_priv
is a system privilege to be granted.

role
is a role to be granted

TO
identifies users or roles to which system privileges and roles are granted.

PUBLIC
grants system privileges or roles to all users.

WITH ADMIN OPTION
allows the grantee to grant the system privilege or role to other users or roles. If you grant a role with ADMIN OPTION, the grantee can also alter or drop the role.

PREREQUISITES:

To grant a system privilege, you must either have been granted the system privilege with the ADMIN OPTION or have been granted GRANT ANY PRIVILEGE system privilege.

To grant a role, you must either have been granted the role with the ADMIN OPTION or have been granted GRANT ANY ROLE system privilege or have created the role.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate both the label at which the system privilege or role was granted to you and the creation label of the grantee user or role.

SEE:
ALTER USER, CREATE USER, GRANT (Object Privileges), REVOKE Privileges and Roles)

GRANT command (Object Privileges)

PURPOSE:

To grant privileges for a particular object to users and roles. To grant system privileges and roles, use the GRANT command (System Privileges and Roles).

SYNTAX:

```
GRANT {object_priv | ALL [PRIVILEGES]} [ (column [, column] ...) ]
   [, {object_priv | ALL [PRIVILEGES]} [ (column [, column] ...) ] ] ...
    ON [schema.]object
    TO {user | role | PUBLIC} [, {user | role | PUBLIC}] ...
    [WITH GRANT OPTION]
```

where:

object_priv
is an object privilege to be granted. You can substitute any of these values:

* ALTER
* DELETE
* EXECUTE
* INDEX
* INSERT
* REFERENCES
* SELECT
* UPDATE

ALL PRIVILEGES
grants all the privileges for the object that you have been granted with the GRANT OPTION. The user who owns the schema containing an object automatically has all privileges on the object with the GRANT OPTION.

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

ON
identifies the object on which the privileges are granted. If you do not qualify object with schema, Oracle assumes the object is in your own schema. The object can be one of these types:

* table
* view
* sequence
* procedure, function, or package
* snapshots
* synonym for a table, view, sequence, snapshot, procedure, function, or package

TO

identifies users or roles to which the object privilege is granted.

PUBLIC

grants object privileges to all users.

WITH GRANT OPTION

allows the grantee to grant the object privileges to other users and roles. The grantee must be a user or PUBLIC, rather than a role.

PREREQUISITES:

The object must be in your own schema or you must have been granted the object privileges with the GRANT OPTION.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the label at which the object privilege was granted to you and the creation label of the grantee user or role.

SEE:

GRANT (System Privileges and Roles), REVOKE (Object Privileges)

INSERT command

PURPOSE:

To add rows to a table or to a view's base table.

SYNTAX:

```
INSERT INTO [schema.]{table | view}[@dblink]
    [ (column [, column] ...) ]
    {VALUES (expr [, expr] ...) | subquery}
```

where:

schema
: is the schema containing the table or view. If you omit schema, Oracle assumes the table or view is in your own schema.

table
view
: is name of the table into which rows are to be inserted. If you specify view, Oracle inserts rows into the view's base table.

dblink
: is a complete or partial name of a database link to a remote database where the table or view is located. You can only insert rows into a remote table or view if you are using Oracle with the distributed option.

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

column
: is a column of the table or view. In the inserted row, each column in this list is assigned a value from the VALUES clause or the subquery.

  If you omit one of the table's columns from this list, the column's value for the inserted row is the column's default value as specified when the table was created. If you omit the column list altogether, the VALUES clause or query must specify values for all columns in the table.

VALUES
: specifies a row of values to be inserted into the table or view. You must specify a value in the VALUES clause for each column in the column list.

subquery
: is a SELECT statement that returns rows that are inserted into the table. The select list of this subquery must have the same number of columns as the column list of the INSERT statement.

PREREQUISITES:

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

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

The INSERT ANY TABLE system privilege also allows you to insert rows into any table or any view's base table.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the creation label of the table or view:

* If the creation label of the table or view is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the creation label of your table or view is noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

DELETE, UPDATE

LOCK TABLE command

PURPOSE:

To lock one or more tables in a specified mode. This lock manually overrides automatic locking and permits or denies access to a table or view by other users for the duration of your operation.

SYNTAX:

```
LOCK TABLE [schema.]{table | view}[@dblink]
          [, [schema.]{table | view}[@dblink] ]...
     IN lockmode MODE
     [NOWAIT]
```

where:

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

table view
is the name of the table to be locked. If you specify view, Oracle locks the view's base tables.

dblink
is a database link to a remote Oracle7 database where the table or view is located. You can only lock tables and views on a remote database if you are using Oracle with the distributed option. All tables locked by a LOCK TABLE statement must be on the same database.

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

lockmode
is one of these:

* ROW SHARE
* ROW EXCLUSIVE
* SHARE UPDATE
* SHARE
* SHARE ROW EXCLUSIVE
* EXCLUSIVE

NOWAIT
specifies that Oracle returns control to you immediately if the specified table is already locked by another user. In this case, Oracle returns a message indicating that the table is already locked by another user.

If you omit this clause, Oracle waits until the table is available, locks it, and returns control to you.

PREREQUISITES:

The table or view must be in your own schema or you must have LOCK ANY TABLE system privilege or you must have any object privilege on the table or view.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the creation label of the table or view or you must have READUP system privilege.

SEE:

COMMIT, DELETE, INSERT, ROLLBACK, SAVEPOINT, UPDATE

NOAUDIT command (SQL Statements)

PURPOSE:

To stop auditing chosen by the AUDIT command (SQL Statements). To stop auditing chosen by the AUDIT command (Schema Objects), use the NOAUDIT command (Schema Objects).

SYNTAX:

```
NOAUDIT {statement_opt | system_priv}
     [, {statement_opt | system_priv} ] ...
    [BY user [, user] ...]
    [WHENEVER [NOT] SUCCESSFUL]
```

where:

statement_opt
is a statement option for which auditing is stopped.

system_priv
is a system privilege for which auditing is stopped.

BY
stops auditing only for SQL statements issued by specified users in their subsequent sessions. If you omit this clause, Oracle stops auditing for all users' statements.

WHENEVER SUCCESSFUL
stops auditing only for SQL statements that complete successfully.

NOT
stops auditing only for statements that result in Oracle errors.

If you omit the WHENEVER clause entirely, Oracle stops auditing for all statements, regardless of success or failure.

PREREQUISITES:

You must have AUDIT SYSTEM system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the label at which the auditing option was set or you must satisfy one of these criteria:

* If the auditing option was set at a label higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the auditing option was set at a label lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the auditing option was set at a label noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
AUDIT (SQL Statements), NOAUDIT (Schema Objects)

NOAUDIT command (Schema Objects)

PURPOSE:

To stop auditing chosen by the AUDIT command (Schema Objects). To stop auditing chosen by the AUDIT command (SQL Statements), use the NOAUDIT command (SQL Statements).

SYNTAX:

```
NOAUDIT object_opt [, object_opt] ...
    ON [schema.]object
    [WHENEVER [NOT] SUCCESSFUL]
```

where:

object_opt
stops auditing for particular operations on the object.

ON
identifies the object on which auditing is stopped. If you do not qualify object with schema, Oracle assumes the object is in your own schema.

WHENEVER SUCCESSFUL
stops auditing only for SQL statements that complete successfully.

NOT
option stops auditing only for statements that result in Oracle errors.

If you omit the WHENEVER clause entirely, Oracle stops auditing for all statements, regardless of success or failure.

PREREQUISITES:

The object on which you stop auditing must be in your own schema or you must have AUDIT ANY system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the label at which the auditing option was set or you must satisfy one of these criteria:

* If the auditing option was set at a label higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the auditing option was set at a label lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the auditing option was set at a label noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
AUDIT (Schema Objects), NOAUDIT (SQL Statements)

RECOVER clause

PURPOSE:

To perform media recovery.

SYNTAX:

```
RECOVER     [AUTOMATIC] [FROM 'location']
        { [DATABASE]    [ UNTIL CANCEL
                        | UNTIL TIME date
                        | UNTIL CHANGE integer
                        | USING BACKUP CONTROLFILE ]
         [PARALLEL integer | NOPARALLEL]
        | TABLESPACE tablespace [, tablespace] ...
          [PARALLEL integer | NOPARALLEL]
        | DATAFILE 'filename' [, 'filename'] ...
          [PARALLEL integer | NOPARALLEL]
        | LOGFILE 'filename'
        | CONTINUE [DEFAULT]
        | CANCEL }
```

where:

AUTOMATIC
automatically generates the names of the redo log files to apply during media recovery. If you omit this option, Oracle prompts you for names of redo log files and you must specify them by issuing ALTER DATABASE statements with the LOGFILE parameter.

FROM
specifies the location from which the archived redo log file group is read. The value of this parameter must be a fully-specified file location following the conventions of your operating system. If you omit this parameter, Oracle assumes the archived redo log file group is in the location specified by the initialization parameter LOG_ARCHIVE_DEST

DATABASE
recovers the entire database. This is the default option. You can only use this option when the database is closed.

PARALLEL
specifies the number of recovery processes used to apply redo entries to datafiles. The specified value overrides the RECOVERY_PARALLELISM initialization parameter. Specified values must be positive integers greater than 1.

NOPARALLEL
specifies that recovery should proceed using only one recovery process.

UNTIL CANCEL
performs cancel-based recovery. This option recovers the database

until you cancel recovery by issuing an ALTER DATABASE statement with a RECOVER clause containing the CANCEL option.

UNTIL TIME
performs 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'.

UNTIL CHANGE
performs change-based recovery. This parameter recovers the database to a transaction consistent state immediately prior to the system change number (SCN) specified by integer.

USING BACKUP CONTROLFILE
specifies that a backup control file is being used instead of the current control file.

TABLESPACE
recovers only the specified tablespaces. You can use this option if the database is open or closed, provided the tablespaces to be recovered are not being used.

DATAFILE
recovers the specified data files. You can use this option when the database is open or closed, provided the data files to be recovered are not being used.

LOGFILE
continues media recovery by applying the specified the redo log file.

CONTINUE
continues multi-instance recovery after it has been interrupted to disable a thread.

CONTINUE DEFAULT
continues recovery by applying the redo log file that Oracle has automatically generated.

CANCEL
terminates cancel-based recovery.

PREREQUISITES:

The RECOVER clause must appear in a ALTER DATABASE statement. You must have the privileges necessary to issue this statement. For information on these privileges, see the ALTER DATABASE command.

You must also have the OSDBA role enabled. You cannot be connected to Oracle through the multi-threaded server architecture. Your instance must have the database mounted in exclusive mode.

SEE:
ALTER DATABASE

RENAME command

PURPOSE:

To rename a table, view, sequence, or private synonym.

SYNTAX:

```
RENAME old TO new
```

where:

old
: is the current name of an existing table, view, sequence, or private synonym.

new
: is the new name to be given to the existing object.

PREREQUISITES:

The object must be in your own schema.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the object's creation label or you must satisfy one of these criteria:

* If the object's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the object's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the object's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, CREATE VIEW

REVOKE command (System Privileges and Roles)

PURPOSE:

To revoke system privileges and roles from users and roles. To revoke object privileges from users and roles, use the REVOKE command (Object Privileges).

SYNTAX:

```
REVOKE {system_priv | role} [, {system_priv | role}] ...
    FROM {user | role | PUBLIC} [, {user | role | PUBLIC}] ...
```

where:

system_priv
is a system privilege to be revoked.

role
is a role to be revoked.

FROM
identifies users and roles from which the system privileges or roles are revoked.

PUBLIC
revokes the system privilege or role from all users.

PREREQUISITES:

You must have been granted the system privilege or role with the ADMIN OPTION. Also, you can revoke any role if you have the GRANT ANY ROLE system privilege.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the label at which the system privilege or role was granted or you must satisfy one of these criteria:

* If the label at which the system privilege or role was granted is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the label at which the system privilege or role was granted is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the label at which the system privilege or role is noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:
GRANT (System Privileges and Roles), REVOKE (Object Privileges)

REVOKE command (Object Privileges)

PURPOSE:

To revoke object privileges for a particular object from users and roles. To revoke system privileges or roles, use the REVOKE command (System Privileges and Roles).

SYNTAX:

```
REVOKE {object_priv | ALL [PRIVILEGES]}
    [, {object_priv | ALL [PRIVILEGES]} ] ...
    ON [schema.]object
    FROM {user | role | PUBLIC} [, {user | role | PUBLIC}] ...
    [CASCADE CONSTRAINTS]
```

where:

object_priv
: is an object privilege to be revoked. You can substitute any of these values:

  * ALTER
  * DELETE
  * EXECUTE
  * INDEX
  * INSERT
  * REFERENCES
  * SELECT
  * UPDATE

ALL PRIVILEGES
: revokes all object privileges that you have granted to the revokee.

ON
: identifies the object on which the object privileges are revoked. This object can be one of these types:

  * table
  * view
  * sequence
  * procedure, stored function, or package
  * snapshot
  * synonym for a table, view, sequence, procedure, stored function, package, or snapshot

  If you do not qualify object with schema, Oracle assumes the object is in your own schema.

FROM
: identifies users and roles from which the object privileges are revoked.

  PUBLIC
  : revokes object privileges from all users.

CASCADE CONSTRAINTS

drops any referential integrity constraints that the revokee has defined using REFERENCES privilege that you are now revoking. You must specify this option along with the REFERENCES privilege or the ALL PRIVILEGES option if the revokee has exercised the REFERENCES privilege to define a referential integrity constraint.

PREREQUISITES:

You must have previously granted the object privileges to each user and role.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the label at which you granted the object privilege or you must satisfy one of these criteria:

* If the label at which you granted the object privilege is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the label at which you granted the object privilege is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the label at which you granted the object privilege is noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

GRANT (Object Privileges), REVOKE (System Privileges and Roles)

ROLLBACK command

PURPOSE:

To undo work done in the current transaction.

You can also use this command to manually undo the work done by an in-doubt distributed transaction.

SYNTAX:

```
ROLLBACK [WORK]
    [ TO [SAVEPOINT] savepoint
    | FORCE 'text' ]
```

where:

WORK
is optional and is provided for ANSI compatibility.

TO
rolls back the current transaction to the specified savepoint. If you omit this clause, the ROLLBACK statement rolls back the entire transaction.

FORCE
manually rolls back an in-doubt distributed transaction. The transaction is identified by the 'text' containing its local or global transaction ID. To find the IDs of such transactions, query the data dictionary view DBA_2PC_PENDING.

ROLLBACK statements with the FORCE clause are not supported in PL/SQL.

PREREQUISITES:

To roll back your current transaction, no privileges are necessary.

To manually roll back an in-doubt distributed transaction that you originally committed, you must have FORCE TRANSACTION system privilege. To manually roll back an in-doubt distributed transaction originally committed by another user, you must have FORCE ANY TRANSACTION system privilege.

SEE:
COMMIT, SAVEPOINT, SET TRANSACTION

SAVEPOINT command

PURPOSE:

To identify a point in a transaction to which you can later roll back.

SYNTAX:

```
SAVEPOINT savepoint
```

where:

savepoint
is the name of the savepoint to be created.

PREREQUISITES:

None.

SEE:
COMMIT, ROLLBACK, SET TRANSACTION

SELECT command

PURPOSE:

To retrieve data from one or more tables, views, or snapshots.

SYNTAX:

```
SELECT [DISTINCT | ALL] { *
                        | { [schema.]{table | view | snapshot}.*
                          | expr }  [ [AS] c_alias ]
                       [, { [schema.]{table | view | snapshot}.*
                          | expr } [ [AS] c_alias ]  ] ... }
    FROM [schema.]{table | view | snapshot}[@dblink] [t_alias]
      [, [schema.]{table | view | snapshot}[@dblink] [t_alias] ] ...
    [WHERE condition ]
    [ [START WITH condition] CONNECT BY condition]
    [GROUP BY expr [, expr] ... [HAVING condition] ]
    [{UNION | UNION ALL | INTERSECT | MINUS} SELECT command ]
    [ORDER BY {expr|position} [ASC | DESC]
           [, {expr|position} [ASC | DESC]] ...]
    [FOR UPDATE [OF [[schema.]{table | view}.]column
                 [, [[schema.]{table | view}.]column] ...] [NOWAIT] ]
```

where:

DISTINCT
: returns only one copy of each set of duplicate rows selected. Duplicate rows are those with matching values for each expression in the select list.

ALL
: returns all rows selected, including all copies of duplicates.

  The default is ALL.

*
: selects all columns from all tables, views, or snapshots listed in the FROM clause.

table.*
view.*
snapshot.*
: selects all columns from the specified table, view, or snapshot. You can use the schema qualifier to select from a table, view, or snapshot in a schema other than your own.

  If you are using Trusted Oracle, the * does not select the ROWLABEL column. To select this column, you must explicitly specify it in the select list.

expr
: selects an expression, usually based on columns values, from one of the tables, views, or snapshots in the FROM clause. A column name

in this list can only contain be qualified with schema if the table, view, or snapshot containing the column is qualified with schema in the FROM clause.

c_alias
: provides a different name for the column expression and causes the alias to be used in the column heading. A column alias does not affect the actual name of the column. Column aliases can be referenced in the ORDER BY clause but in no other clauses in a statement.

schema
: is the schema containing the selected table, view, or snapshot. If you omit schema, Oracle assumes the table, view, or snapshot is in your own schema.

table
view
snapshot
: is the name of a table, view, or snapshot from which data is selected.

dblink
: is complete or partial name for a database link to a remote database where the table, view, or snapshot is located. Note that this database need not be an Oracle7 database.

  If you omit dblink, Oracle assumes that the table, view, or snapshot is on the local database.

t_alias
: provides a different name for the table, view, or snapshot for the purpose of evaluating the query and is most often used in a correlated query. Other references to the table, view, or snapshot throughout the query must refer to the alias.

WHERE
: restricts the rows selected to those for which the condition is TRUE. If you omit this clause, Oracle returns all rows from the tables, views, or snapshots in the FROM clause.

START WITH
CONNECT BY
: returns rows in a hierarchical order.

GROUP BY
: groups the selected rows based on the value of expr for each row and returns a single row of summary information for each group.

HAVING
: restricts the groups of rows returned to those groups for which the specified condition is TRUE. If you omit this clause, Oracle returns summary rows for all groups.

UNION

UNION ALL
INTERSECT
MINUS
combines the rows returned by two SELECT statement using a set operation.

AS
can optionally precede a column alias. To comply with the ANSI SQL92 standard, column aliases must be preceded by the AS keyword.

ORDER BY
orders rows returned by the statement.

expr
orders rows based on their value for expr. The expression is based on columns in the select list or columns in the tables, views, or snapshots in the FROM clause.

position
orders rows based on their value for the expression in this position of the select list.

ASC
DESC
specifies either ascending or descending order. ASC is the default.

The ORDER BY clause can reference column aliases defined in the SELECT list.

FOR UPDATE
locks the selected rows.

NOWAIT
returns control to you if the SELECT statement attempts to lock a row that is locked by another user. If you omit this clause, Oracle waits until the row is available and then returns the results of the SELECT statement.

PREREQUISITES:

For you to select data from a table or snapshot, the table or snapshot must be in your own schema or you must have SELECT privilege on the table or snapshot.

For you to select rows from the base tables of a view, the owner of the schema containing the view must have SELECT privilege on the base tables. Also, if the view is in a schema other than your own, you must have SELECT privilege on the view.

The SELECT ANY TABLE system privilege also allows you to select data from any table or any snapshot or any view's base table.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the creation label of each queried table, view, or snapshot or you must have READUP system privileges.

SEE:

DELETE, UPDATE

SET ROLE command

PURPOSE:

To enable and disable roles for your current session.

SYNTAX:

```
SET ROLE     { role [IDENTIFIED BY password]
         [, role [IDENTIFIED BY password] ] ...
          | ALL [EXCEPT role [, role] ...]
          | NONE }
```

where:

role
: is a role to be enabled for the current session. Any roles not listed are disabled for the current session.

password
: is the password for a role. If the role has a password, you must specify the password to enable the role.

ALL EXCEPT
: enables all roles granted to you for the current session, except those listed in the EXCEPT clause. Roles listed in the EXCEPT clause must be roles granted directly to you; they cannot be roles granted to you through other roles. You cannot use this option to enable roles with passwords that have been granted directly to you.

  If you list a role in the EXCEPT clause that has been granted to you both directly and through another role, the role is still enabled by virtue of your enabling the role to which it has been granted.

NONE
: disables all roles for the current session.

PREREQUISITES:

You also must already have been granted the roles that you name in this statement.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must dominate the label at which these roles were granted to you.

SEE:

ALTER USER, CREATE ROLE

SET TRANSACTION command

PURPOSE:

To perform one of these operations on your current transaction:

* establish your current transaction as either a read-only or a read-write transaction
* assign your current transaction to a specified rollback segment

SYNTAX:

```
SET TRANSACTION
    { READ ONLY
    | READ WRITE
    | USE ROLLBACK SEGMENT rollback_segment }
```

where:

READ ONLY
establishes the current transaction as a read-only transaction.

READ WRITE
establishes the current transaction as a read-write transaction.

USE ROLLBACK SEGMENT
assigns the current transaction to the specified rollback segment. This option also establishes the transaction as a read-write transaction.

You cannot use the READ ONLY option and the USE ROLLBACK SEGMENT clause in a single SET TRANSACTION statement or in different statements in the same transaction. Read-only transactions do not generate rollback information and therefore are not assigned rollback segments.

PREREQUISITES:

A SET TRANSACTION statement must be the first statement in your transaction. However, every transaction need not begin with a SET TRANSACTION statement.

SEE:
COMMIT, ROLLBACK, SAVEPOINT

STORAGE clause

PURPOSE:

To specify storage characteristics for tables, indexes, clusters, and rollback segments, and the default storage characteristics for tablespaces.

SYNTAX:

```
STORAGE (    [INITIAL             integer [K|M] ]
             [NEXT              integer [K|M] ]
             [PCTINCREASE         integer]
             [MINEXTENTS         integer]
             [MAXEXTENTS          integer]
             [OPTIMAL         {integer  [K|M] | NULL}]
             [FREELIST GROUPS     integer]
             [FREELISTS         integer] )
```

where:

INITIAL
specifies the size in bytes of the object's first extent. Oracle allocates space for this extent when you create the object. You can also use K or M to specify this size in kilobytes or megabytes. The default value is the size of 5 data blocks. The minimum value is the size of 2 data blocks. The maximum value varies depending on your operating system. Oracle rounds values up to the next multiple of the data block size.

NEXT
specifies the size in bytes of the next extent to be allocated to the object. You can also use K or M to specify the size in kilobytes or megabytes. The default value is the size of 5 data blocks. The minimum value is the size of 1 data block. The maximum value varies depending on your operating system. Oracle rounds values up to the next multiple of the data block size.

PCTINCREASE
specifies the percent by which each extent after the second grows over the previous extent. The default value is 50, meaning that each subsequent extent is 50% larger than the preceding extent. The minimum value is 0, meaning all extents after the first are the same size. The maximum value varies depending on your operating system.

You cannot specify PCTINCREASE for rollback segments. Rollback segments always have a PCTINCREASE value of 0.

Oracle rounds the calculated size of each new extent up to the next multiple of the data block size.

MINEXTENTS
specifies the total number of extents allocated when the segment is created. This parameter allows you to allocate a large amount of

space when you create an object, even if the space available is not contiguous. The default and minimum value is 1, meaning that Oracle only allocates the initial extent, except for rollback segments for which the default and minimum value is 2. The maximum value varies depending on your operating system.

If the MINEXTENTS value is greater than 1, then Oracle calculates the size of subsequent extents based on the values of the INITIAL, NEXT, and PCTINCREASE parameters.

MAXEXTENTS
specifies the total number of extents, including the first, that Oracle can allocate for the object. The minimum value is 1. The default and maximum values vary depending your data block size.

OPTIMAL
specifies an optimal size in bytes for a rollback segment. You can also use K or M to specify this size in kilobytes or megabytes. Oracle tries to maintain this size for the rollback segment by dynamically deallocating extents when their data is no longer needed for active transactions. Oracle deallocates as many extents as possible without reducing the total size of the rollback segment below the OPTIMAL value. This parameter is only for rollback segments and not for other objects.

NULL
specifies no optimal size for the rollback segment, meaning that Oracle never deallocates the rollback segment's extents. This is the default behavior.

The value of this parameter cannot be less than the space initially allocated for the rollback segment specified by the MINEXTENTS, INITIAL, NEXT, and PCTINCREASE parameters. The maximum value varies depending on your operating system. Oracle rounds values to the next multiple of the data block size.

FREELIST GROUPS
specifies the number of groups of free lists for a table, cluster, or index. The default and minimum value for this parameter is 1. You should only use this parameter if you are using Oracle with the Parallel Server option in parallel mode.

FREELISTS
specifies the number of free lists for each of the free list groups for the table, cluster, or index. The default and minimum value for this parameter is 1, meaning that each free list group contains one free list. The maximum value of this parameter depends on the data block size. If you specify a FREELISTS value that is too large, Oracle returns an error indicating the maximum value.

You can only specify the FREELISTS parameter in CREATE TABLE, CREATE CLUSTER, and CREATE INDEX statements. You can only specify the FREELIST GROUPS parameter in CREATE TABLE and CREATE CLUSTER statements.

PREREQUISITES:

The STORAGE clause can appear in commands that create or alter any of these objects:

* clusters
* indexes
* rollback segments
* snapshots
* snapshot logs
* tables
* tablespaces

To change the value of a STORAGE parameter, you must have the privileges necessary to issue one of these commands.

SEE:
CREATE CLUSTER, CREATE INDEX, CREATE ROLLBACK SEGMENT, CREATE TABLE, CREATE TABLESPACE

TRUNCATE command

PURPOSE:

To remove all rows from a table or cluster.

SYNTAX:

```
TRUNCATE {TABLE [schema.]table | CLUSTER [schema.]cluster}
    [ {DROP | REUSE} STORAGE]
```

where:

TABLE
specifies the schema and name of the table to be truncated. If you omit schema, Oracle assumes the table is in your own schema. This table cannot be part of a cluster.

When you truncate a table, Oracle also automatically deletes all data in the table's indexes.

CLUSTER
specifies the schema and name of the cluster to be truncated. If you omit schema, Oracle assumes the cluster is in your own schema. You can only truncate an indexed cluster, not a hash cluster.

When you truncate a cluster, Oracle also automatically deletes all data in the cluster's tables' indexes.

DROP STORAGE
deallocates the space from the deleted rows from the table or cluster. This space can subsequently be used by other objects in the tablespace.

REUSE STORAGE
leaves the space from the deleted rows allocated to the table or cluster. This space can be subsequently used only by new data in the table or cluster resulting from inserts or updates.

The DROP STORAGE or REUSE STORAGE option that you choose also applies to the space freed by the data deleted from associated indexes.

If you omit both the REUSE STORAGE and DROP STORAGE options, Oracle uses the DROP STORAGE option by default.

PREREQUISITES:

The table or cluster must be in your schema or you must have DELETE ANY TABLE system privilege.

If you are using Trusted Oracle, your DBMS label must match the creation label of the table or cluster or you must satisfy one of these criteria:

* If the creation label of the table or cluster is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the creation label of the table or cluster is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the creation label of the table or cluster is noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

DELETE, DROP CLUSTER, DROP TABLE

UPDATE command

PURPOSE:

To change existing values in a table or in a view's base table.

SYNTAX:

```
UPDATE [schema.]{table | view}[@dblink] [alias]
    SET { (column [, column] ...) = (subquery)
        |  column = { expr | (subquery) } }
     [,    { (column [, column] ...) = (subquery)
        |  column = { expr | (subquery) } } ] ...
    [WHERE condition]
```

where:

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

table
view
is the name of the table to be updated. If you specify view, Oracle updates the view's base table.

dblink
is a complete or partial name of a database link to a remote database where the table or view is located. You can only use a database link to update a remote table or view if you are using Oracle with the distributed option.

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

alias
is used to relabel the name of the reference in the other clauses of the command.

column
is the name of a column of the table or view that is to be updated. If you omit a column of the table from the SET clause, that column's value remains unchanged.

expr
is the new value assigned to the corresponding column. This expression can contain host variables and optional indicator variables.

subquery
is a SELECT statement that returns new values that are assigned to the corresponding columns.

WHERE
restricts the rows updated to those for which the specified

condition is TRUE. If you omit this clause, Oracle updates all rows in the table or view.

PREREQUISITES:

For you to update values in a table, the table must be in your own schema or you must have UPDATE privilege on the table.

For you to update values in the base table of a view, the owner of the schema containing the view must have UPDATE privilege on the base table. Also, if the view is in a schema other than your own, you must have UPDATE privilege on the view.

The UPDATE ANY TABLE system privilege also allows you to update values in any table or any view's base table.

If you are using Trusted Oracle in DBMS MAC mode, your DBMS label must match the creation label of the table or view:

* If the creation label of the table or view is higher than your DBMS label, you must have READUP and WRITEUP system privileges
* If the creation label of the table or view is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the creation label of your table or view is noncomparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

SEE:

DELETE, INSERT

Oracle Datatypes

Each literal or column value manipulated by Oracle has a datatype. A value's datatype associates a fixed set of properties with the value. These properties cause Oracle to treat values of one datatype differently from values of another. For example, you can add values of NUMBER datatype, but not values of RAW datatype.

When you create a table or cluster, you must specify an internal datatype for each of its columns. When you create a procedure or stored function, you must specify an internal datatype for each of its arguments. These datatypes define the domain of values that each column can contain or each argument can have. For example, DATE columns cannot accept the value February 29 (except for a leap year) or the values 2 or 'SHOE'. Each value subsequently placed in a column assumes the column's datatype. For example, if you insert '01-JAN-92' into a DATE column, Oracle treats the '01-JAN-92' character string as a DATE value after verifying that it translates to a valid date.

The following table summarizes Oracle internal datatypes. The rest of this section describes these datatypes in detail.

Note: The Oracle Precompilers recognize other datatypes in embedded SQL programs. These datatypes are called external datatypes and are associated with host variables. Do not confuse the internal datatypes with external datatypes.

Code	Internal Datatype	Description
1	VARCHAR2(size)	Variable length character string having maximum length size bytes. Maximum size is 2000. You must specify size for a VARCHAR2.
2	NUMBER(p,s)	Number having precision p and scale s. The precision p can range from 1 to 38. The scale s is primarily for values returned by the ROWID pseudocolumn.
96	CHAR(size)	Fixed length character data of length size bytes. Maximum size is 255. Default size is 1 byte.
106	MLSLABEL	Binary format of an operating system label. This datatype is used primarily with Trusted Oracle.

The codes listed for the datatypes are used internally by Oracle. The datatype code of a column is returned when you use the DUMP function. Note that these codes are only for internal use and may

not necessarily match the datatype codes of expressions of the same datatype.

SEE:

ANSI, DB2, and SQL/DS Datatypes, Character Datatypes, Data Conversion, Datatype Comparison Rules, DATE Datatype, LONG Datatype, MLSLABEL Datatype, NUMBER Datatype, RAW and LONG RAW Datatypes, ROWID Datatype

Character Datatypes

Character datatypes are used to manipulate words and free-form text. These datatypes are used to store character (alphanumeric) data in the database character set. They are less restrictive than other datatypes and consequently have fewer properties. For example, character columns can store all alphanumeric values, but NUMBER columns can only store numeric values.

Character data is stored in strings with byte values corresponding to the character set, such as 7-bit ASCII or EBCDIC Code Page 500, specified when the database was created. Oracle supports both single-byte and multi-byte character sets.

These datatypes are used for character data:

* CHAR
* VARCHAR2

The character datatypes in Oracle7 are different from those in Oracle Version 6.

SEE:
CHAR Datatype, VARCHAR Datatype, VARCHAR2 Datatype

CHAR Datatype

The CHAR datatype specifies a fixed length character string. When you create a table with a CHAR column, you can supply the column length in bytes. Oracle subsequently ensures that all values stored in that column have this length. If you insert a value that is shorter than the column length, Oracle blank-pads the value to column length. If you try to insert a value that is too long for the column, Oracle returns an error.

The default length for a CHAR column is 1 byte. The maximum length of CHAR data is 255 bytes. Oracle compares CHAR values using blank-padded comparison semantics.

SEE:
Character Datatypes, VARCHAR Datatype, VARCHAR2 Datatype

VARCHAR2 Datatype

The VARCHAR2 datatype specifies a variable length character string. When you create a VARCHAR2 column, you can supply the maximum number of bytes of data that it can hold. Oracle subsequently stores each value in the column exactly as you specify it, provided it does not exceed the column's maximum length. If you try to insert a value that exceeds this length, Oracle returns an error.

You must specify a maximum length for a VARCHAR2 column. The maximum length of VARCHAR2 data is 2000 bytes. Oracle compares VARCHAR2 values using non-padded comparison semantics.

SEE:

Character Datatypes, CHAR Datatype, VARCHAR Datatype

VARCHAR Datatype

The VARCHAR datatype is currently synonymous with the VARCHAR2 datatype. Oracle Corporation recommends that you use VARCHAR2 rather than VARCHAR. In a future version of Oracle, VARCHAR might be a separate datatype used for variable length character strings compared with different comparison semantics.

SEE:

Character Datatypes, CHAR Datatype, VARCHAR2 Datatype

NUMBER Datatype

The NUMBER datatype is used to store zero, positive and negative fixed and floating point numbers with magnitudes between 1.0 x 10
digits of precision. If you specify an arithmetic expression whose value has a magnitude greater than or equal to 1.0 x 10
returns an error. You can specify a fixed point number datatype with this syntax:

```
NUMBER(p,s)
```

where:

p
: is the precision, or the total number of digits. Oracle guarantees the portability of numbers with precision ranging from 1 to 38.

s
: is the scale, or the number of digits to the right of the decimal point. The scale can range from -84 to 127.

You can also use one of these alternate forms:

NUMBER(p)
: is a fixed point number with precision p and scale 0.

NUMBER
: is a floating point number with precision 38. Note that a scale value is not applicable for floating point numbers.

SCALE AND PRECISION
: Specify the scale and precision of a number column for extra integrity checking on input. Specifying scale and precision does not force all values to a fixed length. If a value exceeds the precision, Oracle returns an error. If a value exceeds the scale, Oracle rounds it.

  These examples show how Oracle stores data using different precisions and scales.

Actual Data	Specified as	Stored as
7456123.89	NUMBER	7456123.89
7456123.89	NUMBER (9)	7456124
7456123.89	NUMBER (9,2)	7456123.89
7456123.89	NUMBER (9,1)	7456123.9
7456123.8	NUMBER (6)	exceeds precision
7456123.8	NUMBER (15,1)	7456123.8
7456123.89	NUMBER (7,-2)	7456100
7456123.89	NUMBER(7,2)	exceeds precision

NEGATIVE SCALE
: If the scale is negative, the actual data is rounded to the specified number of places to the left of the decimal point. For

example, a specification of (10,-2) means to round to hundreds.

SCALE GREATER THAN PRECISION

You can specify a scale that is greater than precision, although it is uncommon. In this case, the precision specifies the maximum number of digits to the right of the decimal point. As with all number datatypes, if the value exceeds the precision, Oracle returns an error. If the value exceeds the scale, Oracle rounds the value. For example, a column defined as NUMBER(4,5) requires a zero for the first digit after the decimal point and rounds all values past the fifth digit after the decimal point. The following examples show the effects of a scale greater than precision:

Actual Data	Specified as	Stored as
.01234	NUMBER(4,5)	.01234
.00012	NUMBER(4,5)	.00012
.000127	NUMBER(4,5)	.00013
.0000012	NUMBER(2,7)	.0000012
.00000123	NUMBER(2,7)	.0000012

FLOATING POINT NUMBERS

Oracle also allows you to specify floating point numbers. A floating point value either can have a decimal point anywhere from the first to the last digit or can omit the decimal point altogether. A scale value is not applicable to floating point numbers because there is no restriction on the number of digits that can appear after the decimal point.

You can specify floating point numbers with the appropriate forms of the NUMBER datatype discussed in the section on page -. Oracle also supports the ANSI datatype FLOAT. You can specify this datatype using one of these syntactic forms:

FLOAT

specifies a floating point number with decimal precision 38, or a binary precision of 126.

FLOAT(b)

specifies a floating point number with binary precision b. The precision b can range from 1 to 126.

To convert from binary to decimal precision, multiply b by 0.30103. To convert from decimal to binary precision, multiply the decimal precision by 3.32193. The maximum of 126 digits of binary precision is roughly equivalent to 38 digits of decimal precision.

SEE:

DATE Datatype, LONG Datatype

LONG Datatype

LONG columns store variable length character strings containing up to 2 gigabytes, or 2
characteristics of VARCHAR2 columns. You can use LONG columns to store long text strings. Oracle uses LONG columns in the data dictionary to store the text of view definitions. The length of LONG values may also be limited by the memory available on your computer.

You can reference LONG columns in SQL statements in these places:

* SELECT lists
* SET clauses of UPDATE statements
* VALUES clauses of INSERT statements

The use of LONG values are subject to some restrictions:

* A table cannot contain more than one LONG column.
* LONG columns cannot appear in integrity constraints (except for NULL and NOT NULL constraints).
* LONG columns cannot be indexed.
* A procedure or stored function cannot accept a LONG argument.
* A stored function cannot return a LONG value.
* Within a single SQL statement, all LONG columns, sequences, updated tables, and locked tables must be located on the same database.

Also, LONG columns cannot appear in certain parts of SQL statements:

* WHERE, GROUP BY, ORDER BY, or CONNECT BY clauses or with the DISTINCT operator in SELECT statements
* SQL Functions (such as SUBSTR or INSTR)
* expressions or conditions
* select lists of queries containing GROUP BY clauses
* select lists of subqueries or queries combined by set operators
* select lists of CREATE TABLE AS SELECT statements

You can use the Oracle Call Interfaces to retrieve a portion of a LONG value from the database.

SEE:

Character Datatypes, VARCHAR2 Datatype

DATE Datatype

The DATE datatype is used to store date and time information. Although date and time information can be represented in both CHAR and NUMBER datatypes, the DATE datatype has special associated properties.

For each DATE value the following information is stored:

* century
* year
* month
* day
* hour
* minute
* second

You cannot specify a date literal. To specify a date value, you must convert a character or numeric value to a data value with the TO_DATE function. Oracle automatically converts character values that are in the default date format into date values when they are used in date expressions. The default date format is specified by the initialization parameter NLS_DATE_FORMAT and is a string such as 'DD-MON-YY', This example date format includes two-digit number for the day of the month, an abbreviation of the month name, and the last two digits of the year.

If you specify a date value without a time component, the default time is 12:00:00a.m. (midnight). If you specify a date value without a date, the default date is the first day of the current month.

The date function SYSDATE returns the current date and time. For information on the SYSDATE and TO_DATE functions and the default date format, see Operators, Functions, Expressions, and Conditions.

DATE ARITHMETIC:

You can add and subtract number constants as well as other dates from dates. Oracle interprets number constants in arithmetic date expressions as numbers of days. For example, SYSDATE + 1 is tomorrow. SYSDATE - 7 is one week ago. SYSDATE + (10/1440) is ten minutes from now. Subtracting the HIREDATE column of the EMP table from SYSDATE returns the number of days since each employee was hired. You cannot multiply or divide DATE values.

Oracle provides functions for many of the common date operations. For example, the ADD_MONTHS function allows you to add or subtract months from a date. The MONTHS_BETWEEN function returns the number of months between two dates. The fractional portion of the result represents that portion of a 31-day month.

Because each date contains a time component, most results of date operations include a fraction. This fraction means a portion of one day. For example, 1.5 days is 36 hours.

USING JULIAN DATES:

A Julian date is the number of days since Jan 1, 4712 BC. Julian dates allow continuous dating from a common reference. You can use the date format model J with date functions TO_DATE and TO_CHAR to convert between Oracle DATE values and their Julian equivalents.

EXAMPLE:

This statement returns the Julian equivalent of January 1, 1992:

```
SELECT TO_CHAR(TO_DATE('01-01-1992', 'MM-DD-YYYY'),'J')
    FROM DUAL
TO_CHAR(TO_DATE('01-01-1992','MM-DD-YYYY),'J')
-----------------------------------------------
2448623
```

SEE:

Character Datatypes, NUMBER Datatype

RAW and LONG RAW Datatypes

The RAW and LONG RAW datatypes are used for byte-oriented data (for example, binary data or byte strings) to store character strings, floating point data, and binary data such as graphics images and digitized sound. Oracle returns RAW values as hexadecimal character values. RAW data can only be stored and retrieved. You cannot perform string manipulation on RAW data.

RAW is equivalent to VARCHAR2 and LONG RAW to LONG except that there is no conversion between database and session character set. While CHAR, VARCHAR2, and LONG data is automatically converted between the database character set and the user-side character set for the session, there is no such conversion for RAW and LONG RAW data. LONG RAW data is subject to the same restrictions as LONG data.

SEE:

CHAR Datatype, LONG Datatype, VARCHAR2 Datatype

ROWID Datatype

Each row in the database has an address. You can examine a row's address by querying the pseudocolumn ROWID. Values of this pseudocolumn are hexadecimal strings representing the address of each row. These string have the datatype ROWID. For more information on the ROWID pseudocolumn, see Pseudocolumns. You can also create tables and clusters that contain actual columns having the ROWID datatype. Oracle does not guarantee that the values of such columns are valid ROWIDs.

Character values representing ROWIDs:

```
block.row.file
```

where:

block
: is a hexadecimal string identifying the data block of the data file containing the row. The length of this string may vary depending on your operating system.

row
: is a four-digit hexadecimal string identifying the row in the data block. The first row in the block has the number 0.

file
: is a hexadecimal string identifying the database file containing the row. The first data file has the number 1. The length of this string may vary depending on your operating system.

EXAMPLE:

Consider this ROWID value:

```
0000000F.0000.0002
```

The row corresponding to this ROWID is the first row (0000) in the fifteenth data block (0000000F) of the second data file (0002).

SEE:

Pseudocolumns, ROWID

MLSLABEL Datatype

The MLSLABEL datatype is used to store the binary format a label used on a secure operating system. Labels are used by Trusted Oracle to mediate access to information. You can also define columns with this datatype if you are the standard Oracle Server.

ANSI, DB2, and SQL/DS Datatypes

SQL commands that create tables and clusters also accept both ANSI datatypes and datatypes from IBM's products SQL/DS and DB2. Oracle creates columns with Oracle datatypes based on the conversions defined in the following two tables.

ANSI SQL Datatype	Oracle Datatype
CHARACTER(n), CHAR(n)	CHAR(n)
CHARACTER VARYING(n), CHAR VARYING(n)	VARCHAR(n)
NUMERIC(p,s), DECIMAL(p,s), DEC(p,s)[1]	NUMBER(p,s)
INTEGER, INT, SMALLINT	NUMBER(38)
FLOAT(b)[2], DOUBLE PRECISION[3], REAL[4]	NUMBER

SQL/DS or DB2 Datatype	Oracle Datatype
CHARACTER(n)	CHAR(n)
VARCHAR(n)	VARCHAR(n)
LONG VARCHAR	LONG
DECIMAL(p,s)[1]	NUMBER(p,s)
INTEGER, SMALLINT	NUMBER(38)
FLOAT(b)[2]	NUMBER

[1] The NUMERIC, DECIMAL, and DEC datatypes can specify only fixed point numbers. For these datatypes, s defaults to 0.

[2] The FLOAT datatype is a floating point number with a binary precision b. This default precision for this datatype is 126 binary, or 38 decimal.

[3] The DOUBLE PRECISION datatype is a floating point number with binary precision 126.

[4] The REAL datatype is a floating point number with a binary precision of 63, or 18 decimal.

Do not define columns with these SQL/DS and DB2 datatypes because they have no corresponding Oracle datatype:

* GRAPHIC
* LONG VARGRAPHIC
* VARGRAPHIC
* TIME
* TIMESTAMP

Note that data of type TIME and TIMESTAMP can also be expressed as Oracle DATE data.

Datatype Comparison Rules

NUMBER VALUES:

A larger value is considered greater than a smaller one. All negative numbers are less than all positive numbers. Thus, -1 is less than 100; -100 is less than -1.

DATE VALUES:

A later date is considered greater than an earlier one. For example, the date equivalent of '29-MAR-1991' is less than that of '05-JAN-1992' and '05-JAN-1992 1:35pm' is greater than '05-JAN-1992 10:09am'.

CHARACTER STRING VALUES:

Character values are compared using one of these comparison rules:

* blank-padded comparison semantics
* non-padded comparison semantics

The following sections explain these comparison semantics. The results of comparing two character values using different comparison semantics may be different. The following table shows the results of comparing five pairs of character values using each comparison semantic. The last comparison in the table illustrates the differences between the blank-padded and non-padded comparison semantics.

Blank-Padded	Non-Padded
'ab' > 'aa'	'ab' > 'aa'
'ab' > 'a '	'ab' > 'a '
'ab' > 'a'	'ab' > 'a'
'ab' = 'ab'	'ab' = 'ab'
'a ' = 'a'	'a ' > 'a'

BLANK-PADDED COMPARISON SEMANTICS:

If the two values have different lengths, Oracle first adds blanks to the end of the shorter one so their lengths are equal. Oracle then compares the values character by character up to the first character that differs. The value with the greater character in the first differing position is considered greater. If two values have no differing characters, then they are considered equal. This rule means that two values are equal if they differ only in the number of trailing blanks. Oracle uses blank-padded comparison semantics only when both values in the comparison have the datatype CHAR or are text literals.

NON-PADDED COMPARISON SEMANTICS:

Oracle compares two values character by character up to the first character that differs. The value with the greater character in that position is considered greater. If two values of different length are identical up to the end of the shorter one, the longer value is considered greater. If two values of equal length have no differing characters, then the values are considered equal. Oracle uses non-padded comparison semantics whenever one or both values in the comparison have the datatype VARCHAR2.

SINGLE CHARACTERS:

Oracle compares single characters according to their value in the collating sequence of the database character set. One character is greater than another if it has a greater value than the other in the collating sequence.

These are some common character sets:

* 7-bit ASCII (American Standard Code for Information Interchange)
* EBCDIC (Extended Binary Coded Decimal Interchange Code) Code Page 500
* ISO 8859/1 (International Standards Organization)
* JEUC Japan Extended UNIX
* SJIS Japan Shift JIS

Portions of the ASCII and EBCDIC character sets appear in tables in their respective help screens. Note that uppercase and lowercase letters are not equivalent. Also, note that the collating sequence for a character set may not match the linguistic sequence for a particular language.

ASCII CHARACTER SET

The following table shows the 7-bit ASCII character set.

Decimal value	Symbol	Decimal value	Symbol
32	blank	59	;
33	!	60	<
34	"	61	=
35	#	62	>
36	$	63	?
37	%	64	@
38	&	65 - 90	A - Z
39	'	91	[
40	(	92	\
41	)	93	]
42	*	94	
43	++	95	_

Decimal value	Symbol	Decimal value	Symbol

44	,	96	`
45	-	97 - 122	a - z
46	.	123	{
47	/	124	|
48 - 57	0 - 9	125	}
58	:	126	~

EBCDIC CHARACTER SET

The following table shows a common portion of the EBCDIC character set.

Decimal value	Symbol	Decimal value	Symbol
64	blank	97	/
74	?	107	,
75	.	108	%
76	<	109	_
77	(	110	>
Decimal value	**Symbol**	**Decimal value**	**Symbol**
78	++	111	?
79	|	122	:
80	&	123	#
90	!	124	@
91	$	125	'
92	*	126	=
93	)	127	"
94	;	129 - 169	a - z
95	4	193 - 233	A - Z
96	-	240 - 249	0 - 9

SEE:

Comparison Operators

Data Conversion

Generally an expression cannot contain values of different datatypes. For example, an expression cannot multiply 5 by 10 and then add 'JAMES'. However, Oracle supports both implicit and explicit conversion of values from one datatype to another.

IMPLICIT DATA CONVERSION:

Oracle automatically converts a value from one datatype to another when such a conversion makes sense. For example, although the literal string '10' has datatype CHAR, Oracle automatically converts it to the NUMBER datatype if it appears in a numeric expression.

Conditions can also contain values of different datatypes. In such cases, Oracle often implicitly converts values from one datatype to another. In this statement, Oracle implicitly converts '7936' to 7936:

```
SELECT ename
    FROM emp
    WHERE empno = '7936'
```

In this statement, Oracle implicitly converts '12-MAR-1993' to a DATE value using the default date format 'DD-MON-YYYY':

```
SELECT ename
    FROM emp
    WHERE hiredate = '12-MAR-1993'
```

In this statement, Oracle implicitly converts '00002514.0001.0001' to a ROWID value:

```
SELECT ename
    FROM emp
    WHERE ROWID = '00002514.0001.0001'
```

EXPLICIT DATA CONVERSION:

You can also explicitly specify datatype conversions using SQL conversion functions. The following table shows common SQL functions that explicitly convert a value from one datatype to another.

```
  TO
  --
         CHAR       NUMBER            DATE         RAW          ROWID
FROM     ----       ------            ----         ---          -----
----
CHAR    |unnec.    TO_NUMBER         TO_DATE      HEXTORAW    CHARTOROWID
        |
NUMBER |TO_CHAR     unnec.           TO_DATE
        |                          (number, 'J')
DATE    |TO_CHAR    TO_CHAR          unnec.
```

```
        |           (date, 'J')
RAW     |RAWTOHEX                                    unnec.
        |
ROWID   |ROWIDTOCHAR                                                 unnec.
```

Note: "unnec." means unnecessary

IMPLICIT VS. EXPLICIT DATA CONVERSION:

Oracle Corporation recommends that you specify explicit conversions rather than rely on implicit or automatic conversions for these reasons:

* SQL statements are easier to understand when you use explicit datatype conversions functions.
* Automatic datatype conversion can have a negative impact on performance, especially if the datatype of a column value is converted to that of a constant rather than the other way around.
* Implicit conversion depends on the context in which it occurs and may not work the same way in every case.
* Algorithms for implicit conversion are subject to change across software releases and among Oracle products. Behavior of explicit conversions is more predictable.

SEE:

Conversion Functions, CHARTOROWID, CONVERT, HEXTORAW, RAWTOHEX, ROWIDTOCHAR, TO_CHAR (date conversion), TO_DATE, TO_MULTI_BYTE TO_NUMBER, TO_SINGLE_BYTE

Nulls

If a row lacks a value for a particular column, that column is said to be null, or to contain a null. Nulls can appear in columns of any datatype that are not restricted by NOT NULL or PRIMARY KEY integrity constraints. Use a null when the actual value is unknown or when a value would not be meaningful.

Oracle currently treats a character value with a length of zero as null. However, this may not continue to be true in future versions of Oracle.

Do not use null to represent a value of zero, because they are not equivalent. Any arithmetic expression containing a null always evaluates to null. For example, null added to 10 is null. In fact, all operators (except concatenation) return null when given a null operand.

NULLS IN SQL functions:

All scalar functions (except NVL and TRANSLATE) return null when given a null argument. The NVL function can be used to return a value when a null occurs. For example, the expression NVL(COMM,0) returns 0 if COMM is null or the value of COMM if it is not null.

Most group functions ignore nulls. For example, consider a query that averages the five values 1000, null, null, null, and 2000. Such a query ignores the nulls and calculates the average to be (1000+2000)/2 = 1500.

NULLS WITH COMPARISON OPERATORS:

To test for nulls, only use the comparison operators IS NULL and IS NOT NULL. If you use any other operator with nulls and the result depends on the value of the null, the result is unknown. Because null represents a lack of data, a null cannot be equal or unequal to any value or to another null. However, note that Oracle considers two nulls to be equal when evaluating a DECODE expression.

NULLS IN CONDITIONS:

Oracle treats conditions evaluating to unknown values as FALSE. For example, since the condition COMM = NULL is always unknown, a SELECT statement with this condition in its WHERE clause returns no rows. Note that Oracle returns no error message in this case.

The following table summarizes results of conditions using nulls.

If A is:	Condition	Evaluates to:
10	a IS NULL	FALSE
10	a IS NOT NULL	TRUE
NULL	a IS NULL	TRUE
NULL	a IS NOT NULL	FALSE
10	a = NULL	Unknown

```
    10              a != NULL           Unknown
    NULL            a = NULL            Unknown
    NULL            a != NULL           Unknown
```

SEE:

Arithmetic Operators, Comparison Operators, NVL

Pseudocolumns

A pseudocolumn behaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values. This section describes these pseudocolumns:

* CURRVAL
* NEXTVAL
* LEVEL
* ROWID
* ROWNUM

SEE:
CURRVAL and NEXTVAL, LEVEL, ROWNUM, ROWID

CURRVAL and NEXTVAL

A sequence is a schema object that can generate unique sequential values. These values are often used for primary and unique keys. You can refer to sequence values in SQL statements with these pseudocolumns:

CURRVAL
returns the current value of a sequence.

NEXTVAL
increments the sequence and returns the next value.

You must qualify CURRVAL and NEXTVAL with the name of the sequence:

sequence.CURRVAL
sequence.NEXTVAL

To refer to the current or next value of a sequence in the schema of another user, you must have been granted either SELECT object privilege on the sequence or SELECT ANY SEQUENCE system privilege and you must qualify the sequence with the schema containing it:

schema.sequence.CURRVAL
schema.sequence.NEXTVAL

To refer to the value of a sequence on a remote database, you must qualify the sequence with a complete or partial name of a database link:

schema.sequence.CURRVAL@dblink
schema.sequence.NEXTVAL@dblink

If you are using Trusted Oracle in DBMS MAC mode, you can only refer to a sequence if your DBMS label dominates the sequence's creation label or if one of these criteria is satisfied:

* If the sequence's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
* If the sequence's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
* If the sequence's creation label and your DBMS label are noncomparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

If you are using Trusted Oracle in OS MAC mode, you cannot refer to a sequence with a lower creation label than your DBMS label.

USING SEQUENCE VALUES:

You can use CURRVAL and NEXTVAL in these places:

* the SELECT list of a SELECT statement
* the VALUES clause of an INSERT statement
* the SET clause of an UPDATE statement

You cannot use CURRVAL and NEXTVAL in these places:

* a subquery
* a view's query or snapshot's query
* a SELECT statement with the DISTINCT operator
* a SELECT statement with a GROUP BY or ORDER BY clause
* a SELECT statement that is combined with another SELECT statement with the UNION, INTERSECT, or MINUS set operator
* the WHERE clause of a SELECT statement
* DEFAULT value of a column in a CREATE TABLE or ALTER TABLE statement
* the condition of a CHECK constraint

Also, within a single SQL statement, all referenced sequences, LONG columns, updated tables, and locked tables must be located on the same database.

When you create a sequence, you can define its initial value and the increment between its values. The first reference to NEXTVAL returns the sequence's initial value. Subsequent references to NEXTVAL increment the sequence value by the defined increment and return the new value. Any reference to CURRVAL always returns the sequence's current value, which is the value returned by the last reference to NEXTVAL. Note that before you use CURRVAL for a sequence in your session, you must first increment the sequence with NEXTVAL.

You can only increment a sequence once in a single SQL statement. If a statement contains more than one reference to NEXTVAL for a sequence, Oracle increments the sequence once and returns the same value for all occurrences of NEXTVAL. If a statement contains references to both CURRVAL and NEXTVAL, Oracle increments the sequence and returns the same value for both CURRVAL and NEXTVAL regardless of their order within the statement.

A sequence can be accessed by many users concurrently with no waiting or locking.

Example I:

This example selects the current value of the employee sequence:

```
SELECT empseq.currval
    FROM DUAL
```

Example II:

This example increments the employee sequence and uses its value for a new employee inserted into the employee table:

```
INSERT INTO emp
    VALUES (empseq.nextval, 'LEWIS', 'CLERK',
            7902, SYSDATE, 1200, NULL, 20)
```

Example III:

This example adds a new order with the next order number to the master order table and then adds suborders with this number to the detail order table:

```
INSERT INTO master_order(orderno, customer, orderdate)
    VALUES (orderseq.nextval, 'Al''s Auto Shop', SYSDATE)
INSERT INTO detail_order (orderno, part, quantity)
    VALUES (orderseq.currval, 'SPARKPLUG', 4)
INSERT INTO detail_order (orderno, part, quantity)
    VALUES (orderseq.currval, 'FUEL PUMP', 1)
INSERT INTO detail_order (orderno, part, quantity)
    VALUES (orderseq.currval, 'TAILPIPE', 2)
```

SEE:

INSERT, Pseudocolumns, SELECT, UPDATE

LEVEL

For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root node, 2 for a child of a root, and so on. A root node is the highest node within an inverted tree. A child node is any non-root node. A parent node is any row that has children. A leaf node is any row without children.

To define a hierarchical relationship in a query, you must use the START WITH and CONNECT BY clauses. For more information on using the LEVEL pseudocolumn, see the SELECT command.

SEE:

Pseudocolumns, SELECT

ROWID

For each row in the database, the ROWID pseudocolumn returns a row's address. ROWID values contain information necessary to locate a row:

* which data block in the data file
* which row in the data block (first row is 0)
* which data file (first file is 1)

In most cases, a ROWID value uniquely identifies a row in the database. However, rows in different tables that are stored together in the same cluster can have the same ROWID.

Values of the ROWID pseudocolumn have the datatype ROWID.

ROWID values have several important uses:

* They are the fastest means of accessing a single row.
* They can show you how a table's rows are stored.
* They are unique identifiers for rows in a table.

A ROWID does not change during the lifetime of its row. However, you should not use ROWID as a table's primary key. If you delete and reinsert a row with the Import and Export utilities, for example, its ROWID may change. If you delete a row, Oracle may reassign its ROWID to a new row inserted later.

Although you can use the ROWID pseudocolumn in the SELECT and WHERE clauses of a query, these pseudocolumn values are not actually stored in the database. You cannot insert, update, or delete a value of the ROWID pseudocolumn.

EXAMPLE:

This statement selects the address of all rows that contain data for employees in department 20:

```
SELECT ROWID, ename
    FROM emp
    WHERE deptno = 20

ROWID                  ENAME
-----------------     ----------
0000000F.0000.0002     SMITH
0000000F.0003.0002     JONES
0000000F.0007.0002     SCOTT
0000000F.000A.0002     ADAMS
0000000F.000C.0002     FORD
```

SEE:

Pseudocolumns, ROWNUM

ROWNUM

For each row returned by a query, the ROWNUM pseudocolumn returns a number indicating the order in which Oracle selects the row from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on.

You can use ROWNUM to limit the number of rows returned by a query, as in this example:

```
SELECT *
    FROM emp
    WHERE ROWNUM < 10
```

You can also use ROWNUM to assign unique values to each row of a table, as in this example:

```
UPDATE tabx
    SET col1 = ROWNUM
```

Oracle assigns a ROWNUM value to each row as it is retrieved, before rows are sorted for an ORDER BY clause, so an ORDER BY clause normally does not affect the ROWNUM of each row. However, if an ORDER BY clause causes Oracle to use an index to access the data, Oracle may retrieve the rows in a different order than without the index, so the ROWNUMs may be different than without the ORDER BY clause.

Note that conditions testing for ROWNUM values greater than a positive integer are always false. For example, this query returns no rows:

```
SELECT * FROM emp
    WHERE ROWNUM > 1
```

The first row fetched is assigned a ROWNUM of 1 and makes the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 and also makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

SEE:
Pseudocolumns, ROWID

Comments (SQL)

You can associate comments with SQL statements and schema objects.

COMMENTS WITHIN SQL STATEMENTS:

Comments within SQL statements do not affect the statement execution, but they may make your application easier for you to read and maintain. You may want to include a comment in a statement that describes the statement's purpose within your application.

A comment can appear between any keywords, parameters or punctuation marks in a statement. You can include a comment in a statement using either of these means:

* Begin the comment with /*. Proceed with the text of the comment. This text can span multiple lines. End the comment with */. The opening and terminating characters need not be separated from the text by a space or a line break.
* Begin the comment with -- (two hyphens). Proceed with the text of the comment. This text cannot extend to a new line. End the comment with a line break.

A SQL statement can contain multiple comments of both styles. The text of a comment can contain any printable characters in your database character set.

You can use comments in a SQL statement to pass instructions, or hints, to the Oracle optimizer. The optimizer uses these hints to choose an execution plan for the statement.

Note that you cannot use these styles of comments between SQL statements in a SQL script. You can use the SQL*DBA or SQL*Plus REMARK command for this purpose.

EXAMPLE:

These statements contain many comments:

```
SELECT ename, sal + NVL(comm, 0), job, loc
/* Select all employees whose compensation is
greater than that of Jones.*/
    FROM emp, dept
          /*The DEPT table is used to get the department name.*/
    WHERE emp.deptno = dept.deptno
      AND sal + NVL(comm,0) >             /* Subquery:            */
     (SELECT sal + NVL(comm,0)/* total compensation is sal + comm */
         FROM emp
         WHERE ename = 'JONES')

SELECT ename,                    -- select the name
       sal + NVL(comm, 0)        -- total compensation
       job                       -- job
       loc                       -- and city containing the office
```

```
FROM emp,                     -- of all employees
     dept
WHERE emp.deptno = dept.deptno
  AND sal + NVL(comm, 0) > -- whose compensation is greater than
    (SELECT sal + NVL(comm,0)    -- the compensation
        FROM emp
        WHERE ename = 'JONES')   -- of Jones.
```

COMMENTS ON SCHEMA OBJECTS:

You can associate a comment with a table, view, snapshot, or column using the COMMENT command. Comments associated with schema objects are stored in the data dictionary.

Operators

An operator is used to manipulate individual data items and return a result. These items are called operands or arguments. Operators are represented by special characters or by keywords. For example, the multiplication operator is represented by an asterisk (*) and the operator that tests for nulls is represented by the keywords IS NULL.

SEE:

Arithmetic Operators, Character Operators, Comparison Operators
Logical Operators, Precedence, Set Operators, Other Operators

Precedence

An important property of an operator is its precedence. Precedence is the order in which Oracle evaluates different operators in the same expression. When evaluating an expression containing multiple operators, Oracle evaluates operators with higher precedence before evaluating those with lower precedence. Oracle evaluates operators with equal precedence from left to right within an expression.

The following table lists the levels of precedence among SQL operators from high to low. Operators listed on the same line have the same precedence.

Highest Precedence

Unary + - arithmetic operators PRIOR operator

* / arithmetic operators,

Binary + - arithmetic operators || character operators

All comparison operators,

NOT logical operator,

AND logical operator,

OR logical operator,

Lowest Precedence

SQL OPERATOR PRECEDENCE:

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	SELECT * FROM orders WHERE qtysold = -1 SELECT * FROM emp WHERE -sal < 0
* /	Multiplies, divides. These are binary operators.	UPDATE emp SET sal = sal * 1.1

+ -	Adds, subtracts. These are binary operators.	SELECT sal + comm FROM emp WHERE sysdate - hiredate > 365

You can use parentheses in an expression to override operator precedence. Oracle evaluates expressions inside parentheses before evaluating those outside.

SQL also supports set operators (UNION, UNION ALL, INTERSECT, and MINUS) which combine sets of rows returned by queries, rather than individual data items. All set operators have equal precedence.

EXAMPLE:

Consider this expression:

```
1+2*3
```

Because multiplication has a higher precedence than addition, Oracle first multiplies 2 by 3 and then adds the result to 1.

Arithmetic Operators

You can use an arithmetic operator in an expression to negate, add, subtract, multiply, and divide numeric values. The result of the operation is also a numeric value. Some of these operators are also used in date arithmetic. The following table lists arithmetic operators.

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	SELECT * FROM orders WHERE qtysold = -1 SELECT * FROM emp WHERE -sal < 0
* /	Multiplies, divides. These are binary operators.	UPDATE emp SET sal = sal * 1.1
Operator	Purpose	Example
+ -	Adds, subtracts. These are binary operators.	SELECT sal + comm FROM emp WHERE sysdate - hiredate > 365

Do not use consecutive minus signs with no separation (--) in arithmetic expressions to indicate double negation or the subtraction of a negative value. The characters -- are used to begin comments within SQL statements. You should separate consecutive minus signs with a space or a parenthesis.

SEE:

Datatype Comparison Rules, NUMBER Datatype, DATE Datatype

Character Operators

Character operators are used in expressions to manipulate character strings. The following table lists the single character operator.

The result of concatenating two character strings is another character string. If both character strings are of datatype CHAR, the result has datatype CHAR and is limited to 255 characters. If either string is of datatype VARCHAR2, the result has datatype VARCHAR2 and is limited to 2000 characters. Trailing blanks in character strings are preserved by concatenation, regardless of the strings' datatypes. For more information on the differences between the CHAR and VARCHAR2 datatypes, see the Character Datatypes section.

On most platforms, the concatenation operator is two solid vertical bars, as shown in the following table. However, some IBM platforms use broken vertical bars for this operator. Also, you can concatenate character strings with the CONCAT character function.

```
Operator  Purpose                          Example
--------  -------                          -------
||        Concatenates character strings.  SELECT 'Name is ' || ename
                                           FROM emp
```

Although Oracle treats zero-length character strings as nulls, concatenating a zero-length character string with another operand always results in the other operand, rather than a null. However, this may not continue to be true in future versions of Oracle. To concatenate an expression that might be null, use the NVL function to explicitly convert the expression to a zero-length string.

EXAMPLE:

This example creates a table with both CHAR and VARCHAR2 columns, inserts values both with and without trailing blanks, and then selects these values, concatenating them. Note that for both CHAR and VARCHAR2 columns, the trailing blanks are preserved.

```
CREATE TABLE tab1 (col1 VARCHAR2(6), col2 CHAR(6),
                   col3 VARCHAR2(6), col4 CHAR(6) )

INSERT INTO tab1(col1,  col2,     col3,     col4)
         VALUES ('abc', 'def   ', 'ghi   ', 'jkl')

SELECT col1||col2||col3||col4 "Concatenation"
    FROM tab1

Concatenation
------------------
bcdef   ghi   jkl
```

Comparison Operators (SQL)

Comparison operators are used in conditions that compare one expression to another. The result of comparing one expression to another can be TRUE, FALSE, or unknown.

Operator	Purpose	Example
=	Equality test.	SELECT * FROM emp WHERE sal = 1500
!=, >=, <>	inequality operator may not be available on all platforms.	FROM emp WHERE SAL != 1500

Operator	Purpose	Example
>, <	Greater than and less than tests	SELECT * FROM emp WHERE SAL > 1500 SELECT * FROM emp WHERE sal < 1500
>=, <=	Greater than or equal to, and less than or equal to, tests.	SELECT * FROM emp WHERE sal >= 1500 SELECT * FROM emp WHERE sal <= 1500
IN	Equal to any member of test. Equivalent to = ANY	SELECT * FROM emp WHERE job IN ('CLERK','ANALYST')

Operator	Purpose	Example
		SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30)
NOT IN	Equivalent to != ALL. Evaluates to FALSE if any member of the set is NULL.	SELECT * FROM emp WHERE sal NOT IN (SELECT sal

```
                                                    FROM emp
                                                    WHERE deptno = 30)

                                                    SELECT *
                                                    FROM emp
                                                    WHERE job NOT IN
                                                    ('CLERK','ANALYST')
Operator  Purpose                                   Example
--------  -------                                   -------
ANY,      Compares a value to each value in a       SELECT *
SOME      list or returned by a query.  Must        FROM emp
          be preceded by =, !=, >, <, <=,           WHERE sal = ANY
          or >=.  Evaluates to FALSE if the         (SELECT sal FROM emp
          query returns no rows.                    WHERE deptno = 30)

ALL       Compares a value to every value in a      SELECT *
          list or returned by a query.  Must be     FROM emp
          preceded by =, !=, >, <, <=, or >=.       WHERE sal >=
          Evaluates to TRUE if the query            ALL ( 1400, 3000 )
          returns no rows.

[NOT]     [Not] greater than or equal to x          SELECT *
BETWEEN   and less than or equal to y.              FROM emp
x AND y                                             WHERE sal
                                                    BETWEEN 2000 AND 3000


Operator  Purpose                                   Example
--------  -------                                   -------
EXISTS    TRUE if a subquery returns at least       SELECT dname, deptno
          one row.                                  FROM dept
                                                    WHERE EXISTS
                                                    (SELECT *
                                                    FROM emp
                                                    WHERE dept.deptno =
                                                    emp.deptno)

x [NOT]     TRUE if x does [not] match the          See LIKE Operator
LIKE y      pattern y.  Within y, the character
[ESCAPE z]  % matches any string of zero or more
            characters except null.  The character
            matches any single character.

IS [NOT]  Tests for nulls.  This is the only        SELECT *
NULL      operator that should be used to test      FROM emp
          for NULLS.  See Nulls                     WHERE comm IS NULL
```

SEE:
    Datatype Comparison Rules, Nulls

NOT IN Operator

All rows evaluate to false (and no rows are returned) if any item in the list following a NOT IN operation is null. For example, this statement returns 'TRUE':

```
SELECT 'TRUE'
    FROM emp
    WHERE deptno NOT IN (5,15)
```

However, this statement returns no rows:

```
SELECT 'TRUE'
    FROM emp
    WHERE deptno NOT IN (5,15,null)
```

This example returns no rows because the WHERE clause condition evaluates to:

```
deptno != 5 AND deptno != 15 AND deptno != null
```

Because all conditions that compare a null result in null, the entire expression results in a null. This behavior can easily be overlooked, especially when the NOT IN operator references a subquery.

LIKE Operator

The LIKE operator is used in character string comparisons with pattern matching. The syntax for a condition using the LIKE operator is shown in this diagram:

SYNTAX:

```
char1 [NOT] LIKE char2 [ESCAPE 'c']
```

where:

char1
is a value to be compared with a pattern. This value can have datatype CHAR or VARCHAR2.

NOT
logically inverts the result of the condition, returning FALSE if the condition evaluates to TRUE and TRUE if it evaluates to FALSE.

char2
is the pattern to which char1 is compared. The pattern is a value of datatype CHAR or VARCHAR2 and can contain the special pattern matching characters % and _.

ESCAPE
identifies a single character as the escape character. The escape character can be used to cause Oracle to interpret % or _ literally, rather than as a special character, in the pattern.
If you wish to search for strings containing an escape character, you must specify this character twice. For example, if the escape character is '/', to search for the string 'client/server', you must specify, 'client//server'.

While the equal (=) operator exactly matches one character value to another, the LIKE operator matches a portion of one character value to another by searching the first value for the pattern specified by the second.

With the LIKE operator, you can compare a value to a pattern rather than to a constant. The pattern can only appear after the LIKE keyword. For example, you can issue the following query to find the salaries of all employees with names beginning with 'SM':

```
SELECT sal
    FROM emp
    WHERE ename LIKE 'SM%'
```

The following query finds the salaries of all employees with the name 'SM%', since the query uses the equality operator instead of the LIKE operator:

```
SELECT sal
    FROM emp
    WHERE ename = 'SM%'
```

The following query finds the salaries of all employees with the name 'SM%'. Oracle interprets 'SM%' as a text literal, rather than as a pattern, because it precedes the LIKE operator:

```
SELECT sal
    FROM emp
    WHERE 'SM%' LIKE ename
```

Patterns often use special characters that Oracle matches with different characters in the value:

* An underscore (_) in the pattern matches exactly one character (as opposed to one byte in a multi-byte character set) in the value.
* A percent sign (%) in the pattern can match zero or more characters (as opposed to bytes in a multi-byte character set) in the value. Note that the pattern '%' cannot match a null.

Case is significant in all conditions comparing character expressions including the LIKE and equality (=) operators. You can use the UPPER() function to perform a case insensitive match, as in this condition:

```
UPPER(ename) LIKE 'SM%'
```

When LIKE is used to search an indexed column for a pattern, the performance benefit associated with the index is lost if the first character in the pattern is % or _. If the leading character in the pattern is not % or _, there is some performance benefit to the index because Oracle can restrict the comparison to rows known to begin with the specified first character.

EXAMPLE I:

This condition is true for all ENAME values beginning with MA:

```
ename LIKE 'MA%'
```

All of these ENAME values make the condition TRUE:

```
MARTIN, MA, MARK, MARY
```

Since case is significant, ENAME values beginning with Ma, ma, and mA make the condition FALSE.

EXAMPLE II:

Consider this condition:

```
ename LIKE 'SMITH_'
```

This condition is true for these ENAME values:

```
SMITHE, SMITHY, SMITHS
```

This condition is false for 'SMITH', since the special character _ must match exactly one character of the ENAME value.

THE ESCAPE OPTION:

You can include the actual characters % or _ in the pattern by using the ESCAPE option. The ESCAPE option identifies the escape character. If the escape character appears in the pattern before the character % or _, Oracle interprets this character literally in the pattern, rather than as a special pattern matching character.

EXAMPLE III:

To search for any employees with the character string 'A_B' in their name:

```
SELECT ename
    FROM emp
    WHERE ename LIKE '%A\_B%' ESCAPE '\'
```

The ESCAPE option identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (_). This causes Oracle to interpret the underscore literally, rather than as a special pattern matching character.

Logical Operators (SQL)

A logical operator is used to combine the results of two component conditions to produce a single result based on them or to invert the result of a single condition. The following table lists logical operators.

Operator	Function	Example

NOT	Returns TRUE if the following condition is FALSE; otherwise returns FALSE.	`SELECT * FROM emp WHERE NOT (job IS NULL)`
		
		`SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000)`

Operator	Function	Example
AND	Returns TRUE if both component conditions are TRUE; otherwise returns FALSE.	`SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10`
OR	Returns TRUE if either component conditions are either TRUE or unknown; otherwise returns FALSE.	`SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10`

For example, in the WHERE clause of the following SELECT statement, the AND logical operator is used to ensure that only those hired before 1984 and also earning more than $1000 a month are returned:

```
SELECT *
    FROM emp
    WHERE hiredate < TO_DATE('01-JAN-1984', 'DD-MON-YYYY')
      AND sal > 1000
```

SEE:

AND Operator, NOT Operator, OR Operator

NOT Operator

The following table shows the result of applying the NOT operator to an expression.

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

SEE:
Logical Operators, AND Operator, OR Operator

AND Operator

The following table shows the results of combining two expressions with the AND operator.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

SEE:

Logical Operators, OR Operator, NOT Operator

OR Operator

The following table shows the results of combining two logical expressions with the OR operator.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

SEE:
Logical Operators, AND Operator, NOT Operator

Set Operators

Set operators combines the results of two queries into a single result. The following table lists SQL set operators.

```
Operator   Returns
--------   -------
UNION      All distinct rows selected by either query.

UNION ALL  All rows selected by either query, including all duplicates.

INTERSECT  All distinct rows selected by both queries.

MINUS      All distinct rows selected by the first query but not the
           second.
```

All set operators have equal precedence. If a SQL statement contains multiple set operators, Oracle evaluates them from the left to right if no parentheses explicitly specify another order. To comply with emerging SQL standards, a future version of Oracle will give the INTERSECT operator greater precedence than the other set operators, so you should use parentheses to explicitly specify order of evaluation in queries that use the INTERSECT operator with other set operators.

If two queries combined by a set operator select character data, the datatype of the return values are determined as follows:

* If both queries select values of datatype CHAR, the returned values have datatype CHAR.
* If either or both of the queries select values of datatype VARCHAR2, the returned values have datatype VARCHAR2.

Consider these two queries and their results:

```
SELECT part
    FROM orders_list1

PART
----------
SPARKPLUG
FUEL PUMP
FUEL PUMP
TAILPIPE

SELECT part
    FROM orders_list2

PART
----------
CRANKSHAFT
TAILPIPE
TAILPIPE
```

SEE:

INTERSECT Operator, MINUS Operator, UNION ALL Operator, UNION OPERATOR

UNION Operator

EXAMPLE:

This statement combines the results with the UNION operator, which eliminates duplicate selected rows:

```
SELECT part FROM orders_list1
UNION
SELECT part FROM orders_list2

PART
----------
SPARKPLUG
FUEL PUMP
TAILPIPE
CRANKSHAFT
```

SEE:

INTERSECT Operator, MINUS Operator, Set Operators, UNION ALL OPERATOR

UNION ALL Operator

EXAMPLE:

This statement combines the results with the UNION ALL operator which does not eliminate duplicate selected rows:

```
SELECT part FROM orders_list1
UNION ALL
SELECT part FROM orders_list2

PART
----------
SPARKPLUG
FUEL PUMP
FUEL PUMP
TAILPIPE
CRANKSHAFT
TAILPIPE
TAILPIPE
```

Note that the UNION operator returns only distinct rows that appear in either result, while the UNION ALL operator returns all rows. A PART value that appears multiple times in either or both queries (such as 'FUEL PUMP') is returned only once by the UNION operator, but multiple times by the UNION ALL operator.

SEE:

INTERSECT Operator, MINUS Operator, Set Operators, UNION Operator

INTERSECT Operator

EXAMPLE:

This statement combines the results with the INTERSECT operator which returns only those rows returned by both queries:

```
SELECT part FROM orders_list1
INTERSECT
SELECT part FROM orders_list2

PART
----------
TAILPIPE
```

SEE:

MINUS Operator, Set Operators, UNION ALL Operator, UNION Operator

MINUS Operator

EXAMPLE:

This statement combines the results with the MINUS operator which returns only those rows returned by the first query but not in the second:

```
SELECT part FROM orders_list1
MINUS
SELECT part FROM orders_list2

PART
----------
SPARKPLUG
FUEL PUMP
```

SEE:

INTERSECT Operator, Set Operators, UNION ALL Operator, UNION OPERATOR

Other Operators

The following table lists other SQL operators.

(+),	Indicates that the preceding column is the outer join column in a join.	`SELECT ename, dname FROM emp, dept WHERE dept.deptno = emp.deptno (+)`
PRIOR	Evaluates the following expression for the parent row of the current row in a hierarchical, or tree-structured, query. In such a query, you must use this operator in the CONNECT BY clause to define the relationship between parent and child rows. You can also use this operator in other parts of a SELECT statement that performs a hierarchical query. The PRIOR operator is a unary operator and has the same precedence as the unary + and - arithmetic operators.	`SELECT empno, ename, mgr FROM emp CONNECT BY PRIOR empno = mgr`

Functions (SQL)

A SQL function is similar to an operator in that it manipulates data items and returns a result. Functions differ from operators in the format in which they appear with their arguments. This format allows them to operate on zero, one, two, or more arguments:

```
function(argument, argument, ...)
```

If you call a function with an argument of a datatype other than the datatype expected by the function, Oracle implicitly converts the argument to the expected datatype before performing the function.

If you call a function with a null argument, the function automatically returns null. The only functions that do not follow this rule are CONCAT, REPLACE, DUMP. and NVL.

Do not design your applications to rely on Oracle either to evaluate or to not evaluate all arguments to a function.

Note: You can call a Stored PL/SQL function anywhere that you can call a SQL function. Refer to your PL/SQL documentation for information on how to construct a Stored PL/SQL function for use in a SQL statement.

Functions are of these general types:

* single row (or scalar) functions
* group functions (or aggregate) functions

These functions differ in the number of rows upon which they act. A single row function returns a single result row for every row of a queried table or view, while a group function returns a single result row for a group of queried rows.

Single row functions can appear in select lists (provided the SELECT statement does not contain a GROUP BY clause), WHERE clauses, START WITH clauses, and CONNECT BY clauses.

Group functions can appear in select lists and HAVING clauses. If you use the GROUP BY clause in a SELECT statement, Oracle divides the rows of a queried table or view into groups. In a query containing a GROUP BY clause, all elements of the select list must be either expressions from the GROUP BY clause, expressions containing group functions, or constants. Oracle applies the group functions in the select list to each group of rows and returns a single result row for each group. If you omit the GROUP BY clause, Oracle applies group functions in the select list to all the rows in the queried table or view. You can also use group functions in a HAVING clause in the statement to restrict the result rows returned.

SEE:

Single Row Functions, Number Functions, Character Functions, Date Functions, Conversion Functions, Other Functions, Group Functions

Single Row Functions

Single row functions grouped together by the datatypes of their arguments and return values.

SEE:
Number Functions, Character Functions, Date Functions, OTHER FUNCTIONS

Number Functions:

Number functions accept numeric input and return numeric values. Most of these functions return values that are accurate to 38 decimal digits. The transcendental functions (COS, COSH, EXP, LN, LOG, SIN, SINH, SQRT, TAN, TANH) are accurate to 36 decimal digits.

SEE:

ABS, CEIL, COS, COSH, EXP, FLOOR, LN, LOG, MOD, POWER, ROUND, SIGN, SIN, Single Row Functions, SINH, SQRT, TAN, TANH, TRUNC

ABS

SYNTAX:

```
ABS(n)
```

PURPOSE:

Returns the absolute value of n.

EXAMPLE:

```
SELECT ABS(-15) "Absolute"
    FROM DUAL

Absolute
--------
      15
```

SEE:

SIGN

CEIL

SYNTAX:

```
CEIL(n)
```

PURPOSE:

Returns smallest integer greater than or equal to n.

EXAMPLE:

```
SELECT CEIL(15.7) "Ceiling"
    FROM DUAL

Ceiling
-------
     16
```

SEE:

FLOOR

COS

SYNTAX:

```
COS(n)
```

PURPOSE:

Returns the cosine of n (an angle expressed in radians).

EXAMPLE:

```
SELECT COS(180 * 3.14159265359/180)
"Cosine of 180 degrees"
    FROM DUAL

Cosine of 180 degrees
---------------------
                   -1
```

SEE:
COSH, SIN, SINH, TAN, TANH

COSH

SYNTAX:

```
COSH(n)
```

PURPOSE:

Returns the hyperbolic cosine of n.

EXAMPLE:

```
SELECT COSH(0) "Hyperbolic cosine of 0"
    FROM DUAL

Hyperbolic cosine of 0
----------------------
                       1
```

SEE:

COS, SIN, SINH, TAN, TANH

EXP

SYNTAX:

```
EXP(n)
```

PURPOSE:

Returns e raised to the nth power; e = 2.71828183 ...

EXAMPLE:

```
SELECT EXP(4) "e to the 4th power"
    FROM DUAL

e to the 4th power
------------------
      54.59815
```

SEE:

LN, LOG, POWER, SQRT

FLOOR

SYNTAX:

```
FLOOR(n)
```

PURPOSE:

Returns largest integer equal to or less than n.

EXAMPLE:

```
SELECT FLOOR(15.7) "Floor"
    FROM DUAL

Floor
-----
   15
```

SEE:

CEIL

LN

SYNTAX:

```
LN(n)
```

PURPOSE:

Returns the natural logarithm of n, where n is greater than 0.

EXAMPLE:

```
SELECT LN(95) "Natural log of 95"
    FROM DUAL

Natural log of 95
-----------------
   4.55387689
```

SEE:

EXP, LOG, POWER, SQRT

LOG

SYNTAX:

```
LOG(m,n)
```

PURPOSE:

Returns the logarithm, base m, of n. The base m can be any positive number other than 0 or 1 and n can be any positive number.

EXAMPLE:

```
SELECT LOG(10,100) "Log base 10 of 100"
    FROM DUAL

Log base 10 of 100
------------------
                 2
```

SEE:
EXP, LN, POWER, SQRT

MOD

SYNTAX:

```
MOD(m,n)
```

PURPOSE:

Returns remainder of m divided by n. Returns m if n is 0.

EXAMPLE:

```
SELECT MOD(11,4) "Modulus"
    FROM DUAL

Modulus
-------
  3
```

Note:
This function behaves differently from the classical mathematical modulus function when m is negative. The classical modulus can be expressed in terms of the MOD function with this formula:

```
m - n * FLOOR(m/n)
```

EXAMPLE:

This statement illustrates the difference between the MOD function and the classical modulus:

```
SELECT m, n, MOD(m,n),
m - n * FLOOR(m/n) "Classical Modulus"
    FROM test_mod_table

 M    N  MOD(M,N) Classical Modulus
--- ---- -------- -----------------
 11    4        3                 3
-11    4       -3                 1
 11   -4        3                -1
-11   -4       -3                -3
```

SEE:
ROUND, TRUNC

POWER

SYNTAX:

```
POWER(m,n)
```

PURPOSE:

Returns m raised to the nth power. The base m and the exponent n can be any numbers, but if m is negative, n must be an integer.

EXAMPLE:

```
SELECT POWER(3,2) "Raised"
    FROM DUAL

Raised
------
     9
```

SEE:
EXP, LN, LOG, SQRT

ROUND (NUMBER)

SYNTAX:

```
ROUND(n[,m])
```

PURPOSE:

Returns n rounded to m places right of the decimal point; if m is omitted, to 0 places. m can be negative to round off digits left of the decimal point. m must be an integer.

EXAMPLES:

```
SELECT ROUND(15.193,1) "Round"
    FROM DUAL

Round
-----
15.2

SELECT ROUND(15.193,-1) "Round"
    FROM DUAL

Round
-----
20
```

SEE:
MOD, TRUNC

SIGN

SYNTAX:

```
SIGN(n)
```

PURPOSE:

If n<0, the function returns -1; if n=0, the function returns 0; if n>0, the function returns 1.

EXAMPLE:

```
SELECT SIGN(-15) "Sign"
    FROM DUAL

Sign
----
  -1
```

SEE:

ABS

SIN

SYNTAX:

```
SIN(n)
```

PURPOSE:

Returns the sine of n (an angle expressed in radians).

EXAMPLE:

```
SELECT SIN(30 * 3.14159265359/180)
"Sine of 30 degrees"
    FROM DUAL

Sine of 30 degrees
------------------
                .5
```

SEE:
COS, COSH, SINH, TAN, TANH

SINH

SYNTAX:

```
SINH(n)
```

PURPOSE:

Returns the hyperbolic sine of n.

EXAMPLE:

```
SELECT SINH(1) "Hyperbolic sine of 1"
    FROM DUAL

Hyperbolic sine of 1
--------------------
          1.17520119
```

SEE:

COS, COSH, SIN, TAN, TANH

SQRT

SYNTAX:

```
SQRT(n)
```

PURPOSE:

Returns square root of n. The value n cannot be negative. SQRT returns a "real" result.

EXAMPLE:

```
SELECT SQRT(26) "Square root"
    FROM DUAL

Square root
-----------
 5.09901951
```

SEE:
EXP, LOG, LN, POWER

TAN

SYNTAX:

```
TAN(n)
```

PURPOSE:

Returns the tangent of n (an angle expressed in radians).

EXAMPLE:

```
SELECT TAN(135 * 3.14159265359/180)
"Tangent of 135 degrees"
    FROM DUAL

Tangent of 135 degrees
----------------------
                      -1
```

SEE:
COS, COSH, SIN, SINH, TANH

TANH

SYNTAX:

```
TANH(n)
```

PURPOSE:

Returns the hyperbolic tangent of n.

EXAMPLE:

```
SELECT TANH(.5) "Hyperbolic tangent of .5"
    FROM DUAL

Hyperbolic tangent of .5
------------------------
               .462117157
```

SEE:
COS, COSH, SIN, SINH, TAN

TRUNC (NUMBER)

SYNTAX:

```
TRUNC(n[,m])
```

PURPOSE:

Returns n truncated to m decimal places; if m is omitted, to 0 places. m can be negative to truncate (make zero) m digits left of the decimal point.

EXAMPLES:

```
SELECT TRUNC(15.79,1) "Truncate"
    FROM DUAL

Truncate
--------
    15.7

SELECT TRUNC(15.79,-1) "Truncate"
    FROM DUAL

Truncate
--------
      10
```

SEE:

MOD, ROUND

Character Functions

Single row character functions accept character input and can return both character and number values.

SEE:

Character Functions Returning Character Values, Character Functions Returning Number Values

Character Functions Returning Character Values

Unless otherwise noted, these functions all return values with the datatype VARCHAR2 and are limited in length to 2000 bytes. Functions that return values of datatype CHAR are limited in length to 255 bytes. If the length of the return value exceeds the limit, Oracle truncates it and returns the result without an error.

SEE:

CHR, CONCAT, INITCAP, LOWER, LPAD, LTRIM, NLS_INITCAP, NLS_LOWER, NLS_UPPER, REPLACE, RPAD, RTRIM, SOUNDEX, SUBSTR, SUBSTRB, TRANSLATE, UPPER

CHR

SYNTAX:

```
CHR(n)
```

PURPOSE:

Returns the character having the binary equivalent to n in the database character set.

EXAMPLE:

```
SELECT CHR(75) "Character"
    FROM DUAL

Character
---------
K
```

SEE:

ASCII

CONCAT

SYNTAX:

```
CONCAT(char1, char2)
```

PURPOSE:

Returns char1 concatenated with char2. This function is equivalent to the concatenation operator (||).

EXAMPLE:

This example uses nesting to concatenate three character strings:

```
SELECT CONCAT( CONCAT(ename, ' is a '), job) "Job"
    FROM emp
    WHERE empno = 7900

   Job
-------------------------
JAMES is a CLERK
```

INITCAP

SYNTAX:

```
INITCAP(char)
```

PURPOSE:

Returns char, with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

EXAMPLE:

```
SELECT INITCAP('the soap') "Capitalized"
    FROM DUAL

Capitalized
--------------------
The Soap
```

SEE:
LOWER, NLS_INITCAP, NLS_LOWER, NLS_UPPER, UPPER

LOWER

SYNTAX:

```
LOWER(char)
```

PURPOSE:

Returns char, with all letters lowercase. The return value has the same datatype as the argument char (CHAR or VARCHAR2).

EXAMPLE:

```
SELECT LOWER('MR. SAMUEL HILLHOUSE') "Lowercase"
    FROM DUAL

Lowercase
--------------------
mr. samuel hillhouse
```

SEE:
INITCAP, NLS_INITCAP, NLS_LOWER, NLS_UPPER, UPPER

LPAD

SYNTAX:

```
LPAD(char1,n [,char2])
```

PURPOSE:

Returns char1, left-padded to length n with the sequence of characters in char2; char2 defaults to ' ', a single blank. If char1 is longer than n, this function returns the portion of char1 that fits in n.

The argument n is the total length of the return value as it is displayed on your terminal screen. In most character sets, this is also the number of characters in the return value. However, in some multi-byte character sets, the display length of a character string can differ from the number of characters in the string.

EXAMPLE:

```
SELECT LPAD('Page 1',15,'*.') "LPAD example"
    FROM DUAL

LPAD example
---------------
*.*.*.*.*Page 1
```

SEE:

LTRIM, RPAD, RTRIM

LTRIM

SYNTAX:

```
LTRIM(char[,set])
```

PURPOSE:

Removes characters from the left of char, with initial characters removed up to the first character not in set; set defaults to ' ', a single blank.

EXAMPLE:

```
SELECT LTRIM('xyxXxyLAST WORD','xy')
"Left trim example"
    FROM DUAL

Left trim example
-----------------
XxyLAST WORD
```

SEE:

LPAD, RPAD, RTRIM

NLS_INITCAP

SYNTAX:

```
NLS_INITCAP(char [, 'nlsparams'] )
```

PURPOSE:

Returns char, with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric. The value of 'nlsparams' can have this form:

'NLS_SORT = sort'

where sort is either a linguistic sort sequence or BINARY. The linguistic sort sequence handles special linguistic requirements for case conversions. Note that these requirements can result in a return value of a different length than the char. If you omit 'nlsparams', this function uses the default sort sequence for your session.

EXAMPLE:

```
SELECT NLS_INITCAP('ijsland', 'NLS_SORT = XDutch') "Capitalized"
    FROM DUAL

Capitalized
--------------------
IJsland
```

SEE:

INITCAP, LOWER, NLS_LOWER, NLS_UPPER, UPPER

NLS_LOWER

SYNTAX:

```
NLS_LOWER(char [, 'nlsparams'] )
```

PURPOSE:

Returns char, with all letters lowercase. The 'nlsparams' can have the same form and serve the same purpose as in the NLS_INITCAP function.

EXAMPLE:

```
SELECT NLS_LOWER('CITTA''', 'NLS_SORT = XItalian') "Lowercase"
    FROM DUAL

Lowercase
-----------------
citta
```

SEE:

INITCAP, LOWER, NLS_INITCAP, NLS_UPPER, UPPER

NLS_UPPER

SYNTAX:

```
NLS_UPPER(char [, 'nlsparams'] )
```

PURPOSE:

Returns char, with all letters uppercase. The 'nlsparams' can have the same form and serve the same purpose as in the NLS_INITCAP function.

EXAMPLE:

```
SELECT NLS_UPPER('grob', 'NLS_SORT = XGerman') "Uppercase"
    FROM DUAL

Uppercase
---------
GROSS
```

SEE:
INITCAP, LOWER, NLS_INITCAP, NLS_LOWER, UPPER

REPLACE

SYNTAX:

```
REPLACE(char, search_string [,replacement_string])
```

PURPOSE:

Returns char with every occurrence of search_string replaced with replacement_string. If replacement_string is omitted or null, all occurrences of search_string are removed. If search_string is null, char is returned. This function provides a superset of the functionality provided by the TRANSLATE function. TRANSLATE provides single character, one to one, substitution. REPLACE allows you to substitute one string for another as well as to remove character strings.

EXAMPLE:

```
SELECT REPLACE('JACK and JUE','J','BL') "Changes"
    FROM DUAL

Changes
------------
BLACK and BLUE
```

SEE:

SOUNDEX, SUBSTR, SUBSTRB, TRANSLATE

RPAD

SYNTAX:

```
RPAD(char1, n [,char2])
```

PURPOSE:

Returns char1, right-padded to length n with char2, replicated as any times as necessary; char2 defaults to ' ', a single blank. If char1 is longer than n, this function returns the portion of char1 that fits in n.

The argument n is the total length of the return value as it is displayed on your terminal screen. In most character sets, this is also the number of characters in the return value. However, in some multi-byte character sets, the display length of a character string can differ from the number of characters in the string.

EXAMPLE:

```
SELECT RPAD(ename,11,'ab') "RPAD example"
    FROM emp
    WHERE ename = 'TURNER'

RPAD example
------------
TURNERababa
```

SEE:

LPAD, LTRIM, RTRIM

RTRIM

SYNTAX:

```
RTRIM(char [,set])
```

PURPOSE:

Returns char, with final characters removed after the last character not in set; set defaults to ' ', a single blank.

EXAMPLE:

```
SELECT RTRIM('TURNERyxXxy','xy')
"Right trim example"
FROM DUAL

Right trim example
------------------
TURNERyxX
```

SEE:

LPAD, LTRIM, RPAD

SOUNDEX

SYNTAX:

```
SOUNDEX(char)
```

PURPOSE:

Returns a character string containing the phonetic representation of char. This function allows you to compare words that are spelled differently, but sound alike in English.

The phonetic representation is defined in The Art of Computer Programming, Volume 3: Sorting and Searching, by Donald E. Knuth.

EXAMPLE:

```
SELECT ename
    FROM emp
    WHERE SOUNDEX(ename) =
          SOUNDEX('SMYTHE')

ENAME
----------
SMITH
```

SEE:
REPLACE, SUBSTR, SUBSTRB, TRANSLATE

SUBSTR

SYNTAX:

```
SUBSTR(char,m [,n])
```

PURPOSE:

Returns a portion of char, beginning at character m, n characters long. If m is positive, Oracle counts from the beginning of char to find the first character. If m is negative, Oracle counts backwards from the end of char. The value m cannot be 0. If n is omitted, Oracle returns all characters to the end of char. The value n cannot be less than 1.

EXAMPLES:

```
SELECT SUBSTR('ABCDEFG',3,2) "Substring"
    FROM DUAL

Substring
----------
CD

SELECT SUBSTR('ABCDEFG',-3,2) "Reversed Substring"
    FROM DUAL

Reversed Substring
------------------
EF
```

SEE:
REPLACE, SOUNDEX, SUBSTRB, TRANSLATE

SUBSTRB

SYNTAX:

```
SUBSTRB(char,m [,n])
```

PURPOSE:

The same as SUBSTR, except that the arguments m and n are expressed in bytes, rather than in characters. For a single-byte database character set, SUBSTRB is equivalent to SUBSTR.

EXAMPLE:

Assume a double-byte database character set:

```
SELECT SUBSTRB('ABCDEFG',5,4) "Substring with bytes"
    FROM DUAL

Substring with bytes
--------------------
CD
```

SEE:
REPLACE, SOUNDEX, SUBSTR, TRANSLATE

TRANSLATE

SYNTAX:

```
TRANSLATE(char,from,to)
```

PURPOSE:

Returns char with all occurrences of each character in from replaced by its corresponding character in to. Characters in char that are not in from are not replaced. The argument from can contain more characters than to. In this case, the extra characters at the end of from have no corresponding characters in to. If these extra characters appear in char, they are removed from the return value. You cannot use empty string for to in order to remove all characters in from the return value. Oracle interprets the empty string as null, and if this function has a null argument, it returns null.

EXAMPLES:

This statement translates a license number. All letters 'ABC...Z' are translated to 'X' and all digits '012...9' are translated to '9':

```
SELECT TRANSLATE('2KRW229', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
                 '9999999999XXXXXXXXXXXXXXXXXXXXXXXXXX')
"Translate example"
    FROM DUAL

Translate example
-----------------
9XXX999
```

This statement returns a license number with the characters removed and the digits remaining:

```
SELECT TRANSLATE('2KRW229', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
                 '0123456789')
"Translate example"
    FROM DUAL

Translate example
-----------------
2229
```

SEE:
REPLACE, SOUNDEX, SUBSTR, SUBSTRB

UPPER

SYNTAX:

```
UPPER(char)
```

PURPOSE:

Returns char, with all letters uppercase. The return value has the same datatype as the argument char.

EXAMPLE:

```
SELECT UPPER('Large') "Uppercase"
    FROM DUAL

Uppercase
---------
LARGE
```

SEE:
INITCAP, LOWER, NLS_INITCAP, NLS_LOWER, NLS_UPPER

Character Functions Returning Number Values

SEE:

ASCII, INSTR, INSTRB, LENGTH, LENGTHB, NLSSORT

ASCII

SYNTAX:

```
ASCII(char)
```

PURPOSE:

Returns the decimal representation in the database character set of the first byte of char. If your database character set is 7-bit ASCII, this function returns an ASCII value. If your database character set is EBCDIC Code Page 500, this function returns an EBCDIC value. Note that there is not a similar EBCDIC character function.

EXAMPLE:

```
SELECT ASCII('Q')
    FROM DUAL

ASCII('Q')
----------
        81
```

SEE:

CHR

INSTR

SYNTAX:

```
INSTR(char1,char2[,n[,m]])
```

PURPOSE:

Searches char1 beginning with its nth character for the mth occurrence of char2 and returns the position of the character in char1 that is the first character of this occurrence. If n is negative, Oracle counts and searches backward from the end of char1. The value of m must be positive. The default values of both n and m are 1, meaning Oracle begins searching at the first character of char1 for the first occurrence of char2. The return value is relative to the beginning of char1, regardless of the value of n, and is expressed in characters. If the search is unsuccessful (if char2 does not appear m times after the nth character of char1) the return value is 0.

EXAMPLES:

```
SELECT INSTR('CORPORATE FLOOR','OR', 3, 2) "Instring"
    FROM DUAL

Instring
---------
       14

SELECT INSTR('CORPORATE FLOOR','OR', -3, 2)
"Reversed Instring"
    FROM DUAL

Reversed Instring
-----------------
                2
```

SEE:

INSTRB, LENGTH, LENGTHB, NLSSORT, SUBSTR, SUBSTRB

INSTRB

SYNTAX:

```
INSTRB(char1,char2[,n[,m]])
```

PURPOSE:

The same as INSTR, except that n and the return value are expressed in bytes, rather than in characters. For a single-byte database character set, INSTRB is equivalent to INSTR.

EXAMPLE:

Assume a double-byte database character set:

```
SELECT INSTRB('CORPORATE FLOOR','OR',5,2)
"Instring in bytes"
    FROM DUAL

Instring in bytes
-----------------
               27
```

SEE:
INSTR, LENGTH, LENGTHB, NLSSORT, SUBSTR, SUBSTRB

LENGTH

SYNTAX:

```
LENGTH(char)
```

PURPOSE:

Returns the length of char in characters. If char has datatype CHAR, the length includes all trailing blanks. If char is null, this function returns null.

EXAMPLE:

```
SELECT LENGTH('CANDIDE') "Length in characters"
    FROM DUAL

Length in characters
--------------------
                   7
```

SEE:

INSTR, INSTRB, LENGTHB, NLSSORT, SUBSTR, SUBSTRB

LENGTHB

SYNTAX:

```
LENGTHB(char)
```

PURPOSE:

Returns the length of char in bytes. If char is null, this function returns null. For a single-byte database character set, LENGTHB is equivalent to LENGTH.

EXAMPLE:

Assume a double-byte database character set:

```
SELECT LENGTH('CANDIDE') "Length in bytes"
    FROM DUAL

   Length in bytes
   ---------------
                14
```

SEE:

INSTR, INSTRB, LENGTH, NLSSORT, SUBSTR, SUBSTRB

NLSSORT

SYNTAX:

```
NLSSORT(char [, 'nlsparams'])
```

PURPOSE:

Returns the string of bytes used to sort char. The value of 'nlsparams' can have the form

'NLS_SORT = sort'

where sort is a linguistic sort sequence or BINARY. If you omit 'nlsparams', this function uses the default sort sequence for your session. If you specify BINARY, this function returns char.

EXAMPLE:

This function can be used to specify comparisons based on a linguistic sort sequence rather on the binary value of a string:

```
SELECT * FROM emp
WHERE NLSSORT(ename, 'NLS_SORT = German') >
      NLSSORT('B',   'NLS_SORT = German')
```

SEE:

INSTR, INSTRB, LENGTH, LENGTHB, SUBSTR, SUBSTRB

Date Functions

Date functions operate on values of the DATE datatype. All date functions return a value of DATE datatype, except the MONTHS_BETWEEN function, which returns a number.

SEE:
ADD_MONTHS, LAST_DAY, MONTHS_BETWEEN, NEW_TIME, NEXT_DAY, ROUND, SYSDATE, TRUNC, ROUND and TRUNC

ADD_MONTHS

SYNTAX:

```
ADD_MONTHS(d,n)
```

PURPOSE:

Returns the date d plus n months. The argument n can be any integer. If d is the last day of the month or if the resulting month has fewer days than the day component of d, then the result is the last day of the resulting month. Otherwise, the result has the same day component as d.

EXAMPLE:

```
SELECT TO_CHAR(ADD_MONTHS(hiredate,1),
           'DD-MON-YYYY') "Next month"
    FROM emp
    WHERE ename = 'SMITH'

Next month
-----------
17-JAN-1981
```

SEE:

MONTHS_BETWEEN

LAST_DAY

SYNTAX:

```
LAST_DAY(d)
```

PURPOSE:

Returns the date of the last day of the month that contains d. You might use this function to determine how many days are left in the current month.

EXAMPLES:

```
SELECT SYSDATE, LAST_DAY(SYSDATE) "Last",
LAST_DAY(SYSDATE) - SYSDATE "Days Left"
    FROM DUAL

SYSDATE    Last       Days Left
---------  ---------  ---------
18-NOV-92  30-NOV-92         12

SELECT TO_CHAR(ADD_MONTHS(LAST_DAY(hiredate),5),
'DD-MON-YYYY') "Five months"
FROM emp
    WHERE ename = 'MARTIN'

Five months
-----------
28-FEB-1992
```

SEE:

NEXT_DAY

MONTHS_BETWEEN

SYNTAX:

```
MONTHS_BETWEEN(d1,d2)
```

PURPOSE:

Returns number of months between dates d1 and d2. If d1 is later than d2, result is positive; if earlier, negative. If d1 and d2 are either the same days of the month or both last days of months, the result is always an integer; otherwise Oracle calculates the fractional portion of the result based on a 31-day month and also considers the difference in time components of d1 and d2.

EXAMPLE:

```
SELECT MONTHS_BETWEEN(TO_DATE('02-02-1992', 'MM-DD-YYYY'),
                  TO_DATE('01-01-1992', 'MM-DD-YYYY'))
"Months"
     FROM DUAL

    Months
----------
1.03225806
```

SEE:

ADD_MONTHS

NEW_TIME

SYNTAX:

```
NEW_TIME(d,z1,z2)
```

PURPOSE:

Returns the date and time in time zone z2 when date and time in time zone z1 are d. The arguments z1 and z2 can be any of these text strings:

```
'AST' or 'ADT'        Atlantic Standard or Daylight Time
'BST' or 'BDT'        Bering Standard or Daylight Time
'CST' or 'CDT'        Central Standard or Daylight Time
'EST' or 'EDT'        Eastern Standard or Daylight Time
'GMT'                 Greenwich Mean Time
'HST' or 'HDT'        Alaska-Hawaii Standard Time or Daylight Time.
'MST' or 'MDT'        Mountain Standard or Daylight Time
'NST'                 Newfoundland Standard Time
'PST' or 'PDT'        Pacific Standard or Daylight Time
'YST' or 'YDT'        Yukon Standard or Daylight Time
```

NEXT_DAY

SYNTAX:

```
NEXT_DAY(d,char)
```

PURPOSE:

Returns the date of the first weekday named by char that is later than the date d. The argument char must be a day of the week in your session's date language. The return value has the same hours, minutes, and seconds component as the argument d.

EXAMPLE:

This example returns the date of the next Tuesday after March 15, 1992.

```
SELECT NEXT_DAY('15-MAR-92','TUESDAY') "NEXT DAY"
FROM DUAL

NEXT DAY
---------
17-MAR-92
```

SEE:

LAST_DAY

ROUND (DATE)

SYNTAX:

```
ROUND(d[,fmt])
```

PURPOSE:

Returns d rounded to the unit specified by the format model fmt. If you omit fmt, d is rounded to the nearest day.

EXAMPLE:

```
SELECT ROUND(TO_DATE('27-OCT-92'),'YEAR')
"FIRST OF THE YEAR"
    FROM DUAL

FIRST OF THE YEAR
-----------------
01-JAN-93
```

SEE:

ROUND and TRUNC, TRUNC

SYSDATE

SYNTAX:

```
SYSDATE
```

PURPOSE:

Returns the current date and time. Requires no arguments. In distributed SQL statements, this function returns the date and time on your local database. You cannot use this function in the condition of a CHECK constraint.

EXAMPLE:

```
SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') NOW
   FROM DUAL

NOW
-------------------
10-29-1993 20:27:11
```

SEE:

Pseudocolumns

TRUNC (DATE)

SYNTAX:

```
TRUNC(d,[fmt])
```

PURPOSE:

Returns d with the time portion of the day truncated to the unit specified by the format model fmt. If you omit fmt, d is truncated to the nearest day.

EXAMPLE:

```
SELECT TRUNC(TO_DATE('27-OCT-92', 'DD-MON-YY'), 'YEAR')
"First Of The Year"
    FROM DUAL

FIRST OF THE YEAR
-----------------
01-JAN-92
```

SEE:

ROUND, ROUND and TRUNC

ROUND and TRUNC

The following table lists the format models to be used with the ROUND and TRUNC date functions and the units to which they round and truncate dates. The default model, 'DD', returns the date rounded or truncated to the day with a time of midnight.

The starting day of the week used by the format models DAY, DY, and D is specified implicitly by the initialization parameter NLS_TERRITORY.

Format Model	Rounding or Truncating Unit
CC, SCC,	Century
SYYYY, YYYY YEAR, SYEAR YYY, YY, Y	Year (rounds up on July 1)

Format Model	Rounding or Truncating Unit
IYYY, IYY, IY, I,	ISO Year
Q	Quarter (rounds up on the sixteenth day of the second month of the quarter)
MONTH, MON, MM, RM	Month (rounds up on the sixteenth day)
WW	Same day of the week as the first day of the year
IW	Same day of the week as the first day of the ISO year
W	Same day of the week as the first day of the month
DDD, DD, J	Day
DAY, DY, D	Starting day of the week

Format Model	Rounding or Truncating Unit
HH, HH12, HH24	Hour
MI	Minute

SEE:
ROUND, TRUNC

Conversion Functions

Conversion functions convert a value from one datatype to another. Generally, the form of the function names follows the convention datatype TO datatype. The first datatype is the input datatype; the last datatype is the output datatype.

SEE:
CHARTOROWID, CONVERT, HEXTORAW, RAWTOHEX, ROWIDTOCHAR, TO_CHAR (date conversion), TO_CHAR (label conversion), TO_CHAR (number conversion), TO_DATE, TO_LABEL, TO_MULTI_BYTE, TO_NUMBER, TO_SINGLE_BYTE

CHARTOROWID

SYNTAX:

```
CHARTOROWID(char)
```

PURPOSE:

Converts a value from CHAR or VARCHAR2 datatype to ROWID datatype.

EXAMPLE:

```
SELECT ename
    FROM emp
    WHERE ROWID = CHARTOROWID('0000000F.0003.0002')

ENAME
-----
SMITH
```

SEE:
CHAR Datatype, VARCHAR2 Datatype, ROWID Datatype

CONVERT

SYNTAX:

```
CONVERT(char, dest_char_set [,source_char_set] )
```

PURPOSE:

Converts a character string from one character set to another.

The char argument is the value to be converted.

The dest_char_set argument is the name of the character set to which char is converted.

The source_char_set argument is the name of the character set in which char is stored in the database. The default value is the database character set.

Both the destination and source character set arguments can be either literals or columns containing the name of the character set.

For complete correspondence in character conversion, it is essential that the destination character set contains a representation of all the characters defined in the source character set. Where a character does not exist in the destination character set, a replacement character appears. Replacement characters can be defined as part of a character set definition.

Common character sets include:

US7ASCII
: US 7-bit ASCII character set

WE8DEC
: DEC West European 8-bit character set

WE8HP
: HP West European Laserjet 8-bit character set

F7DEC
: DEC French 7-bit character set

WE8EBCDIC500
: IBM West European EBCDIC Code Page 500

WE8PC850
: IBM PC Code Page 850

WE8ISO8859P1
: ISO 8859-1 West European 8-bit character set

EXAMPLE:

```
SELECT CONVERT('Grob','WE8HP','WE8DEC') "Conversion"
FROM DUAL

Conversion
----------
Grob
```

HEXTORAW

SYNTAX:

```
HEXTORAW(char)
```

PURPOSE:

Converts char containing hexadecimal digits to a raw value.

EXAMPLE:

```
INSERT INTO graphics (raw_column)
    SELECT HEXTORAW('7D')
    FROM DUAL
```

SEE:
RAW and LONG RAW Datatypes, RAWTOHEX

RAWTOHEX

SYNTAX:

```
RAWTOHEX(raw)
```

PURPOSE:

Converts raw to a character value containing its hexadecimal equivalent.

EXAMPLE:

```
SELECT RAWTOHEX(raw_column) "Graphics"
    FROM graphics

Graphics
--------
7D
```

SEE:
HEXTORAW, RAW and LONG RAW Datatypes

ROWIDTOCHAR

SYNTAX:

```
ROWIDTOCHAR(rowid)
```

PURPOSE:

Converts a ROWID value to VARCHAR2 datatype. The result of this conversion is always 18 characters long.

EXAMPLE:

```
SELECT ROWID FROM graphics
WHERE ROWIDTOCHAR(ROWID) LIKE '%F38%'

ROWID
------------------
00000F38.0001.0001
```

SEE:
ROWID Datatype, VARCHAR2 Datatype

TO_CHAR (date conversion)

SYNTAX:

```
TO_CHAR(d [, fmt [, 'nlsparams'] ])
```

PURPOSE:

Converts d of DATE datatype to a value of VARCHAR2 datatype in the format specified by the date format fmt. If you omit fmt, d is converted to a VARCHAR2 value in the default date format.

The 'nlsparams' specifies the language in which month and day names and abbreviations are returned. This argument can have this form:

'NLS_DATE_LANGUAGE = language'

If you omit nlsparams, this function uses the default date language for your session.

EXAMPLE:

```
SELECT TO_CHAR(HIREDATE,'Month DD, YYYY')
"New date format"
    FROM emp
    WHERE ename = 'SMITH'

New date format
-------------------------------
December 17, 1980
```

SEE:

TO_CHAR (label conversion), TO_CHAR (number conversion), TO_DATE

TO_CHAR (label conversion)

SYNTAX:

```
TO_CHAR(label [, fmt])
```

PURPOSE:

Converts label of MLSLABEL datatype to a value of VARCHAR2 datatype, using the optional label format fmt. If you omit fmt, label is converted to a VARCHAR2 value in the default label format.

SEE:

MLSLABEL Datatype, TO_CHAR (date conversion), TO_CHAR (number conversion), TO_LABEL

TO_CHAR (number conversion)

SYNTAX:

```
TO_CHAR(n [, fmt [, 'nlsparams'] ])
```

PURPOSE:

Converts n of NUMBER datatype to a value of VARCHAR2 datatype, using the optional number format fmt. If you omit fmt, n is converted to a VARCHAR2 value exactly long enough to hold its significant digits.

The 'nlsparams' specifies these characters that are returned by number format elements:

* decimal character
* group separator
* local currency symbol
* international currency symbol

This argument can have this form:

'NLS_NUMERIC_CHARACTERS = ''dg''
NLS_CURRENCY = ''text''
NLS_ISO_CURRENCY = ''text'' '

The characters d and g represent the decimal character and group separator, respectively. They must be different single-byte characters. Note that within the quoted string, you must use two single-quotes to represent one around the parameter values.

If you omit 'nlsparams' or any one of the parameters, this function uses the default parameter values for your session.

EXAMPLE:

```
SELECT TO_CHAR(17145,'L099G999',
           'NLS_NUMERIC_CHARACTERS = ''.,''
            NLS_CURRENCY = ''AUD'' ') "Char"
    FROM DUAL

Char
--------------
    AUD017,145
```

SEE:
NUMBER Datatype, TO_CHAR (date conversion), TO_CHAR (label conversion), VARCHAR2 Datatype

TO_DATE

SYNTAX:

```
TO_DATE(char [, fmt [, 'nlsparams'] ])
```

PURPOSE:

Converts char of CHAR or VARCHAR2 datatype to a value of DATE datatype. The fmt is a date format specifying the format of char. If you omit fmt, char must be in the default date format. If fmt is 'J', for Julian, then char must be a number.

The 'nlsparams' has the same purpose in this function as in the TO_CHAR function for date conversion.

Do not use the TO_DATE function with a DATE value for the char argument. The returned DATE value can have a different century value than the original char, depending on fmt or the default date format.

EXAMPLE:

```
INSERT INTO bonus (bonus_date)
    SELECT TO_DATE('January 15, 1989, 11:00 A.M.',
                'Month dd, YYYY, HH:MI A.M.',
                'NLS_DATE_LANGUAGE = American')
    FROM DUAL
```

SEE:

CHAR Datatype, DATE Datatype, VARCHAR2 Datatype

TO_LABEL

SYNTAX:

```
TO_LABEL(char [,fmt])
```

PURPOSE:

Converts char, a value of datatype CHAR or VARCHAR2 containing a label in the format specified by the optional parameter fmt, to a value of MLSLABEL datatype. If you omit fmt, char must be in the default label format.

SEE:

CHAR Datatype, MLSLABEL Datatype, VARCHAR2 Datatype

TO_MULTI_BYTE

SYNTAX:

```
TO_MULTI_BYTE(char)
```

PURPOSE:

Returns char with all of its single-byte characters converted to their corresponding multi-byte characters. Any single-byte characters in char that have no multi-byte equivalents appear in the output string as single-byte characters. This function is only useful if your database character set contains both single-byte and multi-byte characters.

TO_NUMBER

SYNTAX:

```
TO_NUMBER(char [,fmt [, 'nlsparams'] ])
```

PURPOSE:

Converts char, a value of CHAR or VARCHAR2 datatype containing a number in the format specified by the optional format model fmt, to a value of NUMBER datatype.

The 'nlsparams' has the same purpose in this function as in the TO_CHAR function for number conversion.

EXAMPLE:

```
UPDATE emp
    SET sal = sal +
        TO_NUMBER('AUD100.00', 'L999D99'
              'NLS_NUMERIC_CHARACTERS = ''.,''',
               NLS_CURRENCY = ''AUD'' ')
        WHERE ename = 'BLAKE'
```

SEE:
CHAR_Datatype, NUMBER_Datatype, VARCHAR2_Datatype

TO_SINGLE_BYTE

SYNTAX:

```
TO_SINGLE_BYTE(char)
```

PURPOSE:

Returns char with all of its multi-byte characters converted to their corresponding single-byte characters. Any multi-byte characters in char that have no single-byte equivalents appear in the output as multi-byte characters. This function is only useful if your database character set contains both single-byte and multi-byte characters.

EXAMPLE:

```
SELECT TO_CHAR( NEW_TIME(TO_DATE('17:47','hh24:mi'),
              'PST','GMT'), 'hh24:mi') "GREENWICH TIME"
    FROM DUAL

GREENWICH TIME
-------------
01:47
```

Other Functions

SEE:

DUMP, GREATEST, GREATEST_LB, LEAST, LEAST_UB, NVL, UID, USER, USERENV, VSIZE

DUMP

SYNTAX:

```
DUMP(expr [,return_format
   [, start_position [, length]] ] )
```

PURPOSE:

Returns a VARCHAR2 value containing the datatype code, length in bytes, and internal representation of expr. The argument return_format specifies the format of the return value and can have any of these values:

8	returns result in octal notation.
10	returns result in decimal notation.
16	returns result in hexadecimal notation.
17	returns result as single characters.

The arguments start_position and length combine to determine which portion of the internal representation to return. The default is to return the entire internal representation in decimal notation.

If expr is null, this function returns 'NULL'.

EXAMPLES:

```
SELECT DUMP(ename, 8, 3, 2) "OCTAL"
    FROM emp
    WHERE ename = 'SCOTT'

OCTAL
---------------------------------
Type=1 Len=5: 117,124

SELECT DUMP(ename, 10, 3, 2) "ASCII" FROM emp
     WHERE ename = 'SCOTT'

ASCII
----------------------------
Type=1 Len=5: 79,84

SELECT DUMP(ename, 16, 3, 2) "HEX" FROM emp
    WHERE ename = 'SCOTT'

HEX
----------------------------
Type=1 Len=5: 4f,54

SELECT DUMP(ename, 17, 3, 2) "CHAR" FROM emp
    WHERE ename = 'SCOTT'

CHAR
-----------------------
```

```
    Type=1 Len=5: O,T
```

SEE:

VARCHAR2 Datatype

GREATEST

SYNTAX:

```
GREATEST(expr [,expr] ...)
```

PURPOSE:

Returns the greatest of the list of exprs. All exprs after the first are implicitly converted to the datatype of the first prior to the comparison. Oracle compares the exprs using non-padded comparison semantics. Character comparison is based on the value of the character in the database character set. One character is greater than another if it has a higher value. If the value returned by this function is character data, its datatype is always VARCHAR2.

EXAMPLE:

```
SELECT GREATEST('HARRY','HARRIOT','HAROLD') "GREATEST"
    FROM DUAL

GREATEST
--------
HARRY
```

SEE:
GREATEST_LB, LEAST, LEAST_UB

GREATEST_LB

SYNTAX:

```
GREATEST_LB(label [,label] ...)
```

PURPOSE:

Returns the greatest lower bound of the list of labels. Each label must either have datatype MLSLABEL or RAW MLSLABEL or be a quoted literal in the default label format. The return value has datatype RAW MLSLABEL.

SEE:

GREATEST, LEAST, LEAST_UB

LEAST

SYNTAX:

```
LEAST(expr [,expr] ...)
```

PURPOSE:

Returns the least of the list of exprs. All exprs after the first are implicitly converted to the datatype of the first prior to the comparison. Oracle compares the exprs using non-padded comparison semantics. If the value returned by this function is character data, its datatype is always VARCHAR2.

EXAMPLE:

```
SELECT LEAST('HARRY','HARRIOT','HAROLD')
"LEAST"
    FROM DUAL

LEAST
------
HAROLD
```

SEE:
GREATEST, GREATEST_LB, LEAST_UB

LEAST_UB

SYNTAX:

```
LEAST_UB(label [,label] ...)
```

PURPOSE:

Returns the least upper bound of the list of labels. Each label must have datatype MLSLABEL or be a quoted literal in the default label format. The return value has datatype RAW MLSLABEL.

SEE:

GREATEST, GREATEST_LB, LEAST

NVL

SYNTAX:

```
NVL(expr1, expr2)
```

PURPOSE:

If expr1 is null, returns expr2; if expr1 is not null, returns expr1. The arguments expr1 and expr2 can have any datatype. If their datatypes are different, Oracle converts expr2 to the datatype of expr1 before comparing them. The datatype of the return value is always the same as the datatype of expr1, unless expr1 is character data in which case the return value's datatype is VARCHAR2.

EXAMPLE:

```
SELECT ename,
NVL(TO_CHAR(COMM),'NOT APPLICABLE') "COMMISSION"
    FROM emp
    WHERE deptno = 30


ENAME     COMMISSION
--------- -----------
ALLEN     300
WARD      500
MARTIN    1400
BLAKE     NOT APPLICABLE
TURNER    0
JAMES     NOT APPLICABLE
```

SEE:

Nulls

UID

SYNTAX:

```
UID
```

PURPOSE:

Returns an integer that uniquely identifies the current user.

SEE:

USER

USER

SYNTAX:

```
USER
```

PURPOSE:

Returns the current Oracle user with the datatype VARCHAR2.

In a distributed SQL statement, the UID and USER functions identify the user on your local database. You cannot use these functions in the condition of a CHECK constraint.

EXAMPLE:

```
SELECT USER, UID
    FROM DUAL

USER          UID
--------- -------
OPS$KING        9
```

SEE:

UID

USERENV

SYNTAX:

```
USERENV(option)
```

PURPOSE:

Returns information of VARCHAR2 datatype about the current session. This information can be useful for writing an application-specific audit trail table or for determining the language-specific characters currently used by your session. You cannot use this function in the condition of a CHECK constraint. The argument option can have any of these values:

'ENTRYID'
: returns available auditing entry identifier. You cannot use this option in distributed SQL statements.

'LABEL'
: returns your current session label. This option is only applicable for Trusted Oracle.

'LANGUAGE'
: returns the language and territory currently used by your session along with the database character set in this form:

  language_territory.characterset

'SESSIONID'
: returns your auditing session identifier. You cannot use this option in distributed SQL statements.

'TERMINAL'
: returns the operating system identifier for your current session's terminal. In distributed SQL statements, this option returns the identifier for your local session.

EXAMPLE:

```
SELECT USERENV('LANGUAGE') "Language"
    FROM DUAL;

Language
-------------------------
AMERICAN_AMERICA.US7ASCII
```

VSIZE

SYNTAX:

```
VSIZE(expr)
```

PURPOSE:

Returns the number of bytes in the internal representation of expr.
If expr is null, this function returns null.

EXAMPLE:

```
SELECT ename, VSIZE(ename) "BYTES"
    FROM emp
    WHERE deptno = 10

ENAME          BYTES
---------- ---------
CLARK              5
KING               4
MILLER             6
```

Group Functions

Group functions return results based on groups of rows, rather than on single rows. In this way, group functions are different from single row functions.

Many group functions accept these options:

DISTINCT
: This option causes a group function to consider only distinct values of the argument expression.

ALL
: This option causes a group function to consider all values including all duplicates. For example, the DISTINCT average of 1, 1, 1, and 3 is 2; the ALL average is 1.5. If neither option is specified, the default is ALL.

If you use the DISTINCT option with a group function, the size of the function's argument is limited to the size of a data block minus some overhead. This size is specified by the initialization parameter DB_BLOCK_SIZE.

All group functions except COUNT(*) ignore nulls. You can use the NVL in the argument to a group function to substitute a value for a null.

If a query with a group function returns no rows or only rows with nulls for the argument to the group function, the group function returns null.

SEE:
AVG, COUNT, GLB, LUB, MAX, MIN, STDDEV, SUM, VARIANCE

AVG

SYNTAX:

```
AVG([DISTINCT|ALL] n)
```

PURPOSE:

Returns average value of n.

EXAMPLE:

```
SELECT AVG(sal) "Average"
    FROM emp

   Average
----------
2073.21429
```

SEE:
MAX, MIN, SUM

COUNT

SYNTAX:

```
COUNT({* | [DISTINCT|ALL] expr})
```

PURPOSE:

Returns the number of rows in the query.

If you specify expr, this function returns rows where expr is not null. You can count either all rows, or only distinct values of expr.

If you specify the asterisk (*), this function returns all rows, including duplicates and nulls.

EXAMPLES:

```
SELECT COUNT(*) "Total"
    FROM emp

Total
-----
   14

SELECT COUNT(job) "Count"
    FROM emp

Count
-----
   14

SELECT COUNT(DISTINCT job) "Jobs"
    FROM emp

Jobs
----
   5
```

GLB

SYNTAX:

```
GLB([DISTINCT|ALL] label)
```

PURPOSE:

Returns the greatest lower bound of label.

SEE:

LUB

LUB

SYNTAX:

```
LUB([DISTINCT|ALL] label)
```

PURPOSE:

Returns the least upper bound of label.

The return values have datatype MLSLABEL.

SEE:

GLB

MAX

SYNTAX:

```
MAX([DISTINCT|ALL] expr)
```

PURPOSE:

Returns maximum value of expr.

EXAMPLE:

```
SELECT MAX(sal) "Maximum"
    FROM emp

Maximum
-------
5000
```

SEE:
AVG, MIN, SUM

MIN

SYNTAX:

```
MIN([DISTINCT|ALL] expr)
```

PURPOSE:

Returns minimum value of expr.

EXAMPLE:

```
SELECT MIN(hiredate) "Minimum Date"
    FROM emp

Minimum Date
------------
17-DEC-80
```

Note:
The DISTINCT and ALL options have no effect on the MAX and MIN functions.

SEE:
AVG, MAX, SUM

STDDEV

SYNTAX:

```
STDDEV([DISTINCT|ALL] x)
```

PURPOSE:

Returns standard deviation of x, a number. Oracle calculates the standard deviation as the square root of the variance defined for the VARIANCE group function.

EXAMPLE:

```
SELECT STDDEV(sal) "Deviation"
    FROM emp

Deviation
----------
1182.50322
```

SEE:

VARIANCE

SUM

SYNTAX:

```
SUM([DISTINCT|ALL] n)
```

PURPOSE:

Returns sum of values of n.

EXAMPLE:

```
SELECT SUM(sal) "Total"
    FROM emp

Total
-----
29025
```

SEE:
AVG, MAX, MIN

VARIANCE

SYNTAX:

```
VARIANCE([DISTINCT|ALL]x)
```

PURPOSE:

Returns variance of x, a number. Oracle calculates the variance of x using this formula:

where:

xi
: is one of the elements of x.

n
: is the number of elements in the set x. If n is 1, the variance is defined to be 0.

EXAMPLE:

```
SELECT VARIANCE(sal) "Variance"
    FROM emp

  Variance
----------
1389313.87
```

SEE:

STDDEV

Reserved Words (SQL)

A name cannot be an Oracle reserved word. The following list contains these reserved words. Words followed by an asterisk (*) are also ANSI reserved words.

ACCESS	COLUMN	EXCLUSIVE
ADD	COMMENT	EXISTS*
ALL*	COMPRESS	FILE
ALTER	CONNECT	FLOAT*
AND*	CREATE*	FOR*
ANY*	CURRENT*	FROM*
AS*	DATE	GRANT*
ASC*	DECIMAL*	GROUP*
AUDIT	DEFAULT*	HAVING*
BETWEEN*	DELETE*	IDENTIFIED
BY*	DESC*	IMMEDIATE
CHAR*	DISTINCT*	IN*
CHECK*	DROP	INCREMENT
CLUSTER	ELSE	INDEX
INITIAL	OF*	SELECT*
INSERT*	OFFLINE	SESSION
INTEGER*	ON*	SET*
INTERSECT	ONLINE	SHARE
INTO*	OPTION*	SIZE
IS*	OR*	SMALLINT*
LEVEL	ORDER*	START
LIKE*	PCTFREE	SUCCESSFUL
LOCK	PRIOR	SYNONYM
LONG	PRIVILEGES*	SYSDATE
MAXEXTENTS	PUBLIC*	TABLE*
MINUS	RAW	THEN
MODE	RENAME	TO*
MODIFY	RESOURCE	TRIGGER
NOAUDIT	REVOKE	UID
NOCOMPRESS	ROW	UNION*
NOT*	ROWID	UNIQUE*
NOWAIT	ROWLABEL	UPDATE*
NULL*	ROWNUM	USER*
NUMBER	ROWS	VALIDATE
VALUES*	VIEW*	WITH*
VARCHAR	WHENEVER*	
VARCHAR2	WHERE*	

Depending on the Oracle product you plan to use to access the object, names might be further restricted by other product-specific reserved words. For a list of a product's reserved words, see the manual for the specific product, such as the PL/SQL User's Guide and Reference.

Format Models

A format model is a character literal that describes the format of DATE or NUMBER data stored in a character string. You can use a format model as an argument of the TO_CHAR or TO_DATE function for these purposes:

* to specify the format for Oracle to use to return a value from the database to you
* to specify the format for a value you have specified for Oracle to store in the database

Note that a format model does not change the internal representation of the value in the database.

SEE:

Changing the Return Format, Supplying the Correct Format, Date Format Models, Format Model Modifiers, Number Format Models

Changing the Return Format

You can use a format model to specify the format for Oracle to use to return values from the database to you.

Example I:

This statement selects the commission values of the employees in department 30 and uses the TO_CHAR function to convert these commissions into character values with the format specified by the number format model '$9,990.99':

```
SELECT ename employee, TO_CHAR(comm,'$9,990.99') commission
    FROM emp
    WHERE deptno = 30

EMPLOYEE    COMMISSION
--------    ----------
ALLEN          $300.00
WARD           $500.00
MARTIN       $1,400.00
BLAKE
TURNER           $0.00
JAMES
```

Because of this format model, Oracle returns the commissions with leading dollar signs, commas every three digits, and two decimal employees with null in the COMM column.

EXAMPLE II:

This statement selects the dates that each employee from department 20 was hired and uses the TO_CHAR function to convert these dates to character strings with the format specified by the date format model 'fmMonth DD, YYYY':

```
SELECT ename, TO_CHAR(Hiredate,'fmMonth DD, YYYY') hiredate
    FROM emp
    WHERE deptno = 20

ENAME       HIREDATE
---------- --------------------
SMITH       December 17, 1980
JONES       April 2, 1981
SCOTT       April 19, 1987
ADAMS       May 23, 1987
FORD        December 3, 1981
```

With this format model, Oracle returns the hire dates with the month spelled out, two digits for the day, and the century included in the year.

Supplying the Correct Format

You can use format models to specify the format of a value that you are converting from one datatype to another datatype required for a column. When you insert or update a column value, the datatype of the value that you specify must correspond to the column's datatype. For example, a value that you insert into a DATE column must be a value of the DATE datatype or a character string in the default date format (Oracle implicitly converts character strings in the default date format to the DATE datatype). If the value is in another format, you must use the TO_DATE function to convert the value to the DATE datatype. You must also use a format model to specify the format of the character string.

EXAMPLE:

This statement updates JONES' hire date using the TO_DATE function with the format mask 'YYYY MM DD' to convert the character string '1992 05 20' to a DATE value:

```
UPDATE emp
    SET hiredate = TO_DATE('1992 05 20','YYYY MM DD')
    WHERE ename = 'JONES'
```

Number Format Models

You can use number format models in these places:

* in the TO_CHAR function to translate a value of NUMBER datatype to VARCHAR2 datatype
* in the TO_NUMBER function to translate a value of CHAR or VARCHAR2 datatype to NUMBER datatype

All number format models cause the number to be rounded to the specified number of significant digits. If a value has more significant digits to the left of the decimal place than are specified in the format, pound signs (#) replace the value.

Number Format Elements

A number format model is composed of one or more number format elements. The table below lists the elements of a number format model.

Element	Example	Description
9	9999	Number of "9"s specifies number of significant digits returned. Blanks are returned for leading zeroes.
0	0999 9990	Returns a leading zero in this position as a zero, rather than as a blank.
$	$9999	Prefixes value with a dollar sign.
B	B9999	Returns zero value as blank, regardless of "0"s in the format model.
MI	9999MI	Returns "-" after negative values. For positive values, a trailing space is returned.

Element	Example	Description
S	S9999	Returns "+" after positive values and "-" for negative values in this position.
PR	9999PR	Returns negative values in <angle brackets>. For positive values, a leading and trailing space is returned.
D	99D99	Returns the decimal character in this position, separating the integral and fractional parts of a number. Specified by the NLS_NUMERIC_CHARACTERS initialization parameter.
G	9G999	Returns the group separator in this position. Specified by the NLS_NUMERIC_CHARACTERS initialization parameter.
C	C999	Returns the ISO currency symbol in this position. Specified by the NLS_ISO_CURRENCY initialization parameter.

Element	Example	Description
L	L999	Returns the local currency symbol in this position. Specified by the NLS_CURRENCY initialization parameter.
, (comma)	9,999	Returns a comma in this position.
. (period)	99.99	Returns a period in this position, separating the integral and fractional parts of a number.
V	999V99	Multiplies value by 10 to the power n where n is the number of "9"s after the "V".
EEEE	9.999EEEE	Returns value in scientific notation.
RN rn	RN	Returns upper- and lower-case Roman numerals.

Value can be an integer between 1 and 3999.

The MI and PR format elements can only appear in the last position of a number format model. The S format element can only appear in the first or last position.

If a number format model does not contain the MI, S, or PR format elements, negative return values automatically contain a leading negative sign and positive values automatically contain a leading space.

A number format model can contain only a single decimal character (D) or period (.), but it can contain multiple group separators (G) or commas (,). A group separator or comma cannot appear to the right of a decimal character or period in a number format model.

The characters returned by some of these format elements are specified by initialization parameters. The table above lists these elements and parameters.

The characters returned by these format elements can also be implicitly specified by the initialization parameter NLS_TERRITORY. For more information on these parameters, see the ALTER SESSION command. You can also change the characters returned by these format elements for your session with the ALTER SESSION command.

SEE:

ALTER SESSION

Date Format Models

You can use date format models in these places:

* in the TO_CHAR function to translate a DATE value that is in a format other than the default date format
* in the TO_DATE function to translate a character value that is in a format other than the default date format

SEE:
TO_CHAR (date conversion), TO_DATE

Default Date Format

The default date format is specified either explicitly with the initialization parameter NLS_DATE_FORMAT or implicitly with the initialization parameter NLS_TERRITORY. You can also change the default date format for your session with the ALTER SESSION command.

SEE:

ALTER SESSION

Date Format Elements

A date format model is composed of one or more date format elements. The table below lists the date format model elements.

Element	Meaning
SCC or CC	Century, "S" prefixes BC dates with "-".
YYYY or SCCCC	4-digit year; "S" prefixes BC dates with "-".
IYYY	4-digit year based on the ISO standard.
YYY or YY or Y	Last 3, 2, or 1 digit(s) of year.
IYY or IY or I	Last 3, 2, or 1 digit(s) of ISO year.
Y,YYY	Year with comma in this position.
SYEAR or YEAR	Year, spelled out; "S" prefixes BC dates with "-".
RR	Last 2 digits of year; for years in other centuries.
BC or AD	BC/AD indicator.
B.C. or A.D.	BC/AD indicator with periods.
Q	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
MM	Month (01-12; JAN = 01).

Element	Meaning
RM	Roman numeral month (I-XII; JAN = I).
MONTH	Name of month, padded with blanks to length of 9 characters.
MON	Abbreviated name of month.
WW	Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.
IW	Week of year (1-52 or 1-53) based on the ISO standard.
W	Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.
DDD	Day of year (1-366).
DD	Day of month (1-31).
D	Day of week (1-7).
DAY	Name of day, padded with blanks to length of 9 characters.
DY	Abbreviated name of day.

Element	Meaning
J	Julian day; the number of days since January 1, 4712 BC. Numbers specified with 'J' must be integers.
AM or PM	Meridian indicator.
A.M. or P.M.	Meridian indicator with periods.
HH or HH12	Hour of day (1-12).
HH24	Hour of day (0-23).
MI	Minute (0-59).
SS	Second (0-59).

SSSSS	Seconds past midnight (0-86399).
- / , . : ; "text"	Punctuation and quoted text is reproduced in the result.

SEE:

Date Format Models

Date Format Elements and National Language Support

The functionality of some date format elements depends on the country and language in which you are using Oracle. For example, these date format elements return spelled values:

* MONTH
* MON
* DAY
* DY
* BC or AD or B.C. or A.D.
* AM or PM or A.M. or P.M.

The language in which these values are returned is specified either explicitly with the initialization parameter NLS_DATE_LANGUAGE or implicitly with the initialization parameter NLS_LANGUAGE. The values returned by the YEAR and SYEAR date format elements are always in English.

The date format element D returns the number of the day of the week (1-7). The day of the week that is numbered 1 is specified implicitly by the initialization parameter NLS_TERRITORY.

ISO Standard Date Format Elements

Oracle calculates the values returned by the date format elements IYYY, IYY, IY, I, and IW according to the ISO standard.

RR Date Format Element

The RR date format element is similar to the YY date format element, but it provides additional flexibility for storing date values in other centuries. The RR date format element allows you to store twenty-first century dates in the twentieth century by specifying only the last two digits of the year. It will also allow you to store twentieth century dates in the twenty-first century in the same way if necessary.

If you use the TO_DATE function with the YY date format element, the date value returned is always in the current century. If you use the RR date format element instead, the century of the return value varies according to the specified two-digit year and the last two digits of the current year. The following table summarizes the behavior of the RR date format element.

		If the specified two-digit year is:	
		0-49	50-99
If the last two digits of the current year are:	0 - 49	The return date is in the current century.	The return date is in the century before the current one.
	50 - 99	The return date is in the century after the current one.	The return date is in the current century.

The following examples demonstrate the behavior of the RR date format element.

EXAMPLES:

Assume these queries are issued prior to the year 2000:

```
SELECT TO_CHAR(TO_DATE('27-OCT-95', 'DD-MON-RR') ,'YYYY')
"4-digit year"
    FROM DUAL

4-digit year
------------
        1995

SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR') ,'YYYY')
"4-digit year"
    FROM DUAL

4-digit year
------------
        2017
```

Assume these queries are issued in the year 2000 or after:

```
SELECT TO_CHAR(TO_DATE('27-OCT-95', 'DD-MON-RR') ,'YYYY')
"4-digit year"
    FROM DUAL


4-digit year
------------
        1995

SELECT TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR') ,'YYYY')
"4-digit year"
    FROM DUAL

4-digit year
------------
        2017
```

Note that the queries return the same values regardless of whether they are issued before or after the year 2000. The RR date format element allows you to write SQL statements that will return the same values after the turn of the century.

Date Format Element Suffixes

The following table lists suffixes that can be added to date format elements:

Suffix	Meaning	Example	
TH	Ordinal number	DDTH	4TH
SP	Spelled number	DDSP	FOUR
SPTH or THSP	Spelled, ordinal number	DDSPTH	FOURTH

When you add one of these suffixes to a date format element, the return value is always in English.

Capitalization of Date Format Elements

Capitalization in a spelled-out word, abbreviation, or Roman numeral follows capitalization in the corresponding format element. For example, the date format model 'DAY' produces capitalized words like 'MONDAY'; 'Day' produces 'Monday'; and 'day' produces 'monday'.

Punctuation and Character Literals in Date Format Models

You can also include these characters in a date format model:

* punctuation such as hyphens, slashes, commas, periods, and colons
* character literals

These characters appear in the return value in the same location as they appear in the format model. Note that character literals must be enclosed in quotation marks (double quotes).

Format Model Modifiers

You can use the FM and FX modifiers in format models for the TO_CHAR function to control blank padding and exact format checking.

A modifier can appear in a format model more than once. In such case, each subsequent occurrence toggles the effects of the modifier. Its effects are enabled for the portion of the model following its first occurrence, and then disabled for the portion following its second, and then reenabled for the portion following its third, and so on.

SEE:

FM, FX

FM (Fill Mode)

This modifier suppresses blank padding in the return value of the TO_CHAR function:

* In a date format element of a TO_CHAR function, this modifier suppresses blanks in subsequent character elements (such as MONTH) and suppresses leading zeroes for subsequent number elements (such as MI) in a date format model. Since there is no blank padding, the length of the return value may vary. Without FM, the result of a character element is always right padded with blanks to a fixed length and the leading zeros are always returned for a number element.

* In a number format element of a TO_CHAR function, this modifier suppresses blanks added to the left of the number in the result to right-justify it in the output buffer. Without FM, the result is always right-justified in the buffer, resulting in blank-padding to the left of the number.

EXAMPLE V:

This statement uses a date format model to return a character expression that contains the character literal the and a comma.

```
SELECT TO_CHAR(SYSDATE, 'fmDDTH "of" Month, YYYY') Ides
    FROM DUAL

Ides
------------------
3RD of April, 1992
```

Note that this statement also uses the FM modifier. If FM is omitted, the month is blank-padded to nine characters:

```
SELECT TO_CHAR(SYSDATE, 'DDTH "of" Month, YYYY') Ides
    FROM DUAL

Ides
-----------------------
03RD of April    , 1992
```

You can include a single quote in the return value by placing two consecutive single quotes in the format mode

EXAMPLE VI:

This statement places a single quote in the return value by using a date format model that includes two consecutive single quotes:

```
SELECT TO_CHAR(SYSDATE, 'fmDay''"s Special"') Menu
    FROM DUAL

Menu
-----------------
Tuesday's Special
```

Two consecutive single quotes can also be used for the same purpose within a character literal in a format model.

SEE:

FX, FORMAT MODEL MODIFIERS

FX (Format Exact)

This modifier specifies exact matching for the character argument and date format model of a TO_DATE function:

* Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model. Without FX, punctuation and quoted text in the character argument need only match the length and position of the corresponding parts of the format model.
* The character argument cannot have extra blanks. Without FX, Oracle ignores extra blanks.
* Numeric data in the character argument must have the same number of digits as the corresponding element in the format model.

Without FX, numbers in the character argument can omit leading zeroes. When FX is enabled, you can disable this check for leading zeroes by using the FM modifier as well.

If any portion of the character argument violates any of these conditions, Oracle returns an error.

SEE:
FM, FORMAT MODEL MODIFIERS

Expr

PURPOSE:

To specify an expression of any datatype. You must use this notation whenever expr appears in conditions, SQL functions, or SQL commands in other parts of this Manual.

SYNTAX:

Expressions have several forms. Oracle does not accept all forms of expressions in all parts of all SQL commands. The description of each command documents the restrictions on the expressions in the command.

FORM I:

A column, pseudocolumn, constant, sequence number, or NULL.

The pseudocolumn can be either LEVEL, ROWID, or ROWNUM. You can only use a pseudocolumn with a table, rather than with a view or snapshot.

ROWLABEL is a column automatically created by Trusted Oracle in every table in the database. If you are using Trusted Oracle, the expression ROWLABEL returns the row's label. If you are not using Trusted Oracle, the expression ROWLABEL always returns NULL.

SYNTAX:

```
{ [[schema.]{table | view | snapshot} .] {column|pseudo-column|ROWLABEL}
| 'text'
| number
| sequence.{CURRVAL | NEXTVAL}
| NULL }
```

EXAMPLES:

```
emp.ename
this is a text string'
10
```

FORM II:

A host variable with an optional indicator variable. Note that this form of expression can only appear in embedded SQL statements or SQL statements processed in an Oracle Call Interfaces program.

SYNTAX:

```
:host_variable [ [INDICATOR] :indicator_variable]
```

EXAMPLES:

```
:employee_name INDICATOR :employee_name_indicator_var
:department_location
```

FORM III:

A call to a SQL or PL/SQL function.

SYNTAX:

```
function_name [( [DISTINCT | ALL] expr [, expr] ... )]
```

EXAMPLES:

```
LENGTH('BLAKE')
ROUND(1234.567*43)
SYSDATE
```

FORM IV:

A combination of other expressions.

SYNTAX:

```
{ (expr) | +expr | -expr | PRIOR expr
| expr * expr | expr / expr
| expr + expr | expr - expr | expr || expr }
```

EXAMPLES:

```
('CLARK' || 'SMITH')
LENGTH('MOOSE') * 57
SQRT(144) + 72
```

SEE:
DECODED EXPRESSION, List of Expressions

Decoded Expression

An expression using the special DECODE syntax:

SYNTAX:

```
DECODE( expr, search, result [, search, result] ... [, default] )
```

To evaluate this expression, Oracle compares expr to each search value one by one. If expr is equal to a search, Oracle returns the corresponding result. If no match is found, Oracle returns default, or, if default is omitted, returns null. If expr and search contain character data, Oracle compares them using non-padded comparison semantics.

The search, result, and default values can be expressions.

Oracle evaluates each search value only before comparing it to expr, rather than evaluating all search values before comparing any of them with expr. Consequently, Oracle never evaluates a search if a previous search is equal to expr.

Oracle automatically converts expr and each search value to the datatype of the first search value before comparing. Oracle automatically converts the return value to the same datatype as the first result. If the first result has the datatype CHAR or if the first result is null, then Oracle converts the return value to the datatype VARCHAR2.

In a DECODE expression, Oracle considers two nulls to be equivalent. If expr is null, Oracle returns the result of the first search that is also null.

The maximum number of components in the DECODE expression, including expr, searches, results, and default is 255.

EXAMPLE:

This expression decodes the value DEPTNO. If DEPTNO is 10, the expression evaluates to 'ACCOUNTING'; if DEPTNO is 20, it evaluates to 'RESEARCH'; etc. If DEPTNO is not 10, 20, 30, or 40, the expression returns 'NONE'.

```
DECODE (deptno, 10, 'ACCOUNTING',
                20, 'RESEARCH',
                30, 'SALES',
                40, 'OPERATION',
                    'NONE')
```

List of Expressions

A parenthesized list of expressions.

SYNTAX:

```
(expr [, expr] ...)
```

EXAMPLES:

```
(10, 20, 40)
('SCOTT', 'BLAKE', 'TAYLOR')
(LENGTH('MOOSE') * 57, -SQRT(144) + 72, 69)
```

Condition

PURPOSE:

To specify a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or unknown. You must use this syntax whenever condition appears in SQL commands.

SYNTAX:

Conditions can have several forms. The description of each command documents the restrictions on the conditions in the command.

FORM I:

A comparison with expressions or subquery results.

SYNTAX:

```
{ expr {= | != |
| expr_list {= | != |
```

FORM II:

A comparison with any or all members in a list or subquery.
For the syntax description of subquery, see the SELECT command.

SYNTAX:

```
{ expr {= | != |
    {ANY | SOME | ALL} {expr_list | subquery}
| expr_list {= | != |
    {ANY | SOME | ALL} { (expr_list [, expr_list] ...) | subquery} }
```

FORM III:

A test for membership in a list or subquery.

SYNTAX:

```
{ expr [NOT] IN {expr_list | subquery}
| expr_list [NOT] IN { (expr_list [, expr_list]... ) | subquery} }
```

FORM IV:

A test for inclusion in a range.

SYNTAX:

```
expr [NOT] BETWEEN expr AND expr
```

FORM V:

A test for nulls.

SYNTAX:

```
expr IS [NOT] NULL
```

FORM VI:

A test for existence of rows in a subquery.

SYNTAX:

```
EXISTS subquery
```

FORM VII:

A test involving pattern matching.

SYNTAX:

```
char1 [NOT] LIKE char2 [ESCAPE 'c']
```

FORM VIII:

A combination of other conditions.

SYNTAX:

```
{ ( condition )
| NOT condition
| condition AND condition
| condition OR condition }
```

SEE:
DELETE, SELECT, UPDATE

Literals (SQL)

The terms literal and constant value are synonymous here and refer to a fixed data value. For example, 'JACK', 'BLUE ISLAND', and '101' are all character literals. 5001 is a numeric literal. Note that character literals are enclosed in single quotes. The quotes allow Oracle to distinguish them from schema object names.

Many SQL statements and functions require you to specify character and numeric literal values. You can also specify literals as part of expressions and conditions. You can specify character literals with the 'text' notation and numeric literals with the integer or number notation, depending on the context of the literal.

Literals include the following:

* Text
* Integer
* Numbers

SEE:
Text, Integer, Number

Text

PURPOSE:

To specify a text or character string literal. You must use this notation to specify values whenever 'text' or char appear in expressions, conditions, SQL functions, and SQL commands.

SYNTAX:

```
'[c | '']...'
```

where:

c
is any member of the user's character set, except a single quote (').

''
are two single quotes. Because a single quote is used to begin and end character literals, you must use two single quotes to represent one single quote within a literal.

SEE:
Literals, Integer, Number

Integer

PURPOSE:

To specify a positive integer. You must use this notation to specify values whenever integer appears in expressions, conditions, SQL functions, and SQL commands.

SYNTAX:

```
digit[digit]...
```

where:

digit
is one of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

EXAMPLES:

```
7
255
```

SEE:
Literals, Number, Text

Number

PURPOSE:

To specify an integer or a real number. You must use this notation to specify values whenever number appears in expressions, conditions, SQL functions, and SQL commands.

SYNTAX:

```
[+|-]{digit[digit]...[.][digit]...
     |.digit[digit]...}[{e|E}[+|-]digit[digit]...]
```

where:

+
-
: indicates a positive or negative value. If you omit the sign, a positive value is the default.

digit
: is one of 0,1,2,3,4,5,6,7,8 or 9.

e
E
: indicates that the number is specified in scientific notation. The digits after the E specify the exponent. The exponent can range between -130 and 125.

SEE:
Integer, Literals, Number