Oracle7™ Server SQL Reference

Release 7.3
February 1996

Part No. A32538–1

Oracle7™ Server SQL Reference, Release 7.3

Part No. A32538–1

Primary Author: Brian Linden
Contributing Authors: Martin Gruber, Brian Quigley
Contributors: Andrea Borr, Bill Bridge, Geroge Chang, Stephen Faris, John Frazzini, Jyotin Gautam, Gary Hallmark, Michael Hartstein, Terry Hayes, Merrill Holt, Ken Jacobs, Jonathan Klein, Bob Kooi, Andrew Mendelsohn, Mark Moore, Maria Pratt, Hari Sankar, Phil Shaw, Marc Simon, Lynne Thieme, Randall Whitman

Preface

This manual contains a complete description of the Structured Query Language (SQL) used to manage information in an Oracle7 database.

Oracle7 SQL is a superset of the American National Standards Institute (ANSI) and the International Standards Organization (ISO) SQL92 standard at entry level conformance.

This manual notes any features that require the distributed option, Parallel Server option, Parallel Query option, or PL/SQL to be installed. Also noted are parts of Oracle7 SQL that are only used with the Trusted Oracle7 Server. For information on PL/SQL, Oracle's procedural language extension to SQL, see *PL/SQL User's Guide and Reference.*

Brief descriptions of Oracle7 embedded SQL are included in this manual. Detailed descriptions of Oracle7 embedded SQL can be found in *Programmer's Guide to the Oracle Precompilers.*

Audience

This Manual is intended for all users of Oracle7 SQL.

How this Manual is Organized

This Manual is divided into the following parts:

Chapter 1: Introduction
This chapter defines SQL and describes its history as well as the advantages of using it to access relational databases.

Chapter 2: Elements of Oracle7 SQL
This chapter describes the basic building blocks of an Oracle7 database and the Oracle7 SQL.

Chapter 3: Operators, Functions, Expressions, Conditions
This chapter describes how to use SQL operators and functions to combine data into expressions and conditions.

Chapter 4: Commands
This chapter lists and describes all of the SQL commands in alphabetical order.

Appendix A: Differences From Previous Versions
This appendix lists differences in Release 7.2 and previous releases of Oracle7 SQL.

Appendix B: Oracle7 and Standard SQL
This appendix describes Oracle7 compliance with ANSI and ISO standards and lists Oracle7 extensions beyond the standards.

Appendix C: Operating System–Specific Dependencies
This appendix notes places in this manual referring to operating system–specific documentation.

Conventions Used in this Manual

This section explains the conventions used in this Manual including:

- icons
- text
- syntax diagrams and notation
- examples
- example data

Icons

This manual uses the following icons:

OSDoc

Additional Information: This icon indicates information that is contained within Oracle operating system–specific documentation. Such references are noted in Appendix C.

Warning: This icon warns you of a possible danger when using a feature.

Text

The text in this manual adheres to the following conventions:

UPPERCASE	Uppercase text is used to call attention to names of Oracle7 tools commands, keywords, filenames, and initialization parameters.
italics	Italicized text is used call to attention to definitions of terms and parameters of SQL commands.

Syntax Diagrams and Notation

The syntax diagrams and notation in this manual show the complete syntax for SQL commands, functions, and other elements. This section describes syntax diagrams and gives examples of how to write SQL statements. Syntax diagrams are made up of these items:

Keywords Keywords are words that have special meanings in the SQL language. In the syntax diagrams in this manual, keywords appear in uppercase. You must use keywords in your SQL statements exactly as they appear in the syntax diagram, except that they can be either uppercase or lowercase. For example, you must use the CREATE keyword to begin your CREATE TABLE statements just as it appears in the CREATE TABLE syntax diagram.

Parameters Parameters act as place holders in syntax diagrams. They appear in lowercase. Parameters are usually names of database objects, Oracle7 datatype names, or expressions. When you see a parameter in a syntax diagram, substitute an object or expression of the appropriate type in your SQL statement. For example, to write a CREATE TABLE statement, use the name of the table you want to create, such as EMP, in place of the *table* parameter in the syntax diagram. Note that parameter names appear in italics in the text.

This lists shows parameters that appear in the syntax diagrams in this manual and examples of the values you might substitute for them in your statements:

Parameter	***Description***	***Examples***
table	The substitution value must be the name of an object of the type specified by the parameter. For a list of all types of objects, see the section, “Schema Objects” on page 2 – 2.	`emp`
c	The substitution value must be a single character from your database character set.	`T` `s`
’text’	The substitution value must be a text string in single quotes. See the syntax description of ’*text*’ on page 2 – 15.	`’Employee records’`
char	The substitution value must be an expression of datatype CHAR or VARCHAR2 or a character literal in single quotes.	`ename` `’Smith’`
condition	The substitution value must be a condition that evaluates to TRUE or FALSE. See the syntax description of *condition* on page 3 – 78.	`ename > ’A’`

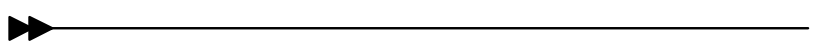
date
d
	The substitution value must be a date constant or an expression of DATE datatype.	`TO_DATE(`
		`’01–Jan–1994’,`
		`’DD–MON–YYYY’)`

expr	The substitution value can be an expression of any datatype as defined in the syntax description of *expr* on page 3 – 73.	`sal + 1000`
integer	The substitution value must be an integer as defined by the syntax description of integer on page 2 – 16.	`72`
label	The substitution value must be an expression of datatype MLSLABEL. For information on such expressions, see the *Trusted Oracle7 Server Administration* guide.	`TO_LABEL(` `’SENSITIVE:ALPHA’)`

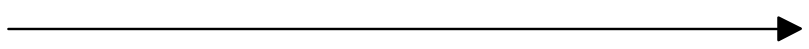
Parameter	***Description***	***Examples***
number *m* *n*	The substitution value must be an expression of NUMBER datatype or a number constant as defined in the syntax description of *number* on page 2 – 17.	`AVG(sal)` `15 * 7`
raw	The substitution value must be an expression of datatype RAW.	`HEXTORAW('7D')`
rowid	The substitution value must be an expression of datatype ROWID.	`00000462.0001.0001`
subquery	The substitution value must be a SELECT statement, which will be used in another SQL statement. See the syntax description of *subquery* on page 4 – 431.	`SELECT ename FROM emp`
:host_variable	The substitution value must be the name of a variable declared in an embedded SQL program. This manual also uses :*host_integer* and :*host_string* to indicate specific datatypes.	`:employee_number`
cursor	The substitution value must be the name of a cursor in an embedded SQL program.	`curs1`
db_name	The substitution value must be the name of a non–default database in an embedded SQL program.	`sales_db`
db_string	The substitution value must be the database identification string for a SQL*Net database connection. For details, see the user's guide for your specific SQL*Net protocol.	
statement_name *block_name*	The substitution value must be an identifier for a SQL statement or PL/SQL block.	`s1` `b1`

Syntax Diagrams This manual uses syntax diagrams to show SQL commands in Chapter 4, "Commands," and to show other elements of the SQL language in Chapter 2, "Elements of Oracle7 SQL," and Chapter 3, "Operators, Functions, Expressions, Conditions." These syntax diagrams use lines and arrows to show syntactic structure. The following list shows the lines and arrows used and their syntactical meaning.

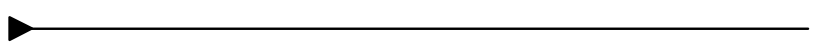
Structure / ***Meaning***



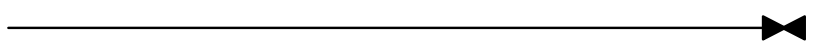
The beginning of a diagram.



The diagram continues on the next line.



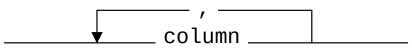
The diagram continues from the previous line.



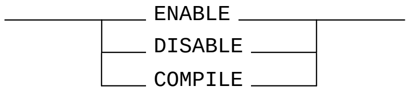
The end of a diagram.

A required item (parameter or keyword). You must use this item.

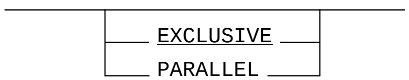
An optional item. You can use the item or omit it.



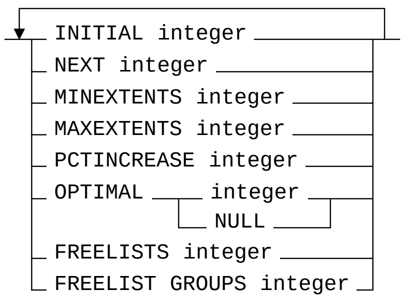
You can optionally repeat the item multiple times. Consecutive items must be separated by a comma.



You must use one of the items.



You can optionally use only one of the items. If there is a default item, it is underlined.



A list of specific items. Each item can only appear once, unless otherwise specified. The items can be listed in any order.

Examples

This manual also contains many examples of SQL statements. These examples show you how to use elements of SQL. The following example shows a CREATE TABLE statement:

```
CREATE TABLE accounts
    (accno    NUMBER,
     owner    VARCHAR2(10),
     balance  NUMBER(7,2) )
```

Note that examples appear in a different font than the text.

Examples follow these case conventions:

- Keywords, such as CREATE and NUMBER, appear in uppercase.
- Names of database objects and their parts, such as ACCOUNTS and ACCNO, appear in lowercase, although they appear in uppercase in the text.

SQL is not case–sensitive (except for quoted identifiers), so you need not follow these conventions when writing your own SQL statements, although your statements may be easier for you to read if you do.

Some Oracle7 tools require you to terminate SQL statements with a special character. For example, SQL statements issued through SQL*Plus may be terminated with a semicolon (;). If you issue these examples statements to Oracle7, you must terminate them with the special character expected by the Oracle7 tool you are using.

Example Data

Many of the examples in this manual operate on sample tables. The definitions of some of these tables appear in a SQL script available on your distribution media. On most operating systems the name of this script is UTLSAMPL.SQL, although its exact name and location may vary depending on your operating system. This script creates sample users and creates these sample tables in the schema of the user SCOTT:

```
CREATE TABLE dept
    (deptno    NUMBER(2)      CONSTRAINT pk_dept PRIMARY KEY,
     dname     VARCHAR2(14),
     loc       VARCHAR2(13) )
CREATE TABLE emp
    (empno     NUMBER(4)      CONSTRAINT pk_emp PRIMARY KEY,
     ename     VARCHAR2(10),
     job       VARCHAR2(9),
     mgr       NUMBER(4),
     hiredate  DATE,
     sal       NUMBER(7,2),
     comm      NUMBER(7,2),
     deptno    NUMBER(2)    CONSTRAINT fk_deptno REFERENCES emp )
CREATE TABLE bonus
    (ename     VARCHAR2(10),
     job       VARCHAR2(9),
     sal       NUMBER,
     comm      NUMBER )
CREATE TABLE salgrade
    (grade     NUMBER,
     losal     NUMBER,
     hisal     NUMBER )
```

The script also fills the sample tables with this data:

```
SELECT * FROM dept

DEPTNO  DNAME      LOC
------- ---------- ---------
10      ACCOUNTING NEW YORK
20      RESEARCH   DALLAS
30      SALES      CHICAGO 40      OPERATIONS BOSTON

SELECT * FROM emp

EMPNO ENAME   JOB          MGR  HIREDATE    SAL   COMM  DEPTNO
----- ------- --------- ------ --------- ------ ------ -------
7369 SMITH   CLERK       7902 17-DEC-80    800             20
7499 ALLEN   SALESMAN    7698 20-FEB-81   1600    300      30
7521 WARD    SALESMAN    7698 22-FEB-81   1250    500      30
7566 JONES   MANAGER     7839 02-APR-81   2975             20
7654 MARTIN  SALESMAN    7698 28-SEP-81   1250   1400      30
7698 BLAKE   MANAGER     7839 01-MAY-81   2850             30
7782 CLARK   MANAGER     7839 09-JUN-81   2450             10
7788 SCOTT   ANALYST     7566 19-APR-87   3000             20
7839 KING    PRESIDENT        17-NOV-81   5000             10
7844 TURNER  SALESMAN    7698 08-SEP-81   1500             30
7876 ADAMS   CLERK       7788 23-MAY-87   1100             20
7900 JAMES   CLERK       7698 03-DEC-81    950             30
7902 FORD    ANALYST     7566 03-DEC-81   3000             20
7934 MILLER  CLERK       7782 23-JAN-82   1300             10

SELECT * FROM salgrade

GRADE LOSAL HISAL
----- ----- -----
1     700   1200
2     1201  1400
3     1401  2000
4     2001  3000
5     3001  9999
```

To perform all the operations of the script, run it when you are logged into Oracle7 as the user SYSTEM.

Your Comments Are Welcome

We value and appreciate your comments as an Oracle7 user and reader of the manuals. As we write, revise, and evaluate, your opinions are the most important input we receive. At the back of this manual is a Reader's Comment Form that we encourage you to use to tell us both what you like and what you dislike about this (or other) Oracle7 manuals. If the form has been used, or you would like to contact us, please use the following address or fax number:

Oracle7 Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065
U.S.A.
FAX: 415–506–7200

Contents

CHAPTER

1

Introduction

Structured Query Language (SQL), pronounced "sequel," is the set of commands that all programs and users must use to access data within the Oracle7 database. Application programs and Oracle7 tools often allow users to access the database without directly using SQL, but these applications in turn must use SQL when executing the user's request. This chapter provides background information on SQL used by most relational database systems. Topics include:

- history of SQL
- SQL standards
- benefits of SQL
- embedded SQL
- lexical conventions
- tools support

History of SQL

The paper, “A Relational Model of Data for large Shared Data Banks,” by Dr. E. F. Codd, was published in June 1970 in the Association of Computer Machinery (ACM) journal, *Communications of the ACM*. Codd’s model is now accepted as the definitive model for relational database management systems (RDBMS). The language, Structured English Query Language(SEQUEL) was developed by IBM Corporation, Inc. to use Codd’s model. SEQUEL later became SQL. In 1979, Relational Software, Inc. (now Oracle Corporation) introduced the first commercially available implementation of SQL. Today, SQL is accepted as the standard RDBMS language.

SQL Standards

Oracle7 SQL complies with industry accepted standards. Oracle Corporation ensures future compliance with evolving SQL standards by actively involving key personnel in SQL standards committees. Industry accepted committees are the American National Standards Institute (ANSI) and the International Standards Organization (ISO), which is affiliated with the International Electrotechnical Commission (IEC), both of which have accepted SQL as the standard language for relational databases. When a new SQL standard is simultaneously published by these organizations, the names of the standards conform to conventions used by the organization, but the technical details are exactly the same.

The latest SQL standard published by ANSI and ISO is often called SQL–92 (and sometimes SQL2). The formal names of the new standard are:

- ANSI X3.135–1992, “Database Language SQL”
- ISO / IEC 9075:1992, “Database Language SQL”

SQL–92 defines three levels of compliance, Entry, Intermediate, and Full. Oracle7, Release 7.2 conforms to Entry level compliance, and has many features that conform to Intermediate or Full level compliance.

Release 7.2 conformance to Entry Level SQL–92 was tested by the National Institute for Standards and Technology (NIST) using the Federal Information Processing Standard (FIPS), FIPS PUB 127–2.

How SQL Works

This section describes many of the reasons for SQL's widespread acceptance by relational database vendors as well as end users. The strengths of SQL benefit all ranges of users including application programmers, database administrators, management, and end users.

Technically speaking, SQL is a data sublanguage. That is to say, the purpose of SQL is to interface to a relational database such as Oracle7, and all SQL statements are instructions to the database. In this it differs from general purposes programming languages like C and Basic. Among the features of SQL are the following:

- it processes sets of data as groups rather than as individual units
- it provides automatic navigation to the data
- it uses statements that are complex and powerful individually, and that therefore stand alone. The flow–control statements of most programming languages are absent in SQL, although they are provided in Oracle's extension to standard SQL called PL/SQL.

Essentially, SQL lets you work with data at the logical level, only being concerned with the implementation details when you want to manipulate them. For example, to retrieve a set of rows from a table, you define a condition used to filter the rows. All rows satisfying the condition are retrieved in a single step and can be passed as a unit to the user , to another SQL statement, or to an application. You need not deal with the rows one by one, nor do you have to worry about how they are physically stored or retrieved. All SQL statements use the optimizer, a part of Oracle7 that determines the fastest means of accessing the specified data. Oracle7 also provides techniques you can use to make the optimizer perform its job better.

SQL provides commands for a variety of tasks including:

- querying data
- inserting, updating, and deleting rows in a table
- creating, replacing, altering, and dropping objects
- controlling access to the database and its objects
- guaranteeing database consistency and integrity

SQL unifies all of the above tasks in one consistent language.

Common Language for All Relational Databases

Because all major relational database management systems support SQL, you can transfer all skills you have gained with SQL from one database to another. In addition, since all programs written in SQL are portable, they can often be moved from one database to another with very little modification.

Embedded SQL

Embedded SQL refers to the use of standard SQL commands embedded within a procedural programming language. Embedded SQL is a collection of these commands:

- all SQL commands, such as SELECT and INSERT, available with SQL with interactive tools
- flow control commands, such as PREPARE and OPEN, which integrate the standard SQL commands with a procedural programming language

Embedded SQL also includes extensions to some standard SQL commands. Chapter 4, "Commands," presents these commands in both standard form and embedded SQL form.

Embedded SQL is supported by the Oracle precompilers. The Oracle precompilers interpret embedded SQL statements and translate them into statements that can be understood by procedural language compilers.

Each of these Oracle precompilers translates embedded SQL programs into a different procedural language:

- the Pro*Ada precompiler
- the Pro*C/C++ precompiler
- the Pro*COBOL precompiler
- the Pro*FORTRAN precompiler
- the Pro*Pascal precompiler
- the Pro*PL/I precompiler

For a definition of the Oracle precompilers, see *Programmer's Guide to the Oracle Precompilers.*

Embedded SQL Terms

The following embedded SQL terms are used throughout this manual:

:host_variable	is a language variable declared according to the rules of the procedural language and used in a SQL statement. A host variable can be a predefined type or a user–defined array and can include an associated indicator variable. You can only use host variables in place of numeric or character expressions. You must precede each host variable by a colon (:) to distinguish it from a schema object name. You cannot use host variables in place of SQL keywords or schema object names. This manual also uses terms for host variables with specific datatypes, such as *:host_integer* and *:host_string*.
cursor	is an identifier for a cursor.
db_name	is an identifier for a non–default database.
db_string	is the database identification string for a SQL*Net connection. For more information about connect strings, see the SQL*Net documentation for your operating system.
statement_name *block_name*	designates an identifier for a SQL statement or PL/SQL block.

Lexical Conventions

The following lexical conventions for issuing SQL statements apply specifically to Oracle's implementation of SQL, but are generally acceptable in all other SQL implementations.

When you issue a SQL statement, you can include one or more tabs, carriage returns, spaces, or comments anywhere a space occurs within the definition of the command. Thus, Oracle7 evaluates the following two statements in the same manner:

```
SELECT ENAME,SAL*12,MONTHS_BETWEEN(HIREDATE,SYSDATE) FROM EMP

SELECT  ENAME,
           SAL * 12,
           MONTHS_BETWEEN( HIREDATE, SYSDATE )
    FROM
        EMP
```

Case is insignificant in reserved words, keywords, identifiers and parameters. However, case is significant in text literals and quoted names. See the syntax description of *'text'* on page 2 – 15.

Tools Support

Most Oracle7 tools support all features of Oracle's SQL. However, not all tools support all features. This manual describes the complete functionality of SQL. If the Oracle7 tool that you are using does not support this complete functionality, you can find a discussion of the restrictions in the manual describing the tool, such as *PL/SQL User's Guide and Reference.*

CHAPTER

2

Elements of Oracle7 SQL

This chapter contains reference information on the basic elements of Oracle7 SQL. Before using any of the commands described in Chapter 4, "Commands," you should familiarize yourself with the concepts covered in this chapter:

- database objects
- object names and qualifiers
- referring to objects and parts
- literals
- text
- integer
- number
- datatypes
- nulls
- pseudocolumns
- comments

Database Objects

Schema Objects

A *schema* is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema. Schema objects can be created and manipulated with SQL and include the following types of objects.

- clusters
- database links
- database triggers*
- indexes
- packages*
- sequences
- snapshots*+
- snapshot logs*
- stored functions*
- stored procedures*
- synonyms
- tables
- views

* These objects are available only if PL/SQL is installed.

\+ These objects are available only if the distributed option is installed.

Non–Schema Objects

Other types of objects are also stored in the database and can be created and manipulated with SQL, but are not contained in a schema:

- profiles
- roles
- rollback segments
- tablespaces
- users

Most of these objects occupy space in the database. In this manual, each type of object is briefly defined in Chapter 4, "Commands" in the section describing the command that creates the database object. These commands begin with the keyword CREATE. For example, for the definition of a cluster, see the CREATE CLUSTER command on page 4 – 164. For an overview of database objects, see *Oracle7 Server Concepts*.

You must provide names for most types of objects when you create them. These names must follow the rules listed in the following sections.

Parts of Objects

Some objects are made up of parts that you must also name, such as:

- columns in a table or view
- integrity constraints on a table
- packaged procedures, packaged stored functions, and other objects stored within a package

Object Names and Qualifiers

This section tells provides:

- rules for naming objects and object location qualifiers
- guidelines for naming objects and qualifiers

Object Naming Rules

The following rules apply when naming objects:

1. Names must be from 1 to 30 characters long with these exceptions:
   - Names of databases are limited to 8 characters.
   - Names of database links can be as long as 128 characters.
2. Names cannot contain quotation marks.
3. Names are not case–sensitive
4. A name must begin with an alphabetic character from your database character set unless surrounded by double quotation marks.
5. Names can only contain alphanumeric characters from your database character set and the characters _, $, and #. You are strongly discourage from using $ and #.

   If your database character set contains multi–byte characters, It is recommended that each name for a user or a role contain at least one single–byte character.

   Names of database links can also contain periods (.) and ampersands (@).
6. A name cannot be an Oracle7 reserved word. The following list contains these reserved words. Words followed by an asterisk (*) are also ANSI reserved words.

Note: You cannot use special characters from European or Asian character sets in a database name, global database name, or database link names. For example, the umlaut is not allowed.

Reserved words

ACCESS	ELSE	MAXEXTENTS	SELECT*
ADD	EXCLUSIVE	MINUS	SESSION
ALL*	EXISTS*	MODE	SET*
ALTER		MODIFY	SHARE
AND*	FILE		SIZE
ANY*	FLOAT*	NOAUDIT	SMALLINT*
AS*	FOR*	NOCOMPRESS	START
ASC*	FROM*	NOT*	SUCCESSFUL
AUDIT		NOWAIT	SYNONYM
	GRANT*	NULL*	SYSDATE
BETWEEN*	GROUP*	NUMBER	
BY*			TABLE*
	HAVING*	OF*	THEN
CHAR*		OFFLINE	TO*
CHECK*	IDENTIFIED	ON*	TRIGGER
CLUSTER	IMMEDIATE	ONLINE	
COLUMN	IN*	OPTION*	UID
COMMENT	INCREMENT	OR*	UNION*
COMPRESS	INDEX	ORDER*	UNIQUE*
CONNECT	INITIAL		UPDATE*
CREATE*	INSERT*	PCTFREE	USER*
CURRENT*	INTEGER*	PRIOR	
	INTERSECT	PRIVILEGES*	VALIDATE
DATE	INTO*	PUBLIC*	VALUES*
DECIMAL	IS*		VARCHAR
DEFAULT*		RAW	VARCHAR2
DELETE*	LEVEL	RENAME	VIEW*
DESC*	LIKE*	RESOURCE	
DISTINCT*	LOCK	REVOKE	WHENEVER
DROP	LONG	ROW	WHERE*
		ROWID	WITH*
		ROWLABEL	
		ROWNUM	
		ROWS	

Depending on the Oracle product you plan to use to access a database 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 *PL/SQL User's Guide and Reference.*

7. The word DUAL should not be used as a name for an object or part. DUAL is the name of a dummy table frequently accessed by Oracle7 tools such as SQL*Plus and SQL*Forms.

8. The Oracle7 SQL language contains other keywords that have special meanings. Because these keywords are not reserved, you can also use them as names for objects and object parts. However, using them as names may make your SQL statements more difficult for you to read.

The following list contains keywords. Keywords marked with asterisks (*) are also ANSI reserved words. For maximum portability to other implementations of SQL, do not use the following words as object names.

Keywords

```
ADMIN
AFTER
ALLOCATE
ANALYZE
ARCHIVE
ARCHIVELOG
AUTHORIZATION*
AVG*

BACKUP
BEGIN*
BECOME
BEFORE
BLOCK
BODY

CACHE
CANCEL
CASCADE
CHANGE
CHARACTER*
CHECKPOINT
CLOSE*
COBOL*
COMMIT*
COMPILE
CONSTRAINT
CONSTRAINTS
CONTENTS
CONTINUE*
CONTROLFILE
COUNT*
CURSOR*
CYCLE

DATABASE
DATAFILE
DBA
DEC*
DECLARE*
DISABLE
DISMOUNT
DOUBLE*
DUMP
EACH
ENABLE
END*
ESCAPE*
EVENTS
EXCEPT
EXCEPTIONS
EXEC*
EXPLAIN
EXECUTE
EXTENT
EXTERNALLY

FETCH*
FLUSH
FREELIST
FREELISTS
FORCE
FOREIGN*
FORTRAN*
FOUND*
FUNCTION

GO*
GOTO*
GROUPS

INCLUDING
INDICATOR*
INITRANS
INSTANCE
INT*

KEY*

LANGUAGE*
LAYER
LINK
LISTS
LOGFILE

MANAGE
MANUAL
MAX*
MAXDATAFILES
MAXINISTANCES
MAXLOGFILES
MAXLOGHISTORY
MAXLOGMEMBERS
MAXTRANS
MAXVALUE
MIN*
MINEXTENTS
MINVALUE
MODULE*
MOUNT

NEXT
NEW
NOARCHIVELOG
NOCACHE
NOCYCLE
NOMAXVALUE
NOMINVALUE
NONE
NOORDER
NORESETLOGS
NORMAL
NOSORT
NUMERIC*

OFF
OLD
ONLY
OPTIMAL
OPEN*
OWN

PACKAGE
PARALLEL
PASCAL*
PCTINCREASE
PCTUSED
PLAN
PLI*
PRECISION*
PRIMARY*
PRIVATE
PROCEDURE*
PROFILE

QUOTA

READ
REAL*
RECOVER
REFERENCES*
REFERENCING
RESETLOGS
RESTRICTED
REUSE
ROLE
ROLES
ROLLBACK*

SAVEPOINT
SCHEMA*
SCN
SECTION*
SEGMENT
SEQUENCE
SHARED
SNAPSHOT
SOME*
SORT
SQLCODE*
SQLERROR*
STATEMENT_ID
STATISTICS
STOP
STORAGE
SUM*
SWITCH
SYSTEM

TABLES
TABLESPACE
TEMPORARY
THREAD
TIME
TRACING
TRANSACTION
TRIGGERS
TRUNCATE

UNDER
UNLIMITED
UNTIL
USE
USING

WHEN
WRITE
WORK*
```

9. A name must be unique across its namespace. Objects in the same namespace must have different names.

Figure 2 – 1 shows the namespaces for schema objects. Objects in the same namespace are grouped by solid lines. Because tables and views are in the same namespace, a table and a view in the same schema cannot have the same name. However, because tables and indexes are in different namespaces, a table and an index in the same schema can have the same name.

Each schema in the database has its own namespaces for the objects it contains. This means, for example, that two tables in different schemas are in different namespaces and can have the same name.

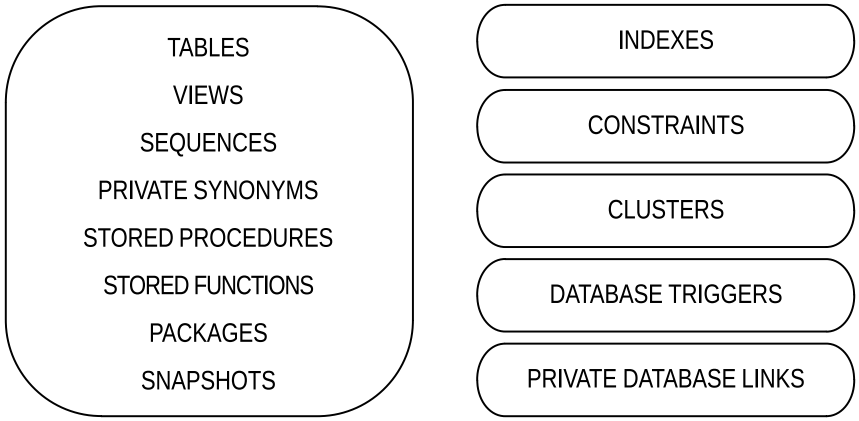


Figure 2 – 1 Namespaces For Schema Objects

Figure 2 – 2 shows the namespaces for other objects. Because the objects in these namespaces are not contained in schemas, these namespaces span the entire database.

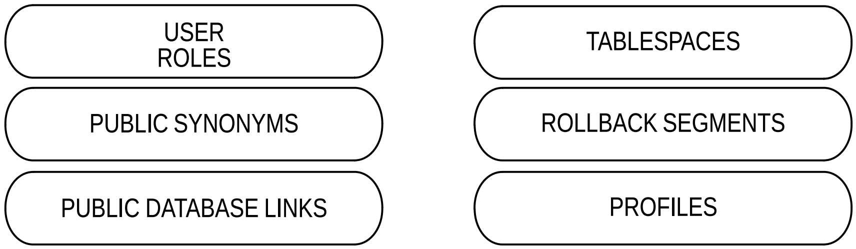


Figure 2 – 2 Namespaces For Other Objects

Columns in the same table or view cannot have the same name. However, columns in different tables or views can have the same name.

Procedures or functions contained in the same package can have the same name, provided that their arguments are not of the same number and datatypes. Creating multiple procedures or functions with the same name in the same package with different arguments is called *overloading* the procedure or function.

10. A name can be enclosed in double quotation marks. Such names can contain any combination of characters including spaces, ignoring rules 3 through 7 in this list. This exception is allowed for portability, but it is recommended that you do not break rules 3 through 7.

    Once you have given an object a name enclosed in double quotation marks, you must use double quotation marks whenever you refer to the object.

    You may want to enclose a name in double quotation marks for any of these reasons:

    - if you want it to contain spaces
    - if you want it to be case–sensitive
    - if you want it to begin with a character other than an alphabetic character, such as a numeric character
    - if you want it to contain characters other than alphanumeric characters and _, $, and #
    - if you want to use a reserved word as a name

    By enclosing names in double quotation marks, you can give the following names to different objects in the same namespace:

    ```
    emp
    "emp"
    "Emp"
    "EMP "
    ```

    Note that Oracle7 interprets the following names the same, so they cannot be used for different objects in the same namespace:

    ```
    emp
    EMP
    "EMP"
    ```

    If you give a user or password a quoted name, the name cannot contain lowercase letters.

    Database link names cannot be quoted.

Examples The following are valid examples of names:

```
ename
horse
scott.hiredate
"EVEN THIS & THAT!"
a_very_long_and_valid_name
```

Although column aliases, table aliases, usernames, or passwords are not objects or parts of objects, they must also follow these naming rules with these exceptions

- Column aliases and table aliases only exist for the execution of a single SQL statement and are not stored in the database, so rule 9 does not apply to them.
- Passwords do not have namespaces, so rule 9 does not apply to apply to them.
- Do not use quotation marks to make usernames and passwords case–sensitive. For additional rules for naming users and passwords, see the CREATE USER command on page 4 – 267.

Object Naming Guidelines

There are several helpful guidelines for naming objects and their parts:

- Use full, descriptive, pronounceable names (or well–known abbreviations).
- Use consistent naming rules.
- Use the same name to describe the same entity or attribute across tables.

When naming objects, balance the objective of keeping names short and easy to use with the objective of making name as long and descriptive as possible. When in doubt, choose the more descriptive name because the objects in the database may be used by many people over a period of time. Your counterpart ten years from now may have difficulty understanding a database with names like PMDD instead of PAYMENT_DUE_DATE.

Using consistent naming rules helps users understand the part that each table plays in your application. One such rule might be to begin the names of all tables belonging to the FINANCE application with FIN_.

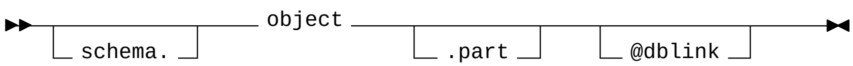
Use the same names to describe the same things across tables. For example, the department number columns of the EMP and DEPT tables are both named DEPTNO.

Referring to Objects and Parts

This section tells you how to refer to objects and their parts in the context of a SQL statement. This section shows you:

- the general syntax for referring to an object
- how Oracle7 resolves a reference to an object
- how to refer to objects in schemas other than your own
- how to refer to objects in remote databases

This syntax diagram shows the general syntax for referring to an object or a part:



where:

object is the name of the object.

schema is the schema containing the object. The *schema* qualifier allows you to refer to an object in a schema other than your own. Note that you must be granted privileges to refer to objects in other schemas. If you omit this qualifier, Oracle7 assumes that you are referring to an object in your own schema.

Only schema objects can be qualified with *schema*. Schema objects are shown in Figure 2 – 1 on page 2 – 6. Other objects, shown in Figure 2 – 2 on page 2 – 6, cannot be qualified with *schema* because they are not schema objects, except for public synonyms which can optionally be qualified with ”PUBLIC” (quotation marks required).

part is a part of the object. This identifier allows you to refer to a part of a schema object, such as a column of a table. Note that not all types of objects have parts.

dblink applies only to those using Oracle7 with the distributed option. This is the name of the database containing the object. The *dblink* qualifier allows you to refer to an object in a database other than your local database. If you omit this qualifier, Oracle7 assumes that you are referring to an object in your local database. Note that not all SQL statements allow you to access objects on remote databases.

You can include spaces around the periods separating the components of the reference to the object, but it is conventional to omit them.

How Oracle7 Resolves Object References

When you refer to an object in a SQL statement, Oracle7 considers the context of the SQL statement and locates the object in the appropriate namespace. If the named object cannot be found in the appropriate namespace, Oracle7 returns an error message. After locating the object, Oracle7 performs the statement's operation on the object.

The following example illustrates how Oracle7 resolves references to objects within SQL statements. Consider this statement that adds a row of data to a table identified by the name DEPT:

```
INSERT INTO dept
    VALUES (50, 'SUPPORT', 'PARIS')
```

Based on the context of the statement, Oracle7 determines that DEPT can be:

- a table in your own schema
- a view in your own schema
- a private synonym for a table or view
- a public synonym

Oracle7 always attempts to resolve an object reference within the namespaces in your own schema before considering namespaces outside your schema. In this example, Oracle7 attempts to resolve the name DEPT in these ways:

1. Oracle7 first attempts to locate the object in the namespace in your own schema containing tables, views, and private synonyms (see Figure 2 – 1 on page 2 – 6). If the object is a private synonym, Oracle7 locates the object for which the synonym stands. This object could be in your own schema, another schema, or on another database. The object could also be another synonym, in which case Oracle7 locates the object for which this synonym stands.

If the object is in the namespace, Oracle7 attempts to perform the statement on the object. In this example, Oracle7 attempts to add the row of data to DEPT. If the object is not of the correct type for the statement, Oracle7 returns an error message. In this example, DEPT must be a table, view, or a private synonym resolving to a table or view. If DEPT is a sequence, Oracle7 returns an error message.

2. If the object is not in the namespace searched in Step 1, Oracle7 searches the namespace containing public synonyms (see Figure 2 – 2 on page 2 – 6). If the object is in the namespace, Oracle7 attempts to perform the statement on it. If the object is not of the correct type for the statement, Oracle7 returns an error message. In this example, if DEPT is a public synonym for a sequence, Oracle7 returns an error message.

Referring to Objects in Other Schemas

To refer to objects in schemas other than your own, prefix the object name with the schema name:

```
schema.object
```

For example, this statement drops the EMP table in the schema SCOTT:

```
DROP TABLE scott.emp
```

Referring to Objects in Remote Databases

To refer to objects in databases other than your local database, follow the object name with the name of the database link to that database. A database link is a schema object that causes Oracle7 to connect to a remote database to access an object there. This section tells you:

- how to create database links
- how to use database links in your SQL statements

Creating Database Links

You can create a database link with the CREATE DATABASE LINK command described in Chapter 4, "Commands," of this manual. The command allows you to specify this information about the database link:

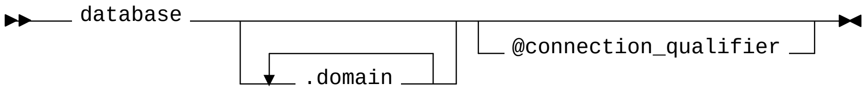
- the name of the database link
- the connect string to access the remote database
- the username and password to connect to the remote database

Oracle7 stores this information in the data dictionary.

Names When you create a database link, you must specify its name. The name of a database link can be as long as 128 bytes and can contain periods (.) and the special character @. In these ways, database link names are different from names of other types of objects.

The name that you give to a database link must correspond to the name of the database to which the database link refers and the location of that database in the hierarchy of database names. The following syntax diagram shows the form of the name of a database link:

dblink :=



where:

database specifies the name of the remote database to which the database link connects. The name of the remote database is specified by its initialization parameter DB_NAME.

domain specifies the domain of the remote database to which the database link connects. If you omit the *domain*s from the name of a database link, Oracle7 expands the name by qualifying *database* with the domain of your local database before storing it in the data dictionary. The domain of a database is specified by the value of its initialization parameter DB_DOMAIN.

connection_qualifier allows you to further qualify a database link. Using connection qualifiers, you can create multiple database links to the same database. For example, you can use connection qualifiers to create multiple database links to different instances of the Oracle7 Parallel Server that access the same database.

Username and Password The username and password are used by Oracle7 to connect to the remote database. The username and password for a database link are optional.

Database String The database string is the specification used by SQL*Net to access the remote database. For information on writing database connect strings, see the SQL*Net documentation for your specific network protocol. The database string for a database link is optional.

Referring to Database Links

Database links are available only to those using Oracle7 with the distributed option. When you issue a SQL statement that contains a database link, you can specify the database link name in one of these forms:

complete — is the complete database link name as stored in the data dictionary including the *database*, *domain*, and optional *connection_qualifier* components.

partial — contains the *database* and optional *connection_qualifier* components, but not the *domain* component.

Oracle7 performs these tasks before connecting to the remote database:

1. If the database link name specified in the statement is partial, Oracle7 expands the name to contain the domain of the local database (specified by the initialization parameter DB_DOMAIN).
2. Oracle7 first searches for a private database link in your own schema with the same name as the database link in the statement, and then, if necessary, searches for a public database link with the same name.

   2.1 Oracle7 always determines the username and password from the first matching database link (either private or public). If the first matching database link has an associated username and password, Oracle7 uses it. If it does not have an associated username and password, Oracle7 uses your current username and password.

   2.2 If the first matching database link has an associated database string, Oracle7 uses it. If not, Oracle7 searches for the next matching (public) database link. If there is no matching database link, or if no matching link has an associated database string, Oracle7 returns an error message.
3. Oracle7 uses the database string to access the remote database. After accessing the remote database, Oracle7 verifies that both of these conditions are true:
   - The name of the remote database (specified by its initialization parameter DB_NAME) must match the *database* component of the database link name.
   - The domain (specified by the initialization parameter DB_DOMAIN) of the remote database must match the domain component of the database link name.

If both of these conditions are true, Oracle7 proceeds with the connection, using the username and password chosen in step 2a. If not, Oracle7 returns an error message.

4. If the connection using the database string, username, and password is successful, Oracle7 attempts to access the specified object on the remote database using the rules for resolving object references and referring to objects in other schemas presented earlier in this section.

You can enable and disable Oracle7 resolution of names for remote objects using the initialization parameter GLOBAL_NAMES and the GLOBAL_NAMES parameter of the ALTER SYSTEM and ALTER SESSION commands.

You cannot use the USERENV('TERMINAL') variable in the WHERE clauses of INSERT, UPDATE, or DELETE statements that access remote objects, although you can do so in SELECT statements.

For more information on remote name resolution, see the "Database Administration" chapter of *Oracle7 Server Distributed Systems, Volume I.*

Literals

The terms literal and constant value are synonymous in this manual 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 quotation marks. The quotation marks allow Oracle7 to distinguish them from schema object names.

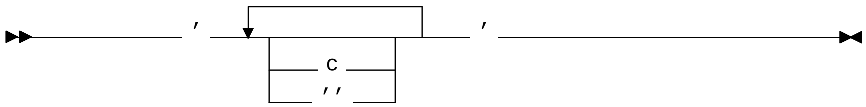
Many SQL statements and functions require you to specify character and numeric literal values. You can also specify literals as part of expressions and conditions. You can specify character literals with the '*text*' notation and numeric literals with the *integer* or *number* notation, depending on the context of the literal. The syntactic forms of these notations appear in the following sections.

Text

Purpose

To specify a text or character literal. You must use this notation to specify values whenever '*text*' or *char* appear in expressions, conditions, SQL functions, and SQL commands in other parts of this manual.

Syntax



Keywords and Parameters

c is any member of the user's character set, except a single quotation mark (').

'' are two single quotation marks. Because a single quotation mark is used to begin and end text literals, you must use two single quotation marks to represent one single quotation mark within a literal.

Usage Notes

A text literal must be enclosed in single quotation marks. This manual uses the terms text literal and character literal interchangeably.

Text literals have properties of both the CHAR and VARCHAR2 datatypes:

- Within expressions and conditions, Oracle7 treats text literals as though they have the datatype CHAR by comparing them using blank–padded comparison semantics.
- A text literal can have a maximum length of 2000 bytes.

Examples

```
'Hello'
'ORACLE.dbs'
'Jackie''s raincoat'
'09–MAR–92'
```

Related Topics

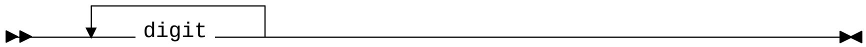
The syntax description of *expr* on page 3 – 73.

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 described in other parts of this manual.

Syntax



Keywords and Parameters

digit is one of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Usage Notes

An integer can store a maximum of 38 digits of precision.

Examples

```
7
255
```

Related Topics

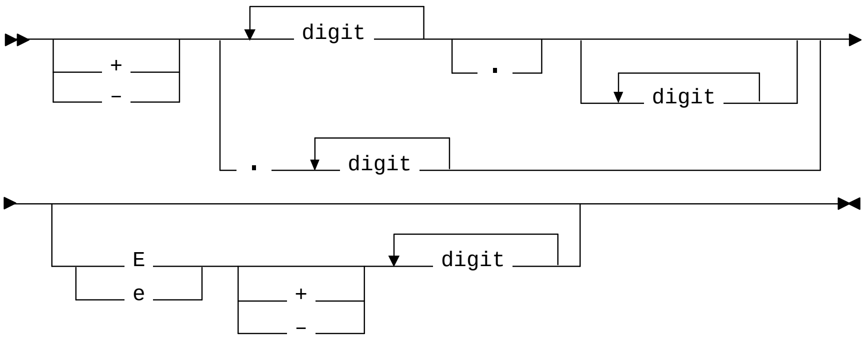
The syntax description of *expr* on page 3 – 73.

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 in other parts of this manual.

Syntax



Keywords and Parameters

+, – 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.

Usage Notes

A *number* can store a maximum of 38 digits of precision.

If you have established a decimal character other than a period (.) with the initialization parameter NLS_NUMERIC_CHARACTERS, you must specify numeric literals with '*text*' notation. In such cases, Oracle7 automatically converts the text literal to a numeric value.

For more information on this parameter, see *Oracle7 Server Reference.*

Examples

```
25
+6.34
0.5
25e–03
–1
```

Related Topics

The syntax description of *expr* on page 3 – 73.

Datatypes

Each literal or column value manipulated by Oracle7 has a datatype. A value's *datatype* associates a fixed set of properties with the value. These properties cause Oracle7 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, Oracle7 treats the '01–JAN–92' character string as a DATE value after verifying that it translates to a valid date.

Table 2 – 1 summarizes Oracle7 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. For information on external datatypes, including how Oracle7 converts between internal and external datatypes, see *Programmer's Guide to the Oracle Precompilers.*

Code	Internal Datatype	Description
1	VARCHAR2(*size*)	Variable length character string having maximum length *size* bytes. Maximum *size* is 2000, and minimum is 1. 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* can range from –84 to 127.
8	LONG	Character data of variable length up to 2 gigabytes, or 2^{31} –1 bytes.
12	DATE	Valid date range from January 1, 4712 BC to December 31, 4712 AD.
23	RAW(*size*)	Raw binary data of length *size* bytes. Maximum *size* is 255 bytes. You must specify *size* for a RAW value.
24	LONG RAW	Raw binary data of variable length up to 2 gigabytes.
69	ROWID (see note below)	Hexadecimal string representing the unique address of a row in its table. This datatype 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 and minimum *size* is 1 byte.
106	MLSLABEL	Binary format of an operating system label. This datatype is used with Trusted Oracle7.

Table 2 – 1 Internal Datatype Summary

The codes listed for the datatypes are used internally by Oracle7. The datatype code of a column is returned when you use the DUMP function.

> **Note:** The DESCRIBE embedded SQL command and the ODESCR call of the Oracle Call Interfaces (OCIs) returns a code of 11 for the 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. Oracle7 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. For a summary of the differences and compatibility issues, see Appendix C “Operating System –Specific Dependendies” of this manual.

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. Oracle7 subsequently ensures that all values stored in that column have this length. If you insert a value that is shorter than the column length, Oracle7 blank–pads the value to column length. If you try to insert a value that is too long for the column, Oracle7 returns an error.

The default for a CHAR column is 1 character and the maximum allowed is 255 characters. A zero–length string can be inserted into a CHAR column, but the column is blank–padded to 1 character when used in comparisons. For information on comparison semantics, see the section “Datatype Comparison Rules” on page 2 – 29.

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. Oracle7 subsequently stores each value in the column exactly as you specify it, provided it does not exceed the column’s maximum length. This maximum must be at least 1 byte, although the actual length of the string stored is permitted to be zero. If you try to insert a value that exceeds the specified length, Oracle7 returns an error.

You must specify a maximum length for a VARCHAR2 column. The maximum length of VARCHAR2 data is 2000 bytes. Oracle7 compares VARCHAR2 values using non–padded comparison semantics. For information on comparison semantics, see the section "Datatype Comparison Rules" on page 2 – 29.

VARCHAR Datatype

The VARCHAR datatype is currently synonymous with the VARCHAR2 datatype. It is recommended that you use VARCHAR2 rather than VARCHAR. In a future version of Oracle7, VARCHAR might be a separate datatype used for variable length character strings compared with different comparison semantics.

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^{-130} and 9.9...9 x 10^{125} (38 9s followed by 88 0s) with 38 digits of precision. If you specify an arithmetic expression whose value has a magnitude greater than or equal to 1.0 x 10^{126}, Oracle7 returns an error.

You can specify a fixed point number using the following form:

```
NUMBER(p,s)
```

where:

p	is the *precision*, or the total number of digits. Oracle7 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 specify an integer using the following form:

`NUMBER(p)`	is a fixed point number with precision *p* and scale 0. (Equivalent to NUMBER(p,0).)

You specify a floating point number using the following form:

`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 fixed point 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, Oracle7 returns an error. If a value exceeds the scale, Oracle7 rounds it.

The following examples show how Oracle7 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, Oracle7 returns an error message. If the value exceeds the scale, Oracle7 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

Oracle7 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 "NUMBER Datatype" on page 2 – 21. Oracle7 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.

LONG Datatype

LONG columns store variable length character strings containing up to 2 gigabytes, or 2^{31}–1 bytes. LONG columns have many of the characteristics of VARCHAR2 columns. You can use LONG columns to store long text strings. Oracle7 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 stored function cannot return a LONG value.
- Within a single SQL statement, all LONG columns, updated tables, and locked tables must be located on the same database.

Also, LONG columns cannot appear in certain SQL statements:

- CREATE SNAPSHOT

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
- UNIQUE clause of a SELECT statement
- the column datatype clause of a CREATE CLUSTER statement
- 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
- select lists in subqueries in INSERT statements

Triggers can use the LONG datatype in the following manner:

- A SQL statement within a trigger can insert data into a LONG column.
- If data from a LONG column can be converted to a constrained datatype (such as CHAR and VARCHAR2), a LONG column can be referenced in a SQL statement within a trigger. Note that the maximum length for these datatypes is 32 Kbytes.
- Variables in triggers cannot be declared using the LONG datatype.
- :NEW and :OLD cannot be used with LONG columns.

You can use the Oracle Call Interfaces to retrieve a portion of a LONG value from the database. See *Programmer's Guide to the Oracle Call Interface.*

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

To specify a date value, you must convert a character or numeric value to a data value with the TO_DATE function. Oracle7 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 a 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 Chapter 3 "Operators, Functions, Expressions, Conditions" of this manual.

Date Arithmetic

You can add and subtract number constants as well as other dates from dates. Oracle7 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.

Oracle7 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. For more information on date functions, see the section "Date Functions" on page 3 – 37.

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 Oracle7 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
```

RAW and LONG RAW Datatypes

The RAW and LONG RAW datatypes are used for data that is not to be interpreted (not converted when moving data between different systems) by Oracle. These datatypes are intended for binary data or byte strings. For example, LONG RAW can be used to store graphics, sound, documents, or arrays of binary data; the interpretation is dependent on the use.

RAW is a variable–length datatype like the VARCHAR2 character datatype, except that SQL*Net (which connects user sessions to the instance) and the Import and Export utilities do not perform character conversion when transmitting RAW or LONG RAW data. In contrast, SQL*Net and Import/Export automatically convert CHAR, VARCHAR2, and LONG data between the database character set to the user session character set (set by the NLS_LANGUAGE parameter of the ALTER SESSION command), if the two character sets are different.

When Oracle automatically converts RAW or LONG RAW data to and from CHAR data, the binary data is represented in hexadecimal form with one hexadecimal character representing every four bits of RAW data. For example, one byte of RAW data with bits 11001011 is displayed and entered as 'CB'.

LONG RAW data cannot be indexed, but RAW data can be indexed.

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 the section "Pseudocolumns" on page 2 – 38. You can also create tables and clusters that contain actual columns having the ROWID datatype. Oracle7 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).

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 Oracle7 to mediate access to information. You can also define columns with this datatype if you are using the standard Oracle7 Server. For more information on Trusted Oracle7, including this datatype and labels, see *Trusted Oracle7 Server Administrator's Guide.*

ANSI, DB2, and SQL/DS Datatypes

SQL commands that create tables and clusters can also both ANSI datatypes and datatypes from IBM's products SQL / DS and DB2. Oracle7 creates columns with Oracle7 datatypes based on the conversions defined in Table 2 – 2 and Table 2 – 3.

ANSI SQL Datatype	Oracle7 Datatype
CHARACTER(*n*) CHAR(*n*)	CHAR(*n*)
CHARACTER VARYING(*n*) CHAR VARYING(*n*)	VARCHAR(*n*)
NUMERIC(*p*,*s*) DECIMAL(*p*,*s*)	NUMBER(*p*,*s*)
INTEGER INT SMALLINT	NUMBER(38)
FLOAT(*b*) [2] DOUBLE PRECISION [3] REAL [4]	NUMBER

Table 2 – 2 ANSI Datatypes Converted to Oracle7 Datatypes

SQL/DS or DB2 Datatype	Oracle7 Datatype
CHARACTER(*n*)	CHAR(*n*)
VARCHAR(*n*)	VARCHAR(*n*)
LONG VARCHAR(*n*)	LONG
DECIMAL(*p*,*s*) [1]	NUMBER(*p*,*s*)
INTEGER SMALLINT	NUMBER(38)
FLOAT(*b*) [2]	NUMBER

Table 2 – 3 SQL / DS and DB2 Datatypes Converted to Oracle7 Datatypes

[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 Oracle7 datatype:

- GRAPHIC
- LONG VARGRAPHIC
- VARGRAPHIC
- TIME
- TIMESTAMP

Note that data of type TIME and TIMESTAMP can also be expressed as Oracle7 DATE data.

Datatype Comparison Rules

This section describes how Oracle7 compares values of each datatype.

Number Values

A larger value is considered greater than a smaller one. All negative numbers are less than zero and 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. Table 2 – 4 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.

The results of blank–padded and non–padded comparisons is shown in Table 2 – 4. Usually, the results of blank–padded and non–padded comparisons are the same. However, note the exception highlighted in bold in Table 2 – 4 where blanks are considered less than any character, which is true in most character sets.

Blank–Padded	Non–Padded
'ab' > 'aa'	'ab' > 'aa'
'ab' > 'a␣'	'ab' > 'a␣'
'ab' > 'a'	'ab' > 'a'
'ab' = 'ab'	'ab' = 'ab'
'a␣' = 'a'	**'a␣' > 'a'**

Table 2 – 4 Results of Comparisons with Blank–Padded and Non–Padded Comparison Semantics

Blank–Padded Comparison Semantics If the two values have different lengths, Oracle7 first adds blanks to the end of the shorter one so their lengths are equal. Oracle7 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. Oracle7 uses blank–padded comparison semantics only when both values in the comparison are either expressions of datatype CHAR, text literals, or values returned by the USER function.

Non–Padded Comparison Semantics Oracle7 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. Oracle7 uses non–padded comparison semantics whenever one or both values in the comparison have the datatype VARCHAR2.

Single Characters

Oracle7 compares single characters according to their numeric values in the database character set. One character is greater than another if it has a greater numeric value than the other in the character set. In Table 2 – 4, blanks are considered less than any character, which is true in most character sets.

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

Portions of the ASCII and EBCDIC character sets appear in Table 2 – 5 and Table 2 – 6. Note that uppercase and lowercase letters are not equivalent. Also, note that the numeric values for the characters of a character set may not match the linguistic sequence for a particular language.

ASCII Character Set

Table 2 – 5 lists 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	_
44	,	96	‘
45	–	97–122	a–z
46	.	123	{
47	/	124	|
48–57	0–9	125	}
58	:	126	~

Table 2 – 5 ASCII Character Set

EBCDIC Character Set

Table 2 – 6 lists a common portion of the EBCDIC character set.

Decimal value	Symbol	Decimal value	Symbol
64	blank	108	%
74	¢	109	_
75	.	110	>
76	<	111	?
77	(	122	:
78	+	123	#
79	|	124	@
80	&	125	’
90	!	126	=
91	$	127	”
92	*	129–137	a–i
93	)	145–153	j–r
94	;	162–169	s–z
95	¬	193–201	A–I
96	–	209–217	J–R
97	/	226–233	S–Z

Table 2 – 6 EBCDIC Character Set

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, Oracle7 supports both implicit and explicit conversion of values from one datatype to another.

Implicit Data Conversion

Oracle7 automatically converts a value from one datatype to another when such a conversion makes sense. Oracle7 performs datatype conversions in these cases:

- When an INSERT or UPDATE statement assigns a value of one datatype to a column of another, Oracle7 converts the value to the datatype of the column.
- When you use a SQL function or operator with an argument with a datatype other than the one it accepts, Oracle7 converts the argument to the accepted datatype.
- When you use a comparison operator on values of different datatypes, Oracle7 converts one of the expressions to the datatype of the other.

Example I The text literal '10' has datatype CHAR. Oracle7 implicitly converts it to the NUMBER datatype if it appears in a numeric expression as in the following statement:

```
SELECT sal + '10'
    FROM emp
```

Example II When a condition compares a character value and a NUMBER value, Oracle7 implicitly converts the character value to a NUMBER value, rather than converting the NUMBER value to a character value. in the following statement, Oracle7 implicitly converts '7936' to 7936:

```
SELECT ename
    FROM emp
    WHERE empno = '7936'
```

If the character value is too short to fit the entire number value, the number value is rounded.

Example III In the following statement, Oracle7 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'
```

Example IV In the following statement, Oracle7 implicitly converts the text literal '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. Table 2 – 7 shows SQL functions that explicitly convert a value from one datatype to another.

TO / FROM	CHAR	NUMBER	DATE	RAW	ROWID
CHAR	unnecessary	TO_NUMBER	TO_DATE	HEXTORAW	CHARTOROWID
NUMBER	TO_CHAR	unnecessary	TO_DATE (number,'J')		
DATE	TO_CHAR	TO_CHAR (date,'J')	unnecessary		
RAW	RAWTOHEX			unnecessary	
ROWID	ROWIDTOCHAR				unnecessary

Table 2 – 7 SQL Functions for Datatype Conversion

For information on these functions, see the section "Conversion Functions" on page 3 – 42.

Note: Note that Table 2 – 7 does not show conversions from LONG and LONG RAW values because it is impossible to specify LONG and LONG RAW values in cases in which Oracle7 can perform implicit datatype conversion. For example, LONG and LONG RAW values cannot appear in expressions with functions or operators. For information on the limitations on LONG and LONG RAW datatypes, see the section "LONG Datatype" on page 2 – 23.

Implicit vs. Explicit Data Conversion

It is recommended 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.

Nulls

If a column in a row has no value, then 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 not known or when a value would not be meaningful.

Oracle7 currently treats a character value with a length of zero as null. However, this may not continue to be true in future versions of Oracle7.

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 Oracle7 considers two nulls to be equal when evaluating a DECODE expression. For information on the DECODE syntax, see the section "Expr" on page 3 – 73.

Nulls in Conditions

A condition that evaluates to UNKNOWN acts almost like FALSE. For example, a SELECT statement with a condition in the WHERE clause that evaluates to UNKNOWN will return no rows. However, a condition evaluating to UNKNOWN differs from FALSE in that further operations on an UNKNOWN condition evaluation will evaluate to UNKNOWN. Thus, NOT FALSE evaluates to TRUE, but NOT UNKNOWN evaluates to UNKNOWN.

Table 2 – 8 shows examples of various evaluations involving nulls in conditions. If the conditions evaluating to UNKNOWN were used in a WHERE clause of a SELECT statement, then no rows would be returned for that query.

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
NULL	a = 10	UNKNOWN
NULL	a != 10	UNKNOWN

Table 2 – 8 Conditions Containing Nulls

For the truth tables showing the results of logical expressions containing nulls, see Table 3 – 6, Table 3 – 7, and Table 3 – 8 beginning on page 3 – 11.

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

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
```

For more information on referring to database links, see the section “Referring to Objects in Remote Databases” on page 2 – 11.

If you are using Trusted Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

If you are using Trusted Oracle7 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 that is not contained in a subquery, snapshot or view
- the SELECT list of a subquery in an INSERT 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 in a DELETE, SELECT, or UPDATE statement
- 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 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 initialize the sequence with NEXTVAL.

If a statement contains more than one reference to NEXTVAL for a sequence, Oracle7 increments the sequence once and returns the same value for all occurrences of NEXTVAL. If a statement contains references to both CURRVAL and NEXTVAL, Oracle7 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. For information on sequences, see the CREATE SEQUENCE command on page 4 – 224.

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 sub–orders 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)
```

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 node that has children. A *leaf node* is any node without children. Figure 2 – 3 shows the nodes of an inverted tree with their LEVEL values.

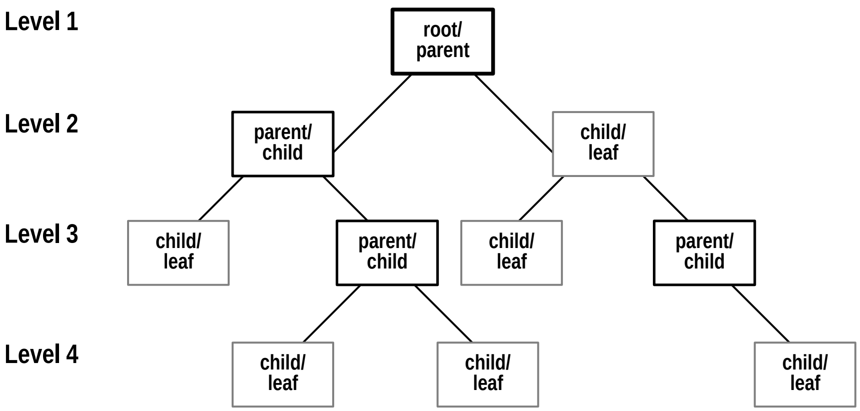


Figure 2 – 3 Hierarchical Tree

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 on page 4 – 405.

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)

Usually, 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. For information on the ROWID datatype, see the section "ROWID Datatype" on page 2 – 27.

ROWID values have several important uses:

- They are the fastest way to access 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, Oracle7 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
```

ROWNUM

For each row returned by a query, the ROWNUM pseudocolumn returns a number indicating the order in which Oracle7 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
```

Oracle7 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 Oracle7 to use an index to access the data, Oracle7 may retrieve the rows in a different order than without the index, so the ROWNUMs may differ 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 makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

Comments

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 Oracle7 optimizer. The optimizer uses these hints to choose an execution plan for the statement. For more information on hints, see the "Tuning SQL Statements" chapter of *Oracle7 Server Tuning*.

Note that you cannot use these styles of comments between SQL statements in a SQL script. You can use the Server Manager or SQL*Plus REMARK command for this purpose. For information on these commands, see *Oracle Server Manager User's Guide* or *SQL*Plus User's Guide and Reference*.

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 + NLV(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 described in Chapter 4, "Commands" of this manual. Comments associated with schema objects are stored in the data dictionary.

CHAPTER

3

Operators, Functions, Expressions, Conditions

This chapter describes methods of manipulating individual data items. For example, standard arithmetic operators such as addition and subtraction are discussed as well as less common functions such as absolute value or string length. Topics include:

- operators
- SQL functions
- user functions
- format models
- expressions
- conditions

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. The tables in the following sections of this chapter list SQL operators.

Unary and Binary Operators

There are two general classes of operators:

unary — A *unary* operator operates on only one operand. A unary operator typically appears with its operand in this format:

```
operator operand
```

binary — A *binary* operator operates on two operands. A binary operator appears with its operands in this format:

```
operand1 operator operand2
```

Other operators with special formats accept more than two operands. If an operator is given a null operator, the result is always null. The only operator that does not follow this rule is concatenation (||).

Precedence

An important property of an operator is its precedence. *Precedence* is the order in which Oracle7 evaluates different operators in the same expression. When evaluating an expression containing multiple operators, Oracle7 evaluates operators with higher precedence before evaluating those with lower precedence. Oracle7 evaluates operators with equal precedence from left to right within an expression.

Table 3 – 1 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	

Table 3 – 1 SQL Operator Precedence

You can use parentheses in an expression to override operator precedence. Oracle7 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

In the following expression multiplication has a higher precedence than addition, so Oracle7 first multiplies 2 by 3 and then adds the result to 1.

```
1+2*3
```

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. Table 3 – 2 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`
+ –	Adds, subtracts. These are binary operators.	`SELECT sal + comm FROM emp` `WHERE SYSDATE – hiredate > 365`

Table 3 – 2 Arithmetic Operators

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. For more information on comments within SQL statements, see the section "Comments" on page 2 – 43.

Character Operators

Character operators are used in expressions to manipulate character strings. Table 3 – 3 lists the single character operator.

Operator	Purpose	Example
||	Concatenates character strings.	`SELECT 'Name is ' || ename FROM emp`

Table 3 – 3 Character Operators

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 section "Character Datatypes" on page 2 – 20.

On most platforms, the concatenation operator is two solid vertical bars, as shown in Table 3 – 3. However, some IBM platforms use broken vertical bars for this operator. When moving SQL script files between systems having different character sets, such as between ASCII and EBCDIC, vertical bars might not be translated into the vertical bar required by the target Oracle7 environment. Because it may be difficult or impossible to control translation performed by operating system or network utilities, the CONCAT character function is provided as an alternative to the vertical bar operator. Its use is recommended in applications that will be moved to environments with differing character sets.

Although Oracle7 treats zero–length character strings as nulls, concatenating a zero–length character string with another operand always results in the other operand, so null can only result from the concatenation of two null strings. However, this may not continue to be true in future versions of Oracle7. 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) );

Table created.

INSERT INTO tab1 (col1,  col2,     col3,     col4)
        VALUES   (’abc’, ’def   ’, ’ghi   ’, ’jkl’);

1 row created.

SELECT col1||col2||col3||col4 ”Concatenation”
        FROM tab1;

Concatenation
------------------------
abcdef   ghi   jkl
```

Comparison Operators

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. For information on conditions, see the section “Condition” on page 3 – 78. Table 3 – 4 lists comparison operators.

Operator	Purpose	Example

=	Equality test.	`SELECT *`
		`FROM emp`
		`WHERE sal = 1500`

!=
^=
¬=
< >
	Inequality test. All forms of the inequality operator may not be available on all platforms.	`SELECT *`
		`FROM emp`
		`WHERE sal != 1500`

>
 	
<
	“Greater than” and “less than” tests.	`SELECT *`
		`FROM emp`
		`WHERE sal > 1500`
		
		`SELECT *`
		`FROM emp`
		`WHERE sal < 1500`

Table 3 – 4 Comparison Operators

Operator	Purpose	Example
>= <=	"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') 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')
ANY SOME	Compares a value to each value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, >=. Evaluates to FALSE if the query returns no rows.	SELECT * FROM emp WHERE sal = ANY (SELECT sal FROM emp WHERE deptno = 30)
ALL	Compares a value to every value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, >=. Evaluates to TRUE if the query returns no rows.	SELECT * FROM emp WHERE sal >= ALL (1400, 3000)

Table 3 – 4 Comparison Operators

Operator	Purpose	Example

`[NOT] BETWEEN x AND y`	[Not] greater than or equal to *x* and less than or equal to *y*.	`SELECT *`
		`FROM emp`
		`WHERE sal`
		`BETWEEN 2000`
		`AND 3000`

`EXISTS`	TRUE if a subquery returns at least one row.	`SELECT dname, deptno`
		`FROM dept`
		`WHERE EXISTS`
		`(SELECT *`
		`FROM emp`
		`WHERE`
		`dept.deptno`
		`= emp.deptno)`

`x [NOT] LIKE y [ESCAPE 'z']`	TRUE if *x* does [not] match the pattern *y*. Within *y*, the character "%" matches any string of zero or more characters except null. The character "_" matches any single character. Any character, excepting percent (%) and underbar (_) may follow ESCAPE; a wilcard character will be treated as a literal if preceded by the escape character.	See the section "LIKE Operator" beginning on page 3 – 8.
		
		`SELECT *`
		`FROM tab1`
		`WHERE col1 LIKE`
		`'A_C/%E%' ESCAPE '/'`

`IS [NOT] NULL`	Tests for nulls. This is the only operator that should be used to test for nulls. See the section "Nulls" on page 2 – 36.	`SELECT dname, deptno`
		`FROM emp`
		`WHERE comm IS NULL`

Table 3 – 4 Comparison Operators

NOT IN Operator

All rows evaluate to UNKNOWN (and no rows are returned) if any item in the list following a NOT IN operation is null. For example, the following statement returns the string 'TRUE':

```
SELECT 'TRUE'
    FROM emp
    WHERE deptno NOT IN (5,15)
```

However, the following statement returns no rows:

```
SELECT 'TRUE'
    FROM emp
    WHERE deptno NOT IN (5,15,null)
```

The above 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:

```
LIKE condition (Form VII) ::=

>>── char1 ─┬───────┬─ LIKE char2 ─┬──────────────┬──><
            └─ NOT ─┘              └─ 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 Oracle7 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. Note that blank padding is *not* used for LIKE comparisons.

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 uses the = operator, rather than the LIKE operator, to find the salaries of all employees with the name ’SM%’:

```
SELECT sal
   FROM emp
   WHERE ename = 'SM%'
```

The following query finds the salaries of all employees with the name ’SM%’. Oracle7 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 usually use special characters that Oracle7 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 Sensitivity and Pattern Matching 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%'
```

Pattern Matching on Indexed Columns When LIKE is used to search an indexed column for a pattern, Oracle7 can use the index to improve the statement’s performance if the leading character in the pattern is not “%” or “_”. In this case, Oracle7 can scan the index by this leading character. If the first character in the pattern is “%” or “_”, the index cannot improve the query’s performance because Oracle7 cannot scan the index.

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.

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 “_” then Oracle7 interprets this character literally in the pattern, rather than as a special pattern matching character.

Example III To search for any employees with the pattern ’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 Oracle7 to interpret the underscore literally, rather than as a special pattern matching character.

Patterns Without % If a pattern does not contain the “%” character, the condition can only be TRUE if both operands have the same length.

Example IV Consider the definition of this table and the values inserted into it:

```
CREATE TABLE freds (f CHAR(6), v VARCHAR2(6))
INSERT INTO freds VALUES ('FRED', 'FRED')
```

Because Oracle7 blank–pads CHAR values, the value of F is blank–padded to 6 bytes. V is not blank–padded and has length 4. Table 3 – 5 shows conditions that evaluate to TRUE and FALSE.

Logical Operators

A logical operator combines the results of two component conditions to produce a single result based on them or to invert the result of a single condition. Table 3 – 5 lists logical operators.

Operator	Function	Example

NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN	`SELECT * FROM emp WHERE NOT (job IS NULL)`
		
		`SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000)`

AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE. Otherwise returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10`
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE. Otherwise returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10`

Table 3 – 5 Logical Operators

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 earning more than $1000 a month are returned:

```
SELECT *
   FROM emp
   WHERE hiredate < TO_DATE('01-JAN-1984', 'DD-MON-YYYY')
     AND sal > 1000
```

NOT Operator

Table 3 – 6 shows the result of applying the NOT operator to a condition.

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

Table 3 – 6 NOT Truth Table

AND Operator

Table 3 – 7 shows the results of combining two expressions with AND.

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Table 3 – 7 AND Truth Table

OR Operator

Table 3 – 8 shows the results of combining two expressions with OR.

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Table 3 – 8 OR Truth Table

Set Operators

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries. Table 3 – 9 lists SQL set operators.

Operator	Returns
UNION	All 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.

Table 3 – 9 Set Operators

All set operators have equal precedence. If a SQL statement contains multiple set operators, Oracle7 evaluates them from the left to right if no parentheses explicitly specify another order. To comply with emerging SQL standards, a future version of Oracle7 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.

The corresponding expressions in the select lists of the component queries of a compound query must match in number and datatype. If component queries 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.

Examples

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
```

The following examples combine the two query results with each of the set operators.

UNION Example

The following statement combines the results with the UNION operator, which eliminates duplicate selected rows:

The following statement shows how datatype must match when columns do not exist in one or the other table:

```
SELECT part, partnum, to_date(null) date_in
    FROM orders_list1
UNION
SELECT part, to_null(null), date_in
    FROM orders_list2

PART       PARTNUM DATE_IN
---------- ------- --------
SPARKPLUS  3323165
SPARKPLUG          10/24/98
FUEL PUMP  3323162
FUEL PUMP          12/24/99
TAILPIPE   1332999
TAILPIPE           01/01/01
CRANKSHAFT 9394991
CRANKSHAFT         09/12/02
SELECT part
    FROM orders_list1
UNION
SELECT part
    FROM orders_list2

PART
----------
SPARKPLUG
FUEL PUMP
TAILPIPE
CRANKSHAFT
```

UNION ALL Example

The following 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.

INTERSECT Example

The following 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
```

MINUS Example The following 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
```

Other Operators

Table 3 – 10 lists other SQL operators.

Operator	Purpose	Example
(+)	Indicates that the preceding column is the outer join column in a join. See the section "Outer Joins" on page 4 – 425.	`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. See the section "Hierarchical Queries" on page 4 – 411.	`SELECT empno, ename, mgr FROM emp CONNECT BY PRIOR empno = mgr`

Table 3 – 10 Other SQL Operators

SQL Functions

A SQL function is similar to an operator in that it manipulates data items and returns a result. SQL 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 SQL function with an argument of a datatype other than the datatype expected by the SQL function, Oracle7 implicitly converts the argument to the expected datatype before performing the SQL function. See the section "Data Conversion" on page 2 – 34.

If you call a SQL function with a null argument, the SQL function automatically returns null. The only SQL functions that do not follow this rule are CONCAT, DECODE, DUMP, NVL, and REPLACE.

SQL functions should not be confused with user functions written in PL/SQL. User functions are described on page 3 – 57.

In the syntax diagrams for SQL functions, arguments are indicated with their datatypes following the conventions described in the Preface of this manual.

SQL functions are of these general types:

- single row (or scalar) functions
- group (or aggregate) functions

The two types of SQL 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, Oracle7 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. Oracle7 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, Oracle7 applies group functions in the select list to all the rows in the queried table or view. You use group functions in the HAVING clause to eliminate groups from the output based on the results of the group functions, rather than on the values of the individual rows of the queried table or view. For more information on the GROUP BY and HAVING clauses, see the section "GROUP BY Clause" on page 4 – 416 and the section "HAVING Clause" on page 4 – 417.

Single Row Functions

The following functions are single row functions grouped together by the datatypes of their arguments and return values.

Number Functions

Number functions accept numeric input and return numeric values. This section lists the SQL number functions. 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, and TANH are accurate to 36 decimal digits. The transcendental functions ACOS, ASIN, ATAN, and ATAN2 are accurate to 30 decimal digits.

ABS

Syntax `ABS(n)`

Purpose Returns the absolute value of *n*

Example

```
SELECT ABS(–15) ”Absolute”
    FROM DUAL

  Absolute
----------
        15
```

ACOS

Syntax `ACOS(n)`

Purpose Returns the arc cosine of *n*. Inputs are in the range of –1 to 1, and outputs are in the range of 0 to pi and are expressed in radians.

Example

```
SELECT ACOS(.3) ”Arc_Cosine”
    FROM DUAL

Arc_Cosine
----------
1.26610367
```

ASIN

Syntax `ASIN(n)`

Purpose Returns the arc sine of *n*. Inputs are in the range of –1 to 1, and outputs are in the range of –pi/2 to pi/2 and are expressed in radians.

Example

```
SELECT ASIN(.3) "Arc_Sine"
    FROM DUAL

  Arc_Sine
----------
.304692654
```

ATAN

Syntax `ATAN(n)`

Purpose Returns the arc tangent of *n*. Inputs are in an unbounded range, and outputs are in the range of –pi/2 to pi/2 and are expressed in radians.

Example

```
SELECT ATAN(.3) "Arc_Tangent"
    FROM DUAL

Arc_Tangent
_----------
 .291456794
```

ATAN2

Syntax `ATAN2(n, m)`

Purpose Returns the arc tangent of *n* and *m*. Inputs are in an unbounded range, and outputs are in the range of –pi to pi, depending on the signs of x and y, and are expressed in radians. Atan2(x,y) is the same as atan2(x/y)

Example

```
SELECT ATAN2(.3, .2) "Arc_Tangent2"
    FROM DUAL

Arc_Tangent2
------------
  .982793723
```

CEIL

Syntax `CEIL(n)`

Purpose Returns smallest integer greater than or equal to *n*.

Example

```
SELECT CEIL(15.7) "Ceiling"
    FROM DUAL

   Ceiling
----------
        16
```

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
```

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
```

EXP

Syntax `EXP(n)`

Purpose Returns *e* raised to the *n*th power; e = 2.71828183 ...

Example

```
SELECT EXP(4) "e to the 4th power"
    FROM DUAL

e to the 4th power
------------------
          54.59815
```

FLOOR

Syntax `FLOOR(n)`

Purpose Returns largest integer equal to or less than *n*.

Example

```
SELECT FLOOR(15.7) ”Floor”
   FROM DUAL

    Floor
----------
        15
```

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
```

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
```

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 using the MOD function with this formula:

```
m – n * FLOOR(m/n)
```

Example The following 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
–11   4       –3                  1
 11  –4        3                 –1
–11  –4       –3                 –3
```

POWER

Syntax

```
POWER(m, n)
```

Purpose Returns *m* raised to the *n*th 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
```

ROUND

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.

Example

```
SELECT ROUND(15.193,1) "Round"
   FROM DUAL

     Round
----------
      15.2
```

Example

```
SELECT ROUND(15.193,-1) "Round"
   FROM DUAL

     Round
----------
        20
```

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
```

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
```

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
```

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
```

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
```

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
```

TRUNC

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
```

Character Functions

Single row character functions accept character input and can return both character and number values.

Character Functions Returning Character Values

This section lists character functions that return 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, Oracle7 truncates it and returns the result without an error message.

CHR

Syntax `CHR(n)`

Purpose Returns the character having the binary equivalent to *n* in the database character set.

Example

```
SELECT CHR(67)||CHR(65)||CHR(84) "Dog"
        FROM DUAL

Dog
---
CAT
```

CONCAT

Syntax `CONCAT(char1, char2)`

Purpose Returns *char1* concatenated with *char2*. This function is equivalent to the concatenation operator (||). For information on this operator, see the section "Character" on page 3 – 4.

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') "Capitals"
   FROM DUAL

Capitals
--------
The Soap
```

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
```

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
```

LTRIM

Syntax

```
LTRIM(char1,n [,set])
```

Purpose Removes characters from the left of *char*, with all the leftmost characters that appear in *set* removed; *set* defaults to a single blank. Oracle7 begins scanning *char* from its first character and removes all characters that appear in set until reaching a character not in set and then returns the result.

Example

```
SELECT LTRIM('xyxXxyLAST WORD','xy') "LTRIM example"
        FROM DUAL

LTRIM example
-------------
Xxy LAST WORD
```

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. For information on sort sequences, see *Oracle7 Server Reference*.

Example

```
SELECT NLS_INITCAP('ijsland', 'NLS_SORT = XDutch') "Capitalized"
    FROM DUAL

Capital
-------
IJsland
```

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 = XGerman')
"Lowercase"
   FROM DUAL

Lower
-----
città
```

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('große', 'NLS_SORT = XGerman') "Uppercase"
   FROM DUAL

Upper
-----
GROSS
```

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
```

RPAD

Syntax

```
RPAD(char1, n [,char2])
```

Purpose Returns *char1*, right–padded to length *n* with *char2*, replicated as many 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,12,'ab') "RPAD example"
    FROM emp
    WHERE ename = 'TURNER'

RPAD example
------------
TURNERababab
```

RTRIM

Syntax

```
RTRIM(char [,set]
```

Purpose Returns *char*, with all the rightmost characters that appear in *set* removed; *set* defaults to a single blank. RTRIM works similarly to LTRIM.

Example

```
SELECT RTRIM('TURNERyxXxy','xy') "RTRIM e.g."
    FROM DUAL

RTRIM e.g
---------
TURNERyxX
```

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, as follows:

- retain the first letter of the string and remove the following letters: a, e, h, i, o, w, y
- assign the numbers to the remaining letters as follows:

```
0 = a, e, h, i, o, w, y
1 = b, f, p, v
2 = c, e, g, j, k, q, s, x, z
3 = d, t = 3
4 = l
5 = m, n
r = 6
```

- if two or more of the numbers are in sequences, remove all but the first
- return the first four bytes padded with 0

Example

```
SELECT ename
    FROM emp
    WHERE SOUNDEX(ename)
        = SOUNDEX('SMYTHE')

ENAME
----------
SMITH
```

SUBSTR

Syntax

```
SUBSTR(char, m [,n])
```

Purpose

Returns a portion of *char*, beginning at character *m*, *n* characters long. If *m* is 0, it is treated as 1. If *m* is positive, Oracle7 counts from the beginning of *char* to find the first character. If *m* is negative, Oracle7 counts backwards from the end of *char*. If *n* is omitted, Oracle7 returns all characters to the end of *char*. If n is less than 1, a null is returned.

Floating point numbers passed as arguments to *substr* are automatically converted to integers.

Example

```
SELECT SUBSTR('ABCDEFG',3.1,4) "Subs"
    FROM DUAL

Subs
----
CDEF

SELECT SUBSTR('ABCDEFG',-5,4) "Subs"
    FROM DUAL

Subs
----
CDEF
```

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.

Floating point numbers passed as arguments to *substrb* are automatically converted to integers.

Example Assume a double–byte database character set:

```
SELECT SUBSTRB('ABCDEFG',5,4.2) "Substring with bytes"
   FROM DUAL

Sub
---
CD
```

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 an empty string for *to* to remove all characters in *from* from the return value. Oracle7 interprets the empty string as null, and if this function has a null argument, it returns null.

Examples The following 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') "Licence"
   FROM DUAL

Translate example
-----------------
9XXX999
```

The following 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
```

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
```

Character Functions Returning Number Values

This section lists character functions that return number values.

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 no similar EBCDIC character function.

Example

```
SELECT ASCII('Q')
    FROM DUAL

ASCII('Q')
----------
        81
```

INSTR

Syntax `INSTR(char1,char2[,n[,m]])`

Purpose Searches *char1* beginning with its *n*th character for the *m*th occurrence of *char2* and returns the position of the character in *char1* that is the first character of this occurrence. If *n* is negative, Oracle7 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 Oracle7 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 *n*th 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
```

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

```
SELECT INSTRB('CORPORATE FLOOR','OR',5,2)
"Instring in bytes"
   FROM DUAL

Instring in bytes
-----------------
               27
```

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
```

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
```

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*. For information on sort sequences, see the "National Language Support" chapter of *Oracle7 Server Reference*..

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')
```

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.

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
```

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.

Example

```
SELECT SYSDATE,
    LAST_DAY(SYSDATE) "Last",
    LAST_DAY(SYSDATE) - SYSDATE "Days Left"
    FROM DUAL

SYSDATE   Last       Days Left
--------- --------- ----------
10-APR-95 30-APR-95         20

SELECT TO_CHAR(
    ADD_MONTHS(
        LAST_DAY(hiredate),5),
        'DD-MON-YYYY') "Five months"
    FROM emp
    WHERE ename = 'MARTIN'

Five months
-----------
28-FEB-1982
```

```
SELECT TO_CHAR(ADD_MONTHS(hiredate,1),
               'DD-MON-YYYY') "Next month"
       FROM emp
       WHERE ename = 'SMITH'

Next month
-----------
17-JAN-1981
```

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 Oracle7 calculates the fractional portion of the result based on a 31–day month and considers the difference in time components of *d1* and *d2*.

Example

```
SELECT MONTHS_BETWEEN(
    TO_DATE('02-02-1995','MM-DD-YYYY'),
    TO_DATE('01-01-1995','MM-DD-YYYY') ) "Months"
    FROM DUAL

    Months
----------
1.03225806
```

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 ADT	Atlantic Standard or Daylight Time
BST BDT	Bering Standard or Daylight Time
CST CDT	Central Standard or Daylight Time
EST EDT	Eastern Standard or Daylight Time
GMT	Greenwich Mean Time
HST HDT	Alaska–Hawaii Standard Time or Daylight Time.
MST MDT	Mountain Standard or Daylight Time
NST	Newfoundland Standard Time
PST PDT	Pacific Standard or Daylight Time
YST 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
```

ROUND

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.

For details on ROUND and TRUNC, see the section "ROUND and TRUNC" on page 3 – 23.

Example

```
SELECT ROUND(TO_DATE('27-OCT-92'),'YEAR')
"FIRST OF THE YEAR"
    FROM DUAL

FIRST OF THE YEAR
-----------------
01-JAN-93
```

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.
```

TRUNC

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. See the next section "ROUND and TRUNC."

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
```

ROUND and TRUNC

Table 3 – 11 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.

Format Model	Rounding or Truncating Unit
CC SCC	Century
SYYYY YYYY YEAR SYEAR YYY YY Y	Year (rounds up on July 1)
IYYY IY 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
HH HH12 HH24	Hour
MI	Minute

Table 3 – 11 Date Format Models for the ROUND and TRUNC Date Functions

The starting day of the week used by the format models DAY, DY, and D is specified implicitly by the initialization parameter NLS_TERRITORY. For information on this parameter, see the "National Language Support" chapter of *Oracle7 Server Reference*.

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. This section lists the SQL conversion functions.

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
```

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('Groß', 'WE8HP', 'WE8DEC')
"Conversion"
   FROM DUAL

Conversion
----------
Groß
```

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
```

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
```

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
```

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. For information on date formats, see the section “Format Models” on page 3 – 59.

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
```

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.

For more information on this function, see *Trusted Oracle7 Server Administrator's Guide*.

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. For information on number formats, see the section "Format Models" on page 3 – 59.

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 = territory '
```

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 quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit '*nlsparams*' or any one of the parameters, this function uses the default parameter values for your session.

Example I

```
SELECT TO_CHAR(–10000,'L99G999D99MI') "Amount"
    FROM DUAL
Amount
--------------------
         $10,000.00–
```

Note how the output above is blank padded to the left of the currency symbol.

Example II

```
SELECT TO_CHAR(-10000,'L99G999D99MI',
    'NLS_NUMERIC_CHARACTERS = '',.''
    NLS_CURRENCY = ''AusDollars'' ') "Amount"
    FROM DUAL
Amount
--------------------
AusDollars10.000,00-
```

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 an integer. For information on date formats, see the section "Format Models" on page 3 – 64.

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.

For information on date formats, see page 3 – 64.

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
```

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. For more information on this function, see *Trusted Oracle7 Server Administrator's Guide*.

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.

Example

```
UPDATE emp
 SET sal = sal +
   TO_NUMBER('100.00', '9G999D99')
 WHERE ename = 'BLAKE'
```

The '*nlsparams*' has the same purpose in this function as in the TO_CHAR function for number conversion.

Example

```
SELECT TO_NUMBER('–AusDollars100','L9G999D99',
   ' NLS_NUMERIC_CHARACTERS = '',.''
     NLS_CURRENCY           = ''AusDollars''
   ') "Amount"
    FROM DUAL

    Amount
----------
      –100
```

TO_SINGLE_BYTE

Syntax

```
TO_SINGLE_BYTE(char)
```

Purpose: Returns *char* with all of its multi–byte character 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.

Other Functions

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'.

For the datatype corresponding to each code, see Table 2 – 1 on page 2 – 19.

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
```

GREATEST

Syntax

```
GREATEST(expr [,expr] ...)
```

Purpose

Returns the greatest of the list of *expr*s. All *expr*s after the first are implicitly converted to the datatype of the first *exprs* before the comparison. Oracle7 compares the *expr*s 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
```

GREATEST_LB

Syntax

```
GREATEST_LB(label [,label] ...)
```

Purpose

Returns the greatest lower bound of the list of *label*s. 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.

For the definition of greatest lower bound and examples of this function, see *Trusted Oracle7 Server Administrator's Guide.*

LEAST

Syntax

```
LEAST(expr [,expr] ...)
```

Purpose

Returns the least of the list of *expr*s. All *expr*s after the first are implicitly converted to the datatype of the first *expr* before the comparison. Oracle7 compares the *expr*s 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
```

LEAST_UB

Syntax

```
LEAST_UB(label [,label] ...)
```

Purpose

Returns the least upper bound of the list of *label*s. Each *label* must have datatype MLSLABEL or be a quoted literal in the default label format. The return value has datatype RAW MLSLABEL. For the definition of least upper bound and examples of this function, see *Trusted Oracle7 Server Administrator's Guide.*

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, Oracle7 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
```

UID

Syntax `UID`

Purpose Returns an integer that uniquely identifies the current user.

USER

Syntax `USER`

Purpose Returns the current Oracle7 user with the datatype VARCHAR2. Oracle7 compares values of this function with blank–padded comparison semantics.

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$BQUIGLEY                           46
```

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 USERENV in the condition of a CHECK constraint. The argument *option* can have any of these values:

’OSDBA’ returns ’TRUE’ if you currently have the OSDBA role enabled and ’FALSE’ if you do not.

’LABEL’ returns your current session label. This option is only applicable for Trusted Oracle7. For more information on this option, see *Trusted Oracle7 Server Administrator’s Guide.*

'LANGUAGE'	returns the language and territory currently used by your session along with the database character set in this form: `language_territory.characterset`
'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. In a distributed environment, this is supported only for remote SELECTs, not for remote INSERTs, UPDATEs, or DELETEs.
'SESSIONID'	returns your auditing session identifier. You cannot use this option in distributed SQL statements. To use this keyword in USERENV, the initialization parameter AUDIT_TRAIL must be set to TRUE.
'ENTRYID'	returns available auditing entry identifier. You cannot use this option in distributed SQL statements. To use this keyword in USERENV, the initialization parameter AUDIT_TRAIL must be set to TRUE.
'CLIENT_INFO'	Returns the value of the client_info field of the current session, as the last value set by the dbms_application_info.set_client_info procedure.
'LANG'	Returns the ISO abbreviation for the language name, a shorter form than the existing 'LANGUAGE' parameter.

Example

```
SELECT USERENV('LANGUAGE') "Language"
    FROM DUAL

Language
----------------------------------------------------
AMERICAN_AMERICA.WE8DEC
```

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. For a discussion of the differences between group functions and single–row functions, see the section “Functions” on page 3 – 17.

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.

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.

AVG

Syntax `AVG([DISTINCT|ALL] n)`

Purpose Returns average value of *n*.

Example

```
SELECT AVG(sal) "Average"
   FROM emp

   Average
----------
2077.21429
```

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
----------
        18
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*. For the definitions of greatest lower bound and example usage, see *Trusted Oracle7 Server Administrator's Guide*.

LUB

Syntax

```
LUB([DISTINCT|ALL] label)
```

Purpose

Returns the least upper bound of *label*.

The return values have datatype MLSLABEL. For the definitions of greatest least upper bound and example usage, see *Trusted Oracle7 Server Administrator's Guide*.

MAX

Syntax

```
MAX([DISTINCT|ALL] expr)
```

Purpose Returns maximum value of *expr*.

Example

```
SELECT MAX(sal) "Maximum"
   FROM emp

   Maximum
----------
      5004
```

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.

STDDEV

Syntax

```
STDDEV([DISTINCT|ALL] x)
```

Purpose Returns standard deviation of *x*, a number. Oracle7 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
```

SUM

Syntax

```
SUM([DISTINCT|ALL] n)
```

Purpose Returns sum of values of *n*.

Example

```
SELECT SUM(sal) "Total"
   FROM emp

     Total
----------
     29081
```

VARIANCE

Syntax

```
VARIANCE([DISTINCT|ALL]x)
```

Purpose Returns variance of *x*, a number. Oracle7 calculates the variance of *x* using this formula:

$$\frac{\sum_{i=1}^{n} x_i^{2} - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^{2}}{n-1}$$

where:

x_i	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
```

User Functions

You can write your own user functions in PL/SQL to provide functionality that is not available in SQL or SQL functions. User functions are used in a SQL statement anywhere SQL functions can be used; that is, wherever expression can occur.

For example, user functions can be used in the following:

- the select list of a SELECT command
- the condition of a WHERE clause
- the CONNECT BY, START WITH, ORDER BY, and GROUP BY clauses
- the VALUES clause of an INSERT command
- the SET clause of an UPDATE command

For a complete description on the creation and usage of user functions, see *Oracle7 Server Application Developer's Guide*.

Prequisites

User functions must be created as top–level PL/SQL functions or declared with a package specification before they can be named within a SQL statement. User functions are created as top–level PL/SQL functions by using the CREATE FUNCTION statement described on page 4 – 188. Packaged functions are specified with a package with the CREATE PACKAGE statement described on page 4 – 198.

To call a packaged user function, you must declare the RESTRICT_REFERENCES pragma in the package specification.

Privileges Required

To use a user function in a SQL expression, you must own or have EXECUTE privilege on the user function. To query a view defined with a user function, you must have SELECT privileges on the view. No separate EXECUTE privileges are needed to select from the view.

Restrictions on User Functions

User functions cannot be used in situations that require an unchanging definition. Thus, a user function:

- cannot be used in a CHECK constraint clause of a CREATE TABLE or ALTER TABLE command
- cannot be used in a DEFAULT clause of a CREATE TABLE or ALTER TABLE command
- cannot contain OUT or IN OUT parameters
- cannot update the database
- cannot read or write package state if the function is a remote function
- cannot use the *parallelism_clause* in SQL commands in the function if the function alters package state
- cannot update variables defined in the function unless the function is a local function and is used in a SELECT list, VALUES clause of an INSERT command, or SET clause of an UPDATE command

Name Precedence

With PL/SQL, the names of database columns take precedence over the names of functions with no parameters. For example, if user SCOTT creates the following two objects in his own schema:

```
CREATE TABLE emp(new_sal NUMBER, ...)
CREATE FUNCTION new_sal RETURN NUMBER IS ,,,;
```

then in the following two statements, the reference to NEW_SAL refers to the column EMP.NEW_SAL:

```
SELECT new_sal FROM emp;
SELECT emp.new_sal FROM emp;
```

To access the function NEW_SAL, you would enter:

```
SELECT scott.new_sal FROM emp;
```

Example I

For example, to call the TAX_RATE user function from schema SCOTT, execute it against the SS_NO and SAL columns in TAX_TABLE, and place the results in the variable INCOME_TAX, specify the following:

```
SELECT scott.tax_rate (ss_no, sal)
    INTO income_tax
    FROM tax_table
    WHERE ss_no = tax_id;
```

Example II Listed below are sample calls to user functions that are allowed in SQL expressions.

```
circle_area (radius)
payroll.tax_rate (empno)
scott.payroll.tax_rate (dependent, empno)@ny
```

Naming Conventions

If only one of the optional schema or package names is given, the first identifier can be either a schema name or a package name. For example, to determine whether PAYROLL in the reference PAYROLL.TAX_RATE is a schema or package name, Oracle proceeds as follows:

- check for the PAYROLL package in the current schema
- if a PAYROLL package is not found, look for a schema name PAYROLL that contains a top–level TAX_RATE function; if no such function is found, an error message is returned
- if the PAYROLL package is found in the current schema, look for a TAX_RATE function in the PAYROLL package; if no such function is found, an error message is returned

You can also refer to a stored top–level function using any synonym that you have defined for it.

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 Oracle7 to use to return a value from the database to you
- to specify the format for a value you have specified for Oracle7 to store in the database

Note that a format model does not change the internal representation of the value in the database.

This section describes how to use:

- number format models
- date format models
- format model modifiers

Changing the Return Format

You can use a format model to specify the format for Oracle7 to use to return values from the database to you.

Example I

The following 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, Oracle7 returns the commissions with leading dollar signs, commas every three digits, and two decimal places. Note that the TO_CHAR function returns null for all employees with null in the COMM column.

Example II

The following 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, Oracle7 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 (Oracle7 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 III

The following 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. If a positive value is extremely large and cannot be represented in the specified format, then the infinity sign (~) replaces the value. Likewise, if a negative value is extremely small and cannot be represented by the specified format, then the negative infinity sign replaces the value (–~).

Number Format Elements

A number format model is composed of one or more number format elements. Examples are shown in Table 3 – 17 on page 3 – 71. Table 3 – 12 lists the elements of a number format model.

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 number format model must not begin with a comma (,). A group separator or comma cannot appear to the right of a decimal character or period in a number format model.

Element	Example	Description
9	9999	Return value with the specified number of digits with a leading space if positive. Return value with the specified number of digits with a leading minus if negative. Leading zeros are blank, except for a zero value, which returns a zero for the integer part of the fixed point number.
0	0999 9990	Return leading zeros. Return trailing zeros.
$	$9999	Return value with a leading dollar sign.
B	B9999	Return blanks for the integer part of a fixed point number when the integer part is zero (regardless of “0”s in the format model).
MI	9999MI	Return negative value with a trailing minus sign ”–”. Returns positive value with a trailing blank.
S	S9999 9999S	Return negative value with a leading minus sign “–”. Return positive value with a leading plus sign “+”. Return negative value with a trailing minus sign “–”. Return positive value with a trailing plus sign “+”.
PR	9999PR	Return negative value in <angle brackets>. Return positive value with a leading and trailing blank.
D	99D99	Return a decimal point (that is, a period “.”) in the specified position.
G	9G999	Return a group separator in the position specified.
C	C999	Return the ISO currency symbol in the specified position.

Table 3 – 12 Number Format Elements

Element	Example	Description
L	L999	Return the local currency symbol in the specified position.
, (comma)	9,999	Return a comma in the specified position.
. (period)	99.99	Return a decimal point (that is, a period “.”) int the specified position.
V	999V99	Return a value multiplied by 10^n (and if necessary, round it up), where *n* is the number of “9”s after the “V”.
EEEE	9.9EEEE	Return a value using in scientific notation.
RN rn	RN	Return a value as Roman numerals in uppercase. Rerturn a value as Roman numerals in lowercase. Value can be an integer between 1 and 3999.
FM	FM90.9	Returns a value with no leading or trailing blanks.

Table 3 – 12 (continued) Number Format Elements

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 of last position of a number format model.

The characters returned by some of these format elements are specified by initialization parameters. Table 3 – 13 lists these elements and parameters.

Element	Description	Initialization Parameter
D	Decimal character	NLS_NUMERIC_CHARACTER
G	Group separator	NLS_NUMERIC_CHARACTER
C	ISO currency symbol	NLS_ISO_CURRENCY
L	Local currency symbol	NLS_CURRENCY

Table 3 – 13 Number Format Element Values Determined by Initialization Parameters

The characters returned by these format elements can also be implicitly specified by the initialization parameter NLS_TERRITORY.

You can also change the characters returned by these format elements for your session with the ALTER SESSION command. For information on this command, see page 4 – 55.

For information on these parameters, see *Oracle7 Server Reference*. You can also change the default date format for your session with the ALTER SESSION command. For information on this command, see page 4 – 55.

Date Format Models

You can use date format models in the following 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

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.

For information on these parameters, see *Oracle7 Server Reference*. You can also change the default date format for your session with the ALTER SESSION command. For information on this command, see page 4 – 55.

Maximum Length

The total length of a date format model cannot exceed 22 characters.

Date Format Elements

A date format model is composed of one or more date format elements as listed in Table 3 – 14. For input format models, format items cannot appear twice and also format items that represent similar information cannot be combined. For example, you cannot use 'SYYYY' and 'BC' in the same format string.

Element	Meaning
- / , . ; : ”text”	Punctuation and quoted text is reproduced in the result.
AD A.D.	AD indicator with or without periods.
AM A.M.	Meridian indicator with or without periods.
BC B.C.	BC indicator. with or without periods.
CC SCC	Century; “S” prefixes BC dates with “–”.
D	Day of week (1–7).
DAY	Name of day, padded with blanks to length of 9 characters.
DD	Day of month (1–31).
DDD	Day of year (1–366).
DY	Abbreviated name of day.
IW	Week of year (1–52 or 1–53) based on the ISO standard.
IYY IY I	Last 3, 2, or 1 digit(s) of ISO year.
IYYY	4–digit year based on the ISO standard.
HH HH12	Hour of day (1–12).
HH24	Hour of day (0–23).
J	Julian day; the number of days since January 1, 4712 BC. Number specified with ’J’ must be integers.
MI	Minute (0–59).
MM	Month (01–12; JAN = 01)
MONTH	Name of month, padded with blanks to length of 9 characters.
MON	Abbreviated name of month.
RM	Roman numeral month (I–XII; JAN = I).

Table 3 – 14 Date Format Elements

Element	Meaning
Q	Quarter of year (1, 2, 3, 4; JAN–MAR = 1)
RR	Last 2 digits of year; for years in other countries.
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.
W	Week of month (1–5) where week 1 starts on the first day of the month and ends on the seventh.
PM P.M.	Meridian indicator with and without periods.
SS	Second (0–59).
SSSSS	Seconds past midnight (0–86399).
Y YYY	Year with comma in this position.
YEAR SYEAR	Year, spelled out; “S” prefixes BC dates with “–”.
YYYY SYYYY	4–digit year; “S” prefixes BC dates with “–”.
YYY YY Y	Last 3, 2, or 1 digit(s) of year.

Table 3 – 14 (continued) Date Format Elements

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 Oracle7. 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.

For information on these initialization parameters, see *Oracle7 Server Reference*.

ISO Standard Date Format Elements

Oracle7 calculates the values returned by the date format elements IYYY, IYY, IY, I, and IW according to the ISO standard. For information on the differences between these values and those returned by the date format elements YYYY, YYY, YY, Y, and WW, see the “National Language Support” chapter of *Oracle7 Server Reference*.

The 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. Table 3 – 15 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.

Table 3 – 15 The RR Date Element Format

The following example demonstrates the behavior of the RR date format element.

Example IV Assume these queries are issued before 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

Table 3 – 16 lists suffixes that can be added to date format elements:

Suffix	Meaning	Example Element	Example Value
TH	Ordinal Number	DDTH	4TH
SP	Spelled Number	DDSP	FOUR
SPTH or THSP	Spelled, ordinal number	DDSPTH	FOURTH

Table 3 – 16 Date Format Element Suffixes

When you add one of these suffixes to a date format element, the return value is always in English.

> **Note:** Date suffixes are only valid on output and cannot be used to insert a date into the database.

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 double quotation marks.

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 a 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 re–enabled for the portion following its third, and so on.

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 and trailing 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 zero 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.

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, Oracle7 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, Oracle7 returns an error message.

Example V Table 3 – 17 shows the results of the following query for different values of *number* and ’*fmt*’:

```
SELECT TO_CHAR(number, 'fmt')
    FROM dual
```

number	'fmt'	Result
–1234567890	9999999999S	'1234567890–'
0	99.99	' 0.00'
+0.1	99.99	' .10'
–0.2	99.99	' –.20'
0	90.99	' 0.00'
+0.1	90.99	' .10'
–0.2	90.99	' –0.20'
0	9999	' 0'
1	9999	' 1'
0	B9999	' '
1	B9999	' 1'
0	B90.99	' '
+123.456	999.999	' 123.456'
–123.456	999.999	'–123.456'
+123.456	FM999.009	'123.456'
+123.456	9.9EEEE	' 1.2E+02'
+1E+123	9.9EEEE	' 1.0E+123'
+123.456	FM9.9EEEE	'1.23E+02'
+123.45	FM999.009	'123.45'
+123.0	FM999.009	'123.00'
+123.45	L999.99	' $123.45'
+123.45	FML99.99	'$123.45'
+1234567890	9999999999S	'1234567890+'

Table 3 – 17 Results of Example Number Conversions

Example VI The following 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, 1995
```

Note that the following 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    , 1995
```

You can include a single quotation mark in the return value by placing two consecutive single quotation marks in the format model.

Example VII The following statement places a single quotation mark in the return value by using a date format model that includes two consecutive single quotation marks:

```
SELECT TO_CHAR(SYSDATE, 'fmDay''"s Special"') Menu
    FROM DUAL

Menu
-----------------
Tuesday's Special
```

Two consecutive single quotation marks can also be used for the same purpose within a character literal in a format model.

Example VIII Table 3 – 18 shows whether the following statement meets the matching conditions for different values of *char* and *'fmt'* using FX:

```
UPDATE table
    SET date_column = TO_DATE(char, 'fmt')
```

char	***'fmt'***	**Match or Error?**
'15/ JAN /1993'	'DD–MON–YYYY'	Match
' 15! JAN % /1993'	'DD–MON–YYYY'	Match
'15/JAN/1993'	'FXDD–MON–YYYY'	Error
'15–JAN–1993'	'FXDD–MON–YYYY'	Match
'1–JAN–1993'	'FXDD–MON–YYYY'	Error
'01–JAN–1993'	'FXDD–MON–YYYY'	Error
'1–JAN–1993'	'FXFMDD–MON–YYYY'	Match

Table 3 – 18 Matching Character Data and Format Models with the FX Format Model Modifier

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.

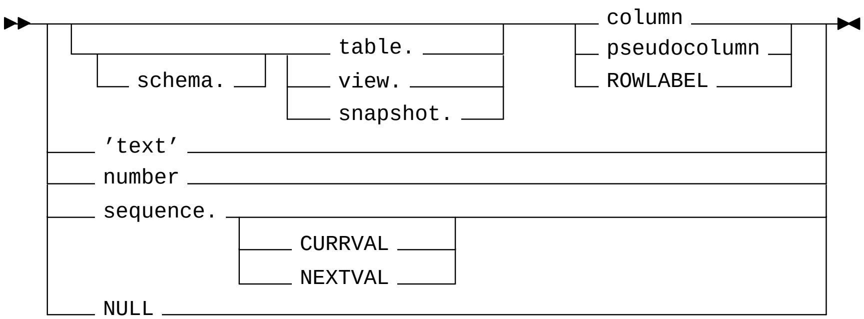
Syntax

Expressions have several forms. Oracle7 does not accept all forms of expressions in all parts of all SQL commands. The description of each command in Chapter 4 “Commands” of this manual documents the restrictions on the expressions in the command.

Form I

A column, pseudocolumn, constant, sequence number, or NULL.

expr (Form I) ::=



In addition to the schema of a user, *schema* can also be “PUBLIC” (double quotation marks required), in which case it must qualify a public synonym for a table, view, or snapshot. Qualifying a public synonym with “PUBLIC” is only supported in Data Manipulation Language commands, not Data Definition Language commands.

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. For more information on pseudocolumns, see the section “Pseudocolumns” on page 2 – 38.

ROWLABEL is a column automatically created by Trusted Oracle7 in every table in the database. If you are using Trusted Oracle7, the expression ROWLABEL returns the row’s label. If you are not using Trusted Oracle7, the expression ROWLABEL always returns NULL. For information on using labels and ROWLABEL, see *Trusted Oracle7 Server Administrator’s Guide.*

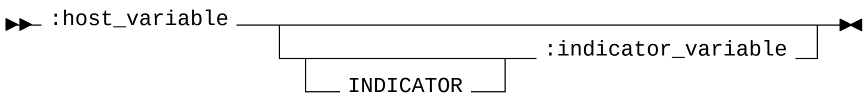
Examples

```
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.

expr (Form II) ::=



Examples

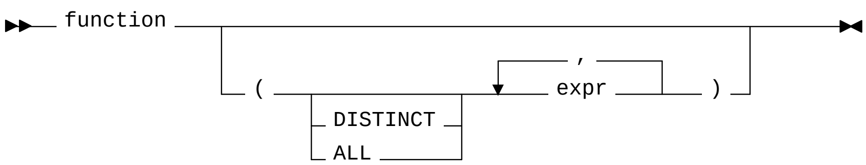
```
:employee_name INDICATOR :employee_name_indicator_var

:department_location
```

Form III

A call to a SQL function.

expr (Form III) ::=



For information on SQL functions, see the section "SQL Functions" on page 3 – 17.

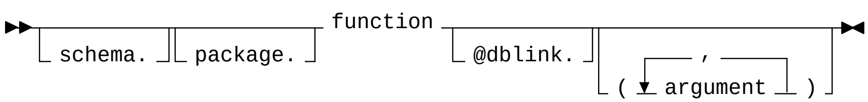
Examples

```
LENGTH('BLAKE')
ROUND(1234.567*43)
SYSDATE
```

Form IV

A call to a user function.

expr (Form IV) ::=



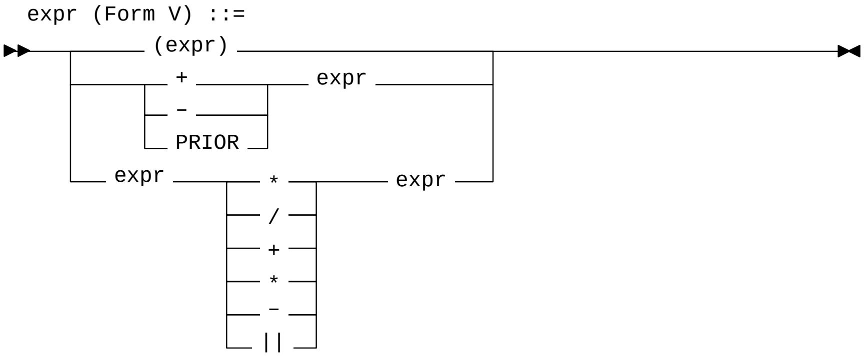
For information on user functions, see the section "User Functions" on page 3 – 57.

Examples

```
circle_area(radius)
payroll.tax_rate(empno)
scott.payrol.tax_rate(dependents, empno)@ny
```

Form V

A combination of other expressions.



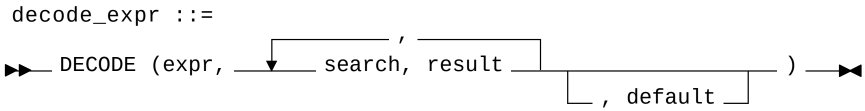
Note that some combinations of functions are inappropriate and are rejected. For example, the LENGTH function is inappropriate within a group function.

Examples

```
('CLARK' || 'SMITH')
LENGTH('MOOSE') * 57
SQRT(144) + 72
my_fun(TO_CHAR(sysdate,'DD–MMM–YY')
```

Decoded Expression

An expression using the special DECODE syntax:



To evaluate this expression, Oracle7 compares *expr* to each *search* value one by one. If *expr* is equal to a *search*, Oracle7 returns the corresponding *result*. If no match is found, Oracle7 returns *default*, or, if *default* is omitted, returns null. If *expr* and *search* contain character data, Oracle7 compares them using non–padded comparison semantics. For information on these semantics, see the section "Datatype Comparison Rules" on page 2 – 29.

The *search*, *result*, and *default* values can be derived from expressions. Oracle7 evaluates each *search* value only before comparing it to *expr*, rather than evaluating all *search* values before comparing any of them with *expr*. Consequently, Oracle7 never evaluates a *search* if a previous *search* is equal to *expr*.

Oracle7 automatically converts *expr* and each *search* value to the datatype of the first *search* value before comparing. Oracle7 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 Oracle7 converts the return value to the datatype VARCHAR2. For information on datatype conversion, see the section "Data Conversion" on page 2 – 34.

In a DECODE expression, Oracle7 considers two nulls to be equivalent. If *expr* is null, Oracle7 returns the *result* of the first *search* that is also null.

The maximum number of components in the DECODE expression, including *expr*, *search*es, *result*s, and *default* is 255.

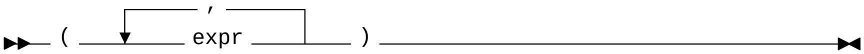
Example This expression decodes the value DEPTNO. If DEPTNO is 10, the expression evaluates to 'ACCOUNTING'; if DEPTNO is 20, it evaluates to 'RESEARCH'; etc. If DEPTNO is not 10, 20, 30, or 40, the expression returns 'NONE'.

```
DECODE (deptno,10, 'ACCOUNTING',
               20, 'RESEARCH',
               30, 'SALES',
               40, 'OPERATION',
                   'NONE')
```

List of Expressions

A parenthesized list of expressions.

expr_list ::=



An expression list can contain up to 254 expressions.

Examples

```
(10, 20, 40)
('SCOTT', 'BLAKE', 'TAYLOR')
(LENGTH('MOOSE') * 57, -SQRT(144) + 72, 69)
```

Usage Notes

An *expression* is a combination of one or more values, operators, and SQL functions that evaluates to a value. An expression generally assumes the datatype of its components.

This simple expression evaluates to 4 and has datatype NUMBER (the same datatype as its components):

```
2*2
```

The following expression is an example of a more complex expression that uses both functions and operators. The expression adds seven days to the current date, removes the time component from the sum, and converts the result to CHAR datatype:

```
TO_CHAR(TRUNC(SYSDATE+7))
```

You can use expressions in any of these places:

- the select list of the SELECT command
- a condition of the WHERE and HAVING clauses
- the CONNECT BY, START WITH, and ORDER BY clauses
- the VALUES clause of the INSERT command
- the SET clause of the UPDATE command

For example, you could use an expression in place of the quoted string 'smith' in this UPDATE statement SET clause:

```
SET ename = 'smith'
```

This SET clause has the expression LOWER(ENAME) instead of the quoted string 'smith':

```
SET ename = LOWER(ename)
```

Related Topics

The section “Functions” on page 3 – 17
The syntax description of *'text'* on page 2 – 15
The syntax description of *number* on page 2 – 17

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 in Chapter 4 “Commands” of this manual.

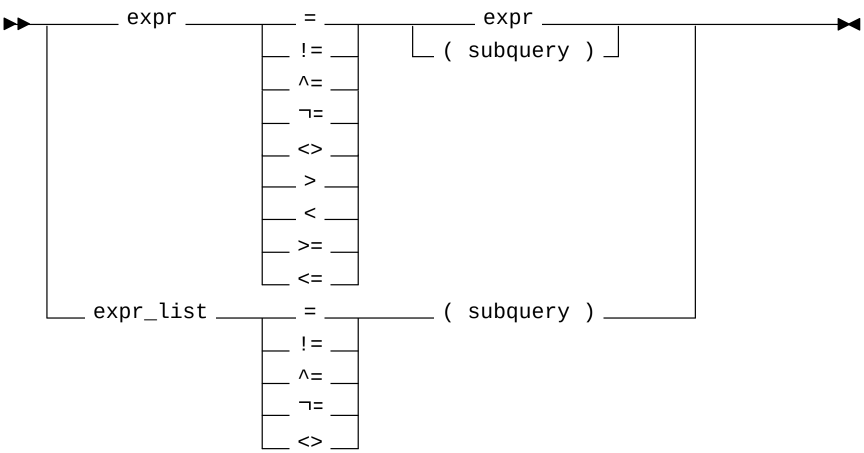
Syntax

Conditions can have several forms. The description of each command in Chapter 4 “Commands” of this manual documents the restrictions on the conditions in the command.

Form I

A comparison with expressions or subquery results.

condition (Form I ::=)

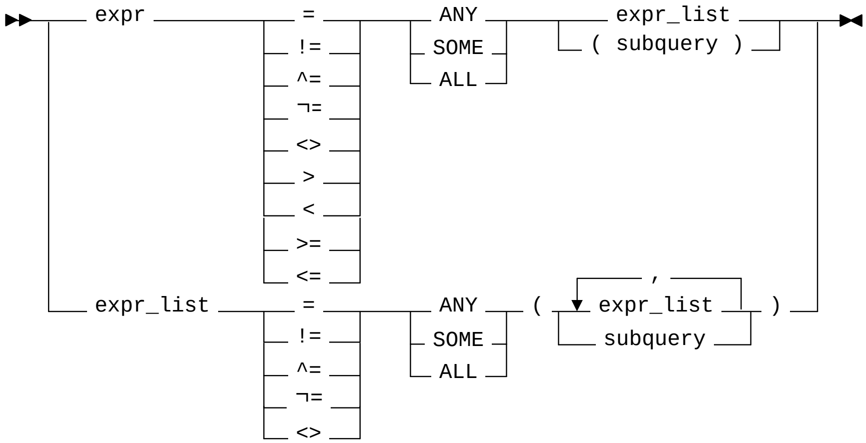


For information on comparison operators, see the section “Comparison Operators” on page 3 – 5.

Form II

A comparison with any or all members in a list or subquery.

condition (Form II) ::=

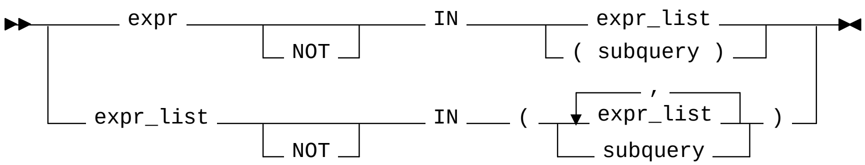


For the syntax of a subquery, see page 4 – 431.

Form III

A test for membership in a list or subquery.

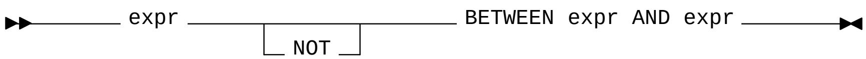
condition (Form III) ::=



Form IV

A test for inclusion in a range.

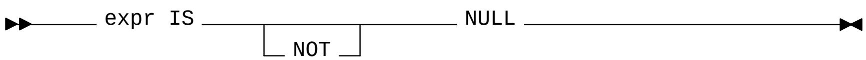
condition (Form IV) ::=



Form V

A test for nulls.

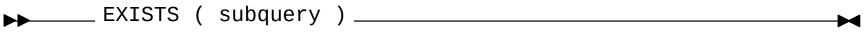
condition (Form V) ::=



Form VI

A test for existence of rows in a subquery.

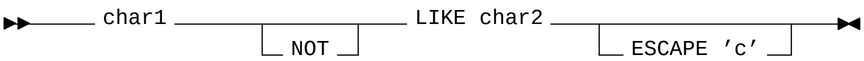
```
condition (Form VI)  ::=
```



Form VII

A test involving pattern matching.

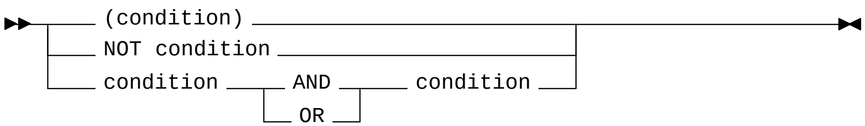
```
condition (Form VII)  ::=
```



Form VIII

A combination of other conditions.

```
condition (Form  VIII)  ::=
```



Usage Notes

You can use a *condition* in the WHERE clause of these statements:

- DELETE
- SELECT
- UPDATE

You can use a *condition* in any of these clauses of the SELECT command:

- WHERE
- START WITH
- CONNECT BY
- HAVING

A condition could be said to be of the “logical” datatype, although Oracle7 does not formally support such a datatype.

The following is a simple condition that always evaluates to TRUE:

```
1 = 1
```

The following is a more complex condition that adds the SAL value to the COMM value (substituting the value 0 for null) and determines whether the sum is greater than the number constant 2500:

```
NVL(sal, 0) + NVL(comm, 0) > 2500
```

Logical operators can combine multiple conditions into a single condition. For example, you can use the AND operator to combine two conditions:

```
(1 = 1) AND (5 < 7)
```

For more information on how to evaluate conditions with logical operators, see the section “Logical beginning” on page 3 – 11.

Examples

```
ename = 'SMITH'
emp.deptno = dept.deptno
hiredate > '01–JAN–88'
job IN ('PRESIDENT', 'CLERK', 'ANALYST')
sal BETWEEN 500 AND 1000
comm IS NULL AND sal = 2000
```

Related Topics

SELECT command on page 4 – 405
UPDATE command on page 4 – 460
DELETE command on page 4 – 286

CHAPTER

4 Commands

This chapter contains descriptions of all SQL commands and some clauses. Commands and clauses appear alphabetically. The description of each command or clause contains the following sections:

Purpose	describes the basic uses of the command.
Prerequisites	lists privileges you must have and steps that you must take before using the command. In addition to the prerequisites listed, most commands also require that the database be open by your instance, unless otherwise noted.
Syntax	shows the keywords and parameters that make up the command. The syntax diagrams used in this chapter are explained in the Preface of this manual.
Keywords and Parameters	describes the purpose of each keyword and parameter. The conventions for keywords and parameters used in this chapter are also explained in the Preface of this manual.
Usage Notes	discusses how and when to use the command.
Examples	shows example statements based on the command.
Related Topics	lists related commands, clauses, and sections of this and other manuals.

Summary of SQL Commands

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

- Data Definition Language commands
- Data Manipulation Language commands
- Transaction Control commands
- Session Control commands
- System Control commands
- Embedded SQL commands

Data Definition Language Commands

Data Definition Language (DDL) commands allow you to perform these tasks:

- create, alter, and drop objects
- grant and revoke privileges and roles
- analyze information on a table, index, or cluster
- establish auditing options
- add comments to the data dictionary

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

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

Oracle7 implicitly commits the current transaction before and after every Data Definition Language statement.

Many Data Definition Language statements may cause Oracle7 to recompile or reauthorize schema objects. For information on how Oracle7 recompiles and reauthorizes schema objects and the circumstances under which a Data Definition Language statement would cause this, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

Data Definition Language commands are not directly supported by PL/SQL, but may be available using packaged procedures supplied by Oracle corporation. For more information, see *PL/SQL User's Guide and Reference*.

Table 4 – 1 shows the Data Definition Language Commands.

Command	Purpose
ALTER CLUSTER	To change the storage characteristics of a cluster. To allocate an extent for a cluster.
ALTER DATABASE	To open / mount the database. To convert an Oracle Version 6 data dictionary when migrating to Oracle7. To prepare to downgrade to an earlier release of Oracle7. To choose archivelog / noarchivelog mode. To perform media recovery. To add / drop / clear redo log file groups members. To rename a data file / redo log file member. To backup the current control file. To backup SQL commands (that can be used to re–create the database) to the trace file. To create a new data file. To resize one or more datafiles. To create a new datafile in place of an old one for recovery purposes. To enable / disable autoextending the size of datafiles. To take a data file online / offline. To enable / disable a thread of redo log file groups. To change the database's global name. To change the MAC mode. To set the DBHIGH or DBLOW labels.
ALTER FUNCTION	To recompile a stored function.
ALTER INDEX	To redefine an index's future storage allocation.
ALTER PACKAGE	To recompile a stored package.
ALTER PROCEDURE	To recompile a stored procedure.
ALTER PROFILE	To add or remove a resource limit to or from a profile.
ALTER RESOURCE COST	To specify a formula to calculate the total cost of resources used by a session.
ALTER ROLE	To change the authorization needed to access a role.

Table 4 – 1 Data Definition Language Commands

Command	Purpose
ALTER ROLLBACK SEGMENT	To change a rollback segment's storage characteristics. To bring a rollback segment online / offline. To shrink a rollback segment to an optimal or given size.
ALTER SEQUENCE	To redefine value generation for a sequence.
ALTER SNAPSHOT	To change a snapshot's storage characteristics, automatic refresh time, or automatic refresh mode.
ALTER SHAPSHOT LOG	To change a snapshot log's storage characteristics.
ALTER TABLE	To add a column / integrity constraint to a table. To redefine a column, to change a table's storage characteristics. To enable / disable / drop an integrity constraint. To enable / disable tables locks on a table. To enable / disable all triggers on a table. To allocate an extent for the table. To allow / disallow writing to a table. To modify the degree of parallelism for a table.
ALTER TABLESPACE	To add / rename data files. To change storage characteristics. To take a tablespace online / offline. To begin / end a backup. To allow / disallow writing to a tablespace.
ALTER TRIGGER	To enable / disable a database trigger.
ALTER USER	To change a user's password, default tablespace, temporary tablespace, tablespace quotas, profile, or default roles.
ALTER VIEW	To recompile a view.
ANALYZE	To collect performance statistics, validate structure, or identify chained rows for a table, cluster, or index.
AUDIT	To choose auditing for specified SQL commands or operations on schema objects.
COMMENT	To add a comment about a table, view, shapshot, or column to the data dictionary.
CREATE CLUSTER	To create a cluster that can contain one or more tables.
CREATE CONTROLFILE	To recreate a control file.
CREATE DATABASE	To create a database.

Table 4 – 1 (continued) Data Definition Language Commands

Command	Purpose
CREATE DATABASE LINK	To create a link to a remote database.
CREATE FUNCTION	To create a stored function.
CREATE INDEX	To create an index for a table or cluster.
CREATE PACKAGE	To create the specification of a stored package.
CREATE PACKAGE BODY	To create the body of a stored package
CREATE PROCEDURE	To create a stored procedure.
CREATE PROFILE	To create a profile and specify its resource limits.
CREATE ROLE	To create a role.
CREATE ROLLBACK SEGMENT	To create a rollback segment.
CREATE SCHEMA	To issue multiple CREATE TABLE, CREATE VIEW, and GRANT statements in a single transaction.
CREATE SEQUENCE	To create a sequence for generating sequential values.
CREATE SHAPSHOT	To create a snapshot of data from one or more remote master tables.
CREATE SNAPSHOT LOG	To create a snapshot log containing changes made to the master table of a snapshot.
CREATE SYNONYM	To create a synonym for a schema object.
CREATE TABLE	To create a table, defining its columns, integrity constraints, and storage allocation.
CREATE TABLESPACE	To create a place in the database for storage of schema objects, rollback segments, and temporary segments, naming the data files to comprise the tablespace.
CREATE TRIGGER	To create a database trigger.
CREATE USER	To create a database user.
CREATE VIEW	To define a view of one or more tables or views.
DROP CLUSTER	To remove a cluster from the database.
DROP DATABASE LINK	To remove a database link.
DROP FUNCTION	To remove a stored function from the database.

Table 4 – 1 (continued) Data Definition Language Commands

Command	Purpose
DROP INDEX	To remove an index from the database.
DROP PACKAGE	To remove a stored package from the database.
DROP PROCEDURE	To remove a stored procedure from the database.
DROP PROFILE	To remove a profile from the database.
DROP ROLE	To remove a role from the database.
DROP ROLLBACK SEGMENT	To remove a rollback segment from the database.
DROP SEQUENCE	To remove a sequence from the database.
DROP SNAPSHOT	To remove a snapshot from the database.
DROP SNAPSHOT LOG	To remove a snapshot log from the database.
DROP SYNONYM	To remove a synonym from the database.
DROP TABLE	To remove a table from the database.
DROP TABLESPACE	To remove a tablespace from the database.
DROP TRIGGER	To remove a trigger from the database.
DROP USER	To remove a user and the objects in the user's schema from the database.
DROP VIEW	To remove a view from the database.
GRANT	To grant system privileges, roles and object privileges to users and roles.
NOAUDIT	To disable auditing by reversing, partially or completely, the effect of a prior AUDIT statement.
RENAME	To change the name of a schema object.
REVOKE	To revoke system privileges, roles, and object privileges from users and roles.
TRUNCATE	To remove all rows from a table or cluster and free the space that the rows used.

Table 4 – 1 (continued) Data Definition Language Commands

Data Manipulation Language Commands

Data Manipulation Language (DML) commands query and manipulate data in existing schema objects. These commands do not implicitly commit the current transaction.

Command	Purpose
DELETE	To remove rows from a table.
EXPLAIN PLAN	To return the execution plan for a SQL statement.
INSERT	To add new rows to a table.
LOCK TABLE	To lock a table or view, limiting access to it by other users.
SELECT	To select data in rows and columns from one or more tables.
UPDATE	To change data in a table.

Table 4 – 2 Data Manipulation Language Commands

All Data Manipulation Language commands except the EXPLAIN PLAN command are supported in PL/SQL.

Transaction Control Commands

Transaction Control commands manage changes made by Data Manipulation Language commands.

Command	Purpose
COMMIT	To make permanent the changes made by statements issued and the beginning of a transaction.
ROLLBACK	To undo all changes since the beginning of a transaction or since a savepoint.
SAVEPOINT	To establish a point back to which you may roll.
SET TRANSACTION	To establish properties for the current transaction.

Table 4 – 3 Transaction Control Commands

All Transaction Control commands except certain forms of the COMMIT and ROLLBACK commands are supported in PL/SQL. For information on the restrictions, see COMMIT on page 4 – 141 and ROLLBACK on page 4 – 397.

Session Control Commands

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

PL/SQL does not support session control commands.

Command	Purpose
ALTER SESSION	To enable/disable the SQL trace facility. To enable/disable global name resolution. To change the values of the session's NLS parameters. For Trusted Oracle7, to change the session label. To change the default label format. In a parallel server, to indicate that the session must access database files as if the session was connected to another instance. 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 procedures from issuing COMMIT and ROLLBACK statements. To change the goal of the cost–based optimization approach.
SET ROLE	To enable/disable roles for the current session.

Table 4 – 4 Session Control Commands

System Control Command

The single System Control command dynamically manages the properties of an Oracle7 instance. This command does not implicitly commit the current transaction.

ALTER SYSTEM is not supported in PL/SQL.

Command	Purpose
ALTER SYSTEM	To alter the Oracle7 instance by performing a specialized function.

Table 4 – 5 System Control Commands

Embedded SQL Commands

Embedded SQL commands place Data Definition Language, Data Manipulation Language, and Transaction Control statements within a procedural language program. Embedded SQL is supported by the Oracle Precompilers.

Command	Purpose
ALLOCATE	To allocate a cursor variable.
CLOSE	To disable a cursor, releasing the resources it holds.
CONNECT	To log on to an Oracle7 instance.
DECLARE CURSOR	To declare a cursor, associating it with a query.
DECLARE DATABASE	To declare the name of a remote database.
DECLARE STATEMENT	To assign a SQL variable name to a SQL statement.
DECLARE TABLE	To declare the structure of a table for semantic checking of embedded SQL statements by the Oracle Precompiler.
DESCRIBE	To initialize a descriptor, a structure holding host variable descriptions.
EXECUTE	To execute a prepared SQL statement or PL/SQL block or to execute an anonymous PL/SQL block.
EXECUTE IMMEDIATE	To prepare and execute a SQL statement containing no host variables.
FETCH	To retrieve rows selected by a query.
OPEN	To execute the query associated with a cursor.
PREPARE	To parse a SQL statement.
TYPE	To perform user–defined equivalencing.
VAR	To perform host variable equivalencing.
WHENEVER	To specify handling for error and warning conditions.

Table 4 – 6 Embedded SQL Commands

ALLOCATE (Embedded SQL)

Purpose

To allocate a cursor variable to be referenced in a PL / SQL block.

Prerequisites

You must define the cursor variable as a SQL_CURSOR pseudotype before allocating the cursor variable.

Syntax

```
▶▶── EXEC SQL ALLOCATE cursor_variable ──────────────────▶
```

Keywords and Parameters

cursor_variable is the cursor variable to be allocated.

Usage Notes

Whereas a cursor is static, a cursor variable is dynamic because it is not tied to a specific query. You can open a cursor variable for any type–compatible query.

For more information on this command, see *PL/SQL User's Guide and Reference* and *Programmer's Guide to the Oracle Precompilers.*

Example

This partial example illustrates the use of the ALLOCATE command in a Pro*C embedded SQL program:

```
EXEC SQL BEGIN DECLARE SECTION;
    SQL_CURSOR emp_cv;
    struct{ ... } emp_rec;
EXEC SQL END DECLARE SECTION;
EXEC SQL ALLOCATE emp_cursor;
EXEC SQL EXECUTE
    BEGIN
        OPEN :emp_cv FOR SELECT * FROM emp;
    END;
END-EXEC;
for (;;)
{EXEC SQL FETCH :emp_cv INTO emp_rec; }
```

Related Topics

CLOSE command on 4 – 139 EXECUTE command on 4 – 332 FETCH command on 4 – 341

ALTER CLUSTER

Purpose

To redefine storage and parallelism characteristics for a cluster.

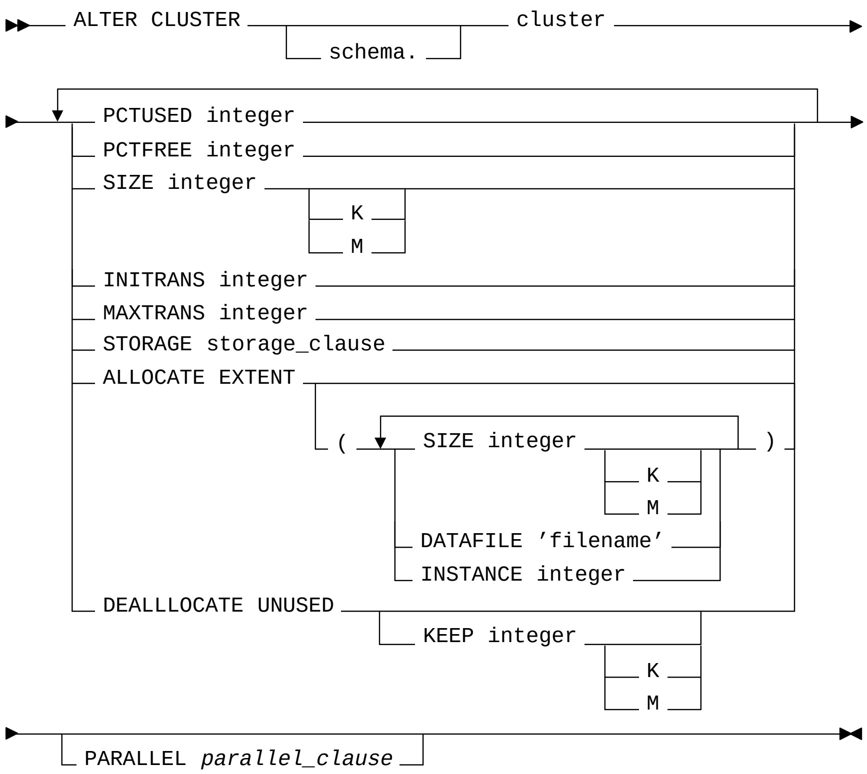
Prerequisites

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

If you are using Trusted Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the cluster. If you omit *schema,* Oracle7 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 on page 4 – 164.

PCTUSED
PCTFREE
INITRANS
MAXTRANS changes the values of these parameters for the cluster. See the PCTUSED, PCTFREE, INITRANS, and MAXTRANS parameters of the CREATE CLUSTER command on page 4 – 164.

STORAGE changes the storage characteristics for the cluster. See the STORAGE clause on page 4 – 449.

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, Oracle7 determines the size based on the values of the cluster's STORAGE parameters.

DATAFILE specifies one of the datafiles in the cluster's tablespace to contain the new extent. If you omit this parameter, Oracle7 chooses the datafile.

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 Oracle7 with the Parallel Server option in parallel mode.

Explicitly allocating an extent with this clause does not cause Oracle7 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.

DEALLOCATE UNUSED

explicitly deallocates unused space at the end of the cluster and make the freed space available for other segments. Only unused space above the high–water mark can be freed. If KEEP is omitted, all unused space is freed. For more information, see the *deallocate_clause* on page 4 – 278.

KEEP specifies the number of bytes above the high–water mark that the cluster will have after deallocation. If the number of remaining extents are less than MINEXTENTS, then MINEXTENTS is set to the current number of extents. If the initial extent becomes smaller than INITIAL, then INITIAL is set to the value of the current initial extent.

PARALLEL specifies the degree of parallelism for creating the cluster and the default degree of parallelism for queries on the cluster once created. For more information, see the *parallel_clause* on page 4 – 378.

Usage Notes

You can perform these tasks with the ALTER CLUSTER command:

- change the MAXTRANS parameter value for data blocks in the cluster
- change the SIZE, PCTUSED, PCTFREE, and INITRANS parameter values for future data blocks in the cluster
- change future storage characteristics with the STORAGE characteristics NEXT, PCTINCREASE, and MAXEXTENTS
- explicitly allocate an extent
- explicitly deallocate space from unused extents

You cannot perform these tasks with the ALTER CLUSTER command:

- change the number or the name of columns in the cluster key
- change the values of the STORAGE parameters INITIAL and MINEXTENTS
- change the tablespace in which the cluster is stored
- remove tables from a cluster (see the DROP CLUSTER command on 4 – 301 and DROP TABLE command on 4 – 318)

Example I The following statement alters the CUSTOMER cluster in the schema SCOTT:

```
ALTER CLUSTER scott.customer
    SIZE 512
    STORAGE (MAXEXTENTS 25)
```

Oracle7 now allocates 512 bytes for each cluster key value. Assuming a data block size of 2 kilobytes, future data blocks within this cluster contain 4 cluster keys per data block, or 2 kilobytes divided by 512 bytes.

The cluster can have a maximum of 25 extents.

Example II The following statement deallocates unused space from CUSTOMER cluster, keeping 30 Kilobytes of unused space for future use:

```
ALTER CLUSTER scott.customer  DEALLOCATE UNUSED KEEP 30 K
```

Related Topics

CREATE CLUSTER command on 4 – 164
CREATE TABLE command on 4 – 245
DROP CLUSTER command on 4 – 301
DROP TABLE command on 4 – 318
STORAGE clause on 4 – 449

ALTER DATABASE

Purpose

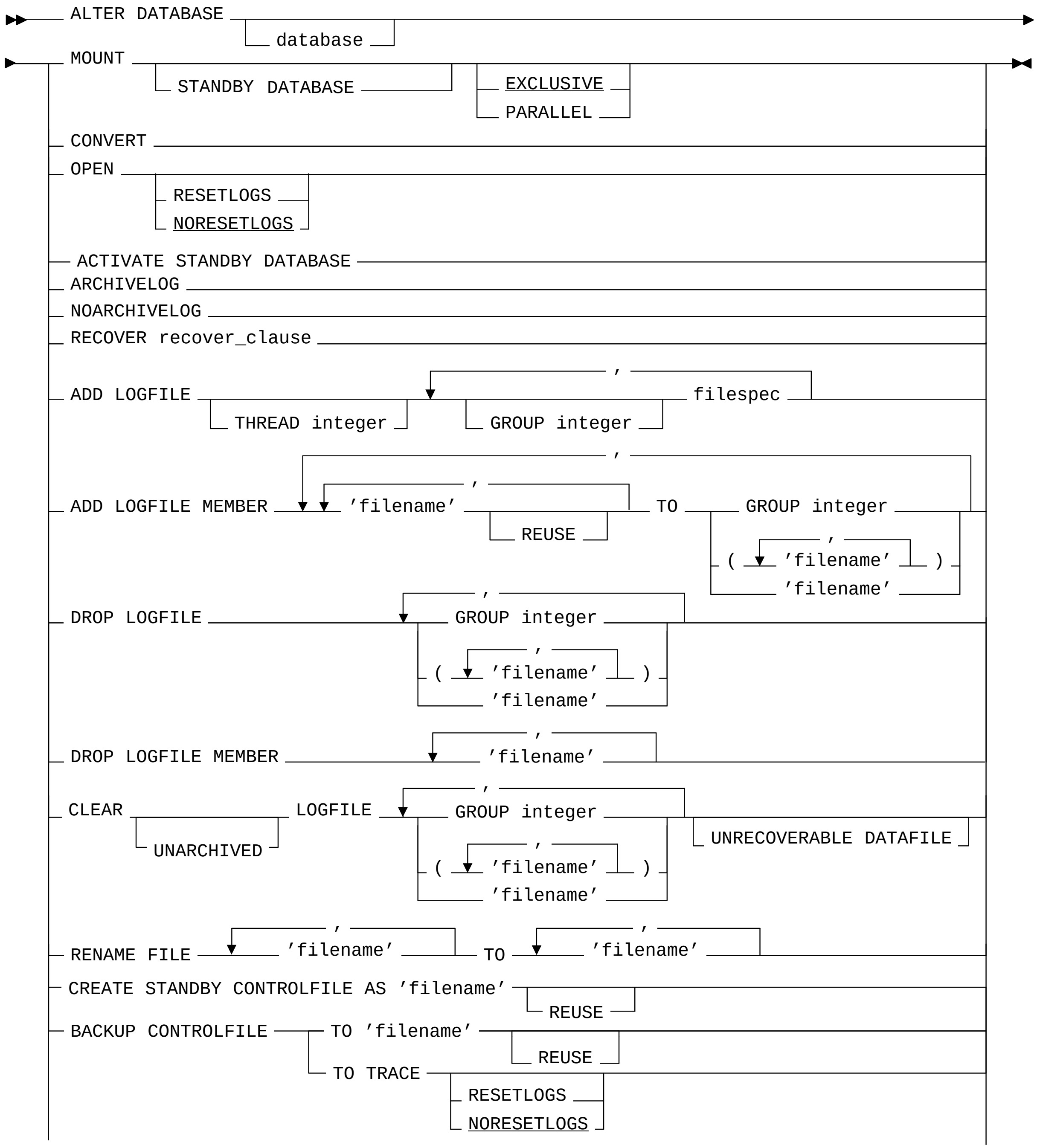
To alter an existing database in one of these ways:

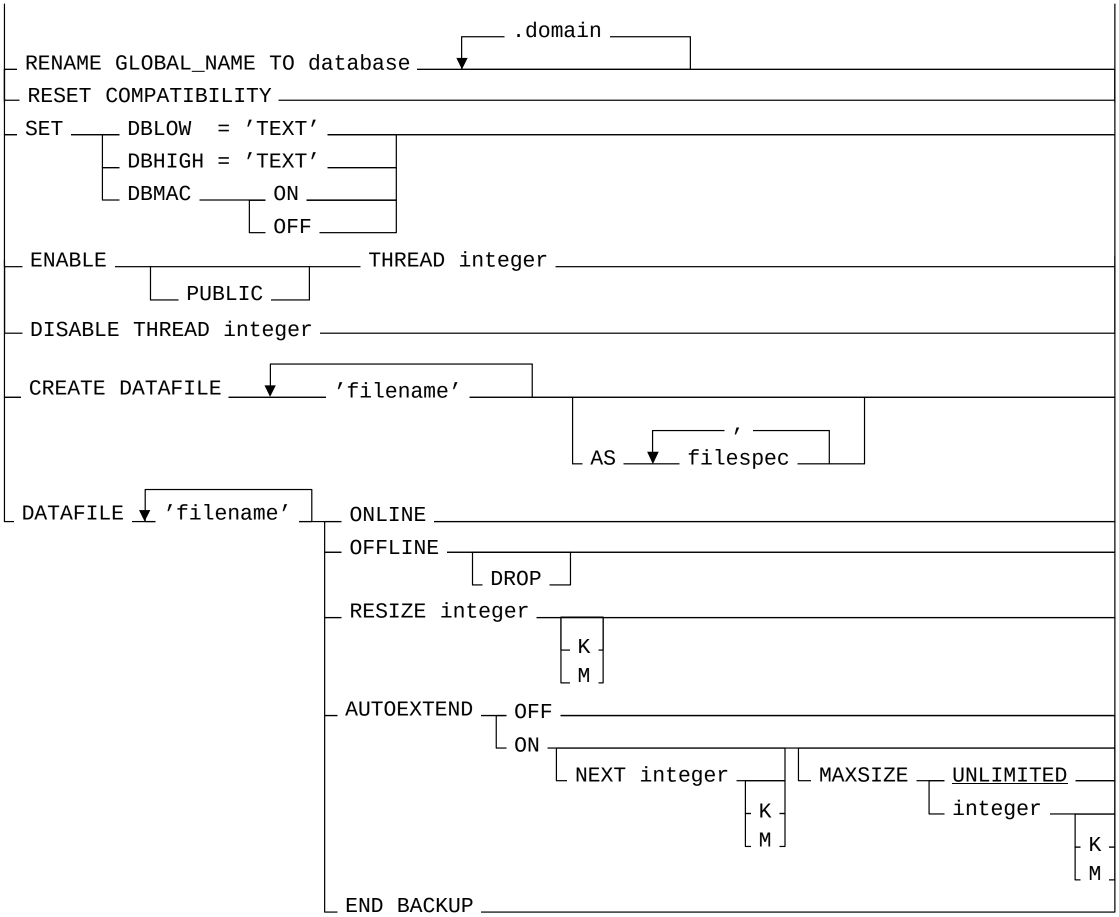
- mount the database or standby database
- convert an Oracle Version 6 data dictionary when migrating to Oracle7
- open the database
- choose archivelog or noarchivelog mode for redo log file groups
- perform media recovery
- add or drop a redo log file group or a member of a redo log file group
- clear and initialize an online redo log file
- rename a redo log file member or a datafile
- backup the current control file
- backup SQL commands (that can be used to re–create the database) to the database's trace file
- take a datafile online or offline
- enable or disable a thread of redo log file groups
- change the database's global name
- prepare to downgrade to an earlier release of Oracle7
- change the MAC mode
- equate the predefined label DBHIGH or DBLOW with an operating system label
- resize one or more datafiles
- create a new datafile in place of an old one for recovery purposes
- enable or disable the autoextending of the size of datafiles

Prerequisites

You must have ALTER DATABASE system privilege.

Syntax


ALTER DATABASE
database
MOUNT
STANDBY DATABASE
EXCLUSIVE
PARALLEL
CONVERT
OPEN
RESETLOGS
NORESETLOGS
ACTIVATE STANDBY DATABASE
ARCHIVELOG
NOARCHIVELOG
RECOVER recover_clause
ADD LOGFILE
THREAD integer
GROUP integer
filespec
ADD LOGFILE MEMBER
'filename'
REUSE
TO
GROUP integer
('filename')
'filename'
DROP LOGFILE
GROUP integer
('filename')
'filename'
DROP LOGFILE MEMBER
'filename'
CLEAR
UNARCHIVED
LOGFILE
GROUP integer
('filename')
'filename'
UNRECOVERABLE DATAFILE
RENAME FILE
'filename'
TO
'filename'
CREATE STANDBY CONTROLFILE AS 'filename'
REUSE
BACKUP CONTROLFILE
TO 'filename'
REUSE
TO TRACE
RESETLOGS
NORESETLOGS


RENAME GLOBAL_NAME TO database
.domain
RESET COMPATIBILITY
SET
DBLOW = 'TEXT'
DBHIGH = 'TEXT'
DBMAC
ON
OFF
ENABLE
PUBLIC
THREAD integer
DISABLE THREAD integer
CREATE DATAFILE
'filename'
AS
,
filespec
DATAFILE
'filename'
ONLINE
OFFLINE
DROP
RESIZE integer
K
M
AUTOEXTEND
OFF
ON
NEXT integer
K
M
MAXSIZE
UNLIMITED
integer
K
M
END BACKUP

Keywords and Parameters

database identifies the database to be altered. If you omit *database*, Oracle7 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.

STANDBY DATABASE mounts the standby database. For more information, see the *Oracle7 Server Administrator's Guide*.

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 Oracle7 with the Parallel Server option. You cannot use this option with the STANDBY option or 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 *Oracle7 Server Migration*.

OPEN

opens the database, making it available for normal use. You must mount the database before you can open it. You cannot open a standby database that has not been activated.

RESETLOGS

resets the current log sequence number to 1 and discards any redo information that was not applied during recovery; ensuring that it will never be applied. This effectively discards all changes to the database. You must use this option to open the database after performing media recovery with an incomplete recovery using the RECOVER UNTIL clause (see page 4 – 382) or 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 the above options after performing incomplete media recovery or complete media recovery with a backup controlfile. In any other case, Oracle7 uses the NORESETLOGS automatically.

ACTIVATE STANDBY DATABASE

changes the state of a standby database to an active database. For more information, see *Oracle7 Server Administrator's Guide.*

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.

RECOVER performs media recovery. See the RECOVER clause on page 4 – 382. You only recover the entire database when the database is closed. You can recover tablespaces or datafiles when the database is open or closed, provided the tablespaces or datafiles to be recovered are offline. You cannot perform media recovery if you are connected to Oracle7 through the multi–threaded server architecture. You can also perform media recovery with the Server Manager recovery dialog box.

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 Oracle7 with the Parallel Server option in parallel mode.

Each *filespec* specifies a redo log file group containing one or more members, or copies. See the syntax description of *filespec* on page 4 – 343.

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, Oracle7 generates its value automatically. You can examine the GROUP value for a redo log file group through the dynamic performance table V$LOG.

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, Oracle7 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 manner as the ADD LOGFILE MEMBER clause. You cannot drop a redo log file group if it needs archiving or is the currently active group. Nor can you drop a redo log file group if doing so would cause the redo thread to contain less than two redo log file groups.

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.

CLEAR LOGFILE — reinitialize an online redo log and optionally not archive the redo log. CLEAR LOGFILE is similar to adding and dropping a redo log except that the command may be issued even if there are only two logs for the thread and also may be issued for the current redo log of a closed thread.

CLEAR LOGFILE cannot be used to clear a log needed for media recovery. If it is necessary to clear a log containing redo after the database checkpoint, then incomplete media recovery will be necessary. The current redo log of an open thread can never be cleared. The current log of a closed thread can be cleared by switching logs in the closed thread.

If the CLEAR LOGFILE command is interrupted by a system or instance failure, then the database may hang. If so, the command must be reissued once the database is restarted. If the failure occurred because of I/O errors accessing one member of a log group, then that member can be dropped and other members added.

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

Warning: Specifying UNARCHIVED will make backups unusable if the redo log is needed for recovery.

UNRECOVERABLE DATAFILE

you must specify UNRECOVERABLE DATAFILE if the tablespace has a datafile offline and the unarchived log must be cleared to bring the tablespace online. If so, then the datafile and entire tablespace must be dropped once the CLEAR LOGFILE command completes.

RENAME FILE
: renames datafiles 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.

CREATE STANDBY CONTROLFILE

create a controlfile to be used to maintain a standby database. For more information, see *Oracle7 Server Administrator's Guide*.

BACKUP CONTROLFILE
: backs up the current control file.

TO '*filename*'
: specifies the file to which the control file is backed up. You must fully specify the '*filename*' using the conventions for your operating system. If the specified file already exists, you must specify the REUSE option.

TO TRACE
: writes SQL statements to the database's trace file, rather than making a physical backup of the control file.

The SQL commands can be used to start up the database, re–create the control file, and recover and open the database appropriately, based on the created control file.

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

RESETLOGS — the SQL statement written to the trace file for starting the database is ALTER DATABASE OPEN RESETLOGS.

NORESETLOGS — the SQL statement written to the trace file for starting the database is ALTER DATABASE OPEN NORESETLOGS.

You can only use the following options when your instance has the database open:

ENABLE — in a parallel server, 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 *domain*s 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.

For more information on global names, see the "Network Administration" chapter of *Oracle7 Server Distributed Systems, Volume I*.

RESET COMPATIBILITY

mark the database to be reset to an earlier version of Oracle7 when the database is next restarted.

Note: RESET COMPATIBILITY will not work unless you have successfully disabled Oracle7 features that affect backward compatibility.

For more information on downgrading to an earlier version of Oracle7, see the "Upgrading and Downgrading" chapter of *Oracle7 Server Migration*.

SET

for Trusted Oracle7, changes one of the following:

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*'.

DBMAC ON — configures Trusted Oracle7 in DBMS MAC mode.

DBMAC OFF — configures Trusted Oracle7 in OS MAC mode.

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 Oracle7. For more information on this clause, see the *Trusted Oracle7 Server Administrator's Guide.*

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:

CREATE DATAFILE

creates a new empty datafile in place of an old one. You can use this option to re–create a datafile 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 datafile. If you omit the AS clause, Oracle7 creates the new file with the same name and size as the file specified by '*filename*'.

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

Oracle7 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 datafile of the SYSTEM tablespace.

DATAFILE

changes one of the following for your database:

ONLINE

brings the datafile online.

OFFLINE

takes the datafile offline.

If the database is open, then you must perform media recovery on the datafile before bringing it back online. This is because a checkpoint is not performed on the datafile before it is taken offline.

DROP	takes a datafile offline when the database is in NOARCHIVELOG mode.
RESIZE	attempts to change the size of the datafile to the specified absolute size in bytes. You can also use K or M to specify this size in kilobytes or megabytes. There is no default, so you must specify a size.
AUTOEXTEND	enables or disables the automatic extension of a datafile.
OFF	disable autoextend if it is turned on. NEXT and MAXSIZE are set to zero. Values for NEXT and MAXSIZE must be respecified in further ALTER DATABASE AUTOEXTEND commands.
ON	enable autoextend.
NEXT	the size in bytes of the next increment of disk space to be automatically allocated to the datafile when more extents are required. You can also use K or M to specify this size in kilobytes or megabytes. The default is one data block.
MAXSIZE	maximum disk space allowed for automatic extension of the datafile.
UNLIMITED	set no limit on allocating disk space to the datafile.
END BACKUP	avoid media recovery on database startup after an online tablespace backup was interrupted by a system failure or instance failure or SHUTDOWN ABORT.

Warning: Do not use ALTER TABLESPACE ... END BACKUP if you have restored any of the files affected from a backup. Media recovery is fully described in the *Oracle7 Server Administrator's Guide.*

Usage Notes

For more information on using the ALTER DATABASE command for database maintenance, see the *Oracle7 Server Administrator's Guide.*

Example I The following statement mounts the database named STOCKS exclusively:

```
ALTER DATABASE stocks     MOUNT EXCLUSIVE
```

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

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

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

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

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

```
ALTER DATABASE stocks
    DROP LOGFILE MEMBER 'diskc:log3.log'
```

Example V The following statement renames a redo log file member:

```
ALTER DATABASE stocks
    RENAME FILE 'diskb:log3.log' TO 'diskd:log3.log'
```

The above statement only changes the member of the redo log group from one file to another. The statement does not actually change the name of the file 'DISKB:LOG3.LOG' to 'DISKD:LOG3.LOG'. You must perform this operation through your operating system.

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

```
ALTER DATABASE stocks DROP LOGFILE GROUP 3
```

Example VII The following statement adds a redo log file group containing three members to thread 5 and assigns it a GROUP parameter value of 4:

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

Example VIII The following statement disables thread 5 in a parallel server:

```
ALTER DATABASE stocks
    DISABLE THREAD 5
```

Example IX The following statement enables thread 5 in a parallel server, making it available to any Oracle7 instance that does not explicitly request a specific thread:

```
ALTER DATABASE stocks
    ENABLE PUBLIC THREAD 5
```

Example X The following statement creates the datafile 'DISK1:DB1.DAT' based on the file 'DISK2:DB1.DAT':

```
ALTER DATABASE
    CREATE DATAFILE 'disk1:db1.dat' AS 'disk2:db1.dat'
```

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

```
ALTER DATABASE
    RENAME GLOBAL_NAME TO sales.australia.acme.com
```

Example XII The following statement attempts to change the size of datafile 'DISK1:DB1.DAT':

```
ALTER DATABASE
    DATAFILE 'disk1:db1.dat' RESIZE 10 M
```

For examples of performing media recovery, see the *Oracle7 Server Administrator's Guide*.

Example XIII The following statement clears a log file:

```
ALTER DATABASE
    CLEAR LOGFILE 'disk3:log.dbf'
```

Related Topics

CREATE DATABASE command 4 – 178
RECOVER, STARTUP, and SHUTDOWN Server Manager commands in the *Oracle Server Manager User's Guide.*

ALTER FUNCTION

Purpose

To recompile a stand–alone stored function.

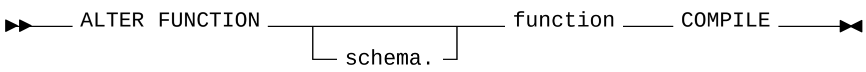
Prerequisites

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

If you are using Trusted Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the function. If you omit *schema*, Oracle7 assumes the function is in your own schema.

function is the name of the function to be recompiled.

COMPILE causes Oracle7 to recompile the function. The COMPILE keyword is required.

Usage Notes

You can use the ALTER FUNCTION command to explicitly recompile a function that is invalid. Explicit recompilation eliminates the need for implicit runtime recompilation and prevents associated runtime compilation errors and performance overhead.

The ALTER FUNCTION command is similar to the ALTER PROCEDURE command on 4 – 42. For information on how Oracle7 recompiles functions and procedures, see the “Dependencies Among Schema Objects” chapter of *Oracle7 Server Concepts*.

> **Note:** This command does not change the declaration or definition of an existing function. To re–declare or redefine a function, you must use the CREATE FUNCTION command (on page 4 – 188) with the OR REPLACE option.

Example

To explicitly recompile the function GET_BAL owned by the user MERRIWEATHER, issue the following statement:

```
ALTER FUNCTION merriweather.get_bal
    COMPILE
```

If Oracle7 encounters no compilation errors while recompiling GET_BAL, GET_BAL becomes valid. Oracle7 can subsequently execute it without recompiling it at runtime. If recompiling GET_BAL results in compilation errors, Oracle7 returns an error message and GET_BAL remains invalid.

Oracle7 also invalidates all objects that depend upon GET_BAL. If you subsequently reference one of these objects without explicitly recompiling it first, Oracle7 recompiles it implicitly at runtime.

Related Topics

ALTER PROCEDURE command on 4 – 42
CREATE FUNCTION command on 4 – 188

ALTER INDEX

Purpose

To change storage allocation for an index or rebuild an index.

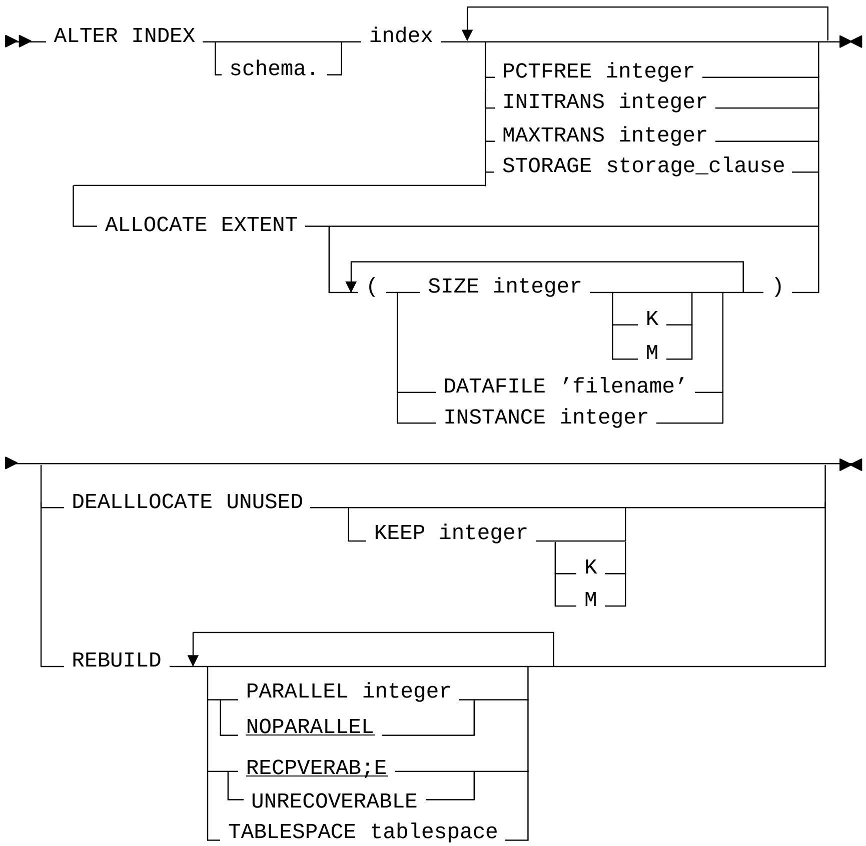
Prerequisites

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

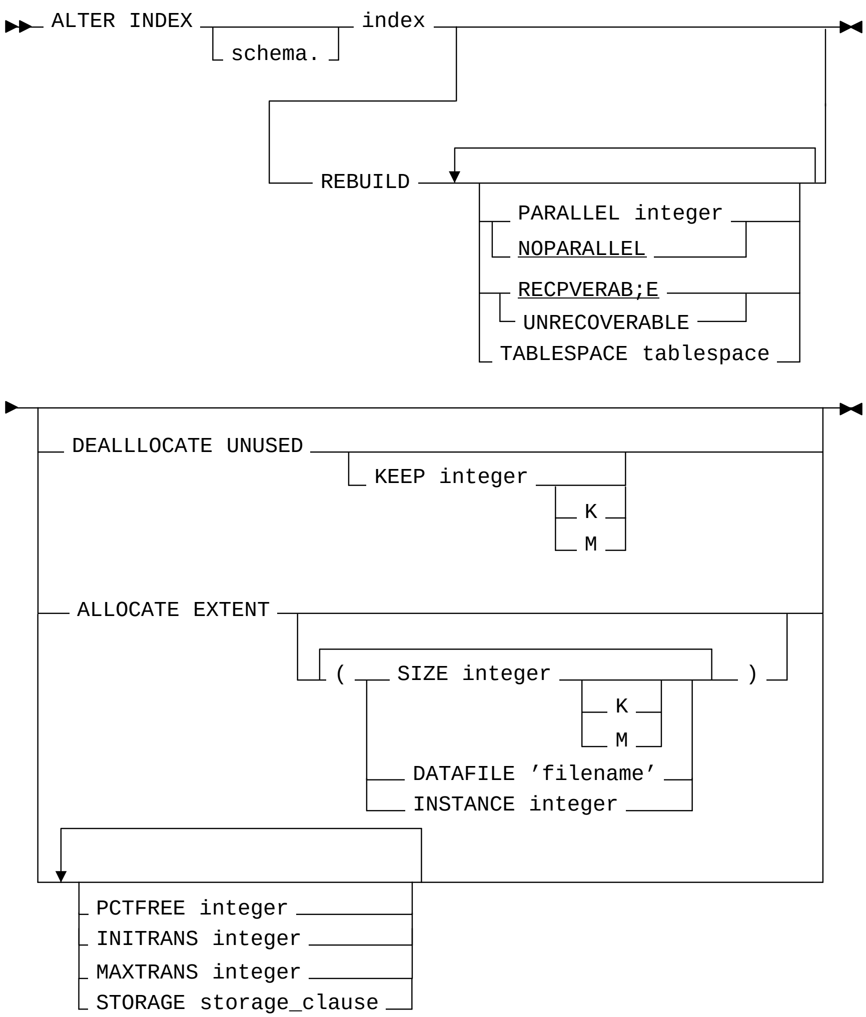
If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the index's creation label or you must satisfy one of these criteria:

- 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax


ALTER INDEX
schema.
index
PCTFREE integer
INITRANS integer
MAXTRANS integer
STORAGE storage_clause
ALLOCATE EXTENT
(
SIZE integer
K
M
)
DATAFILE ’filename’
INSTANCE integer
DEALLLOCATE UNUSED
KEEP integer
K
M
REBUILD
PARALLEL integer
NOPARALLEL
RECPVERAB;E
UNRECOVERABLE
TABLESPACE tablespace

Syntax



Keywords and Parameters

schema is the schema containing the index. If you omit *schema*, Oracle7 assumes the index is in your own schema.

index is the name of the index to be altered.

PCTFREE
INITRANS
MAXTRANS changes the values of these parameters for the index. See the PCTFREE, INITRANS and MAXTRANS parameters of the CREATE TABLE command on page 4 – 245.

STORAGE — changes the storage parameters for the index. See the STORAGE clause on page 4 – 449.

ALLOCATE EXTENT — explicitly allocates a new extent for the index.

- 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, Oracle7 determines the size based on the values of the index's STORAGE parameters.
- DATAFILE — specifies one of the data files in the index's tablespace to contain the new extent. If you omit this parameter, Oracle7 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 Oracle7 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.

DEALLOCATE UNUSED

explicitly deallocates unused space at the end of the index and make the freed space available for other segments. Only unused space above the high–water mark can be freed. If KEEP is omitted, all unused space is freed. For more information, see the *deallocate_clause* on page 4 – 278.

KEEP

specifies the number of bytes above the high–water mark that the index will have after deallocation. If the number of remaining extents are less than MINEXTENTS, then MINEXTENTS is set to the current number of extents. If the initial extent becomes smaller than INITIAL, then INITIAL is set to the value of the current initial extent.

REBUILD

create the index anew using the existing index.

PARALLEL

use *integer* parallel processes to build the new index.

NOPARALLEL

do not use parallel processes to build the new index. This is the default.

RECOVERABLE

specifies that the creation of the index will be logged in the redo log file. This is the default.

If the database is run in ARCHIVELOG mode, media recovery from a backup will recreate the index. You cannot specify RECOVERABLE when using NOARCHIVELOG mode.

UNRECOVERABLE

specifies that the creation of the index will not be logged in the redo log file. As a result, media recovery will not recreate the index.

When this option is used, index creation is faster than the RECOVERABLE option because no redo log entries are written.

TABLESPACE

specifies the tablespace where the rebuilt index will be stored. The default is the default tablespace of the user issuing the command.

Usage Notes

The INITRANS and MAXTRANS parameters as well as the STORAGE and ALLOCATE EXTENT clauses, all have the same function as in the CREATE TABLE command, which is described on page 4 – 245.

Example

This statement alters SCOTT'S CUSTOMER index so that future data blocks within this index use 5 initial transaction entries and an incremental extent of 100 kilobytes:

```
ALTER INDEX scott.customer
    INITRANS 5
    STORAGE (NEXT 100K)
```

Related Topics

CREATE INDEX command on 4 – 192
CREATE TABLE command on 4 – 245
STORAGE clause on 4 – 449

ALTER PACKAGE

Purpose

To recompile a stored package.

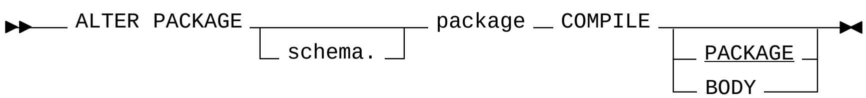
Prerequisites

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

If you are using Trusted Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the package. If you omit *schema*, Oracle7 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.

Usage Notes

You can use the ALTER PACKAGE command to explicitly recompile either a package specification and body or only a package body. Explicit recompilation eliminates the need for implicit runtime recompilation and prevents associated runtime compilation errors and performance overhead.

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

> **Note:** This command does not change the declaration or definition of an existing package. To re–declare or redefine a package, you must use the CREATE PACKAGE or the CREATE PACKAGE BODY command with the OR REPLACE option.

Recompiling Package Specifications

You might want to recompile a package specification to check for compilation errors after modifying the specification. When you issue an ALTER PACKAGE statement with the COMPILE PACKAGE option, Oracle7 recompiles the package specification and body regardless of whether it is invalid. When you recompile a package specification, Oracle7 invalidates any local objects that depend on the specification, such as procedures that call procedures or functions in the package. Note that the body of a package also depends on its specification. If you subsequently reference one of these dependent objects without first explicitly recompiling it, Oracle7 recompiles it implicitly at runtime.

Recompiling Package Bodies

You might want to recompile a package body after modifying it. When you issue an ALTER PACKAGE statement with the COMPILE BODY option, Oracle7 recompiles the package body regardless of whether it is invalid. When you recompile a package body, Oracle7 first recompiles the objects on which the body depends, if any of these objects are invalid. If Oracle7 recompiles the body successfully, the body becomes valid. If recompiling the body results in compilation errors, Oracle7 returns an error and the body remains invalid. You can then debug the body using the predefined package DBMS_OUTPUT. Note that recompiling a package body does not invalidate objects that depend upon the package specification.

For more information on debugging packages, see the “Using Procedures and Packages” chapter of *Oracle7 Server Application Developer’s Guide*. For information on how Oracle7 maintains dependencies among schema objects, including remote objects, see the “Dependencies Among Schema Objects” chapter of *Oracle7 Server Concepts*.

Example I This statement explicitly recompiles the specification and body of the ACCOUNTING package in the schema BLAIR:

```
ALTER PACKAGE blair.accounting
    COMPILE PACKAGE
```

If Oracle7 encounters no compilation errors while recompiling the ACCOUNTING specification and body, ACCOUNTING becomes valid. BLAIR can subsequently call or reference all package objects declared in the specification of ACCOUNTING without runtime recompilation. If recompiling ACCOUNTING results in compilation errors, Oracle7 returns an error message and ACCOUNTING remains invalid.

Oracle7 also invalidates all objects that depend upon ACCOUNTING. If you subsequently reference one of these objects without explicitly recompiling it first, Oracle7 recompiles it implicitly at runtime.

Example II To recompile the body of the ACCOUNTING package in the schema BLAIR, issue the following statement:

```
ALTER PACKAGE blair.accounting
    COMPILE BODY
```

If Oracle7 encounters no compilation errors while recompiling the package body, the body becomes valid. BLAIR can subsequently call or reference all package objects declared in the specification of ACCOUNTING without runtime recompilation. If recompiling the body results in compilation errors, Oracle7 returns an error message and the body remains invalid.

Because the following statement recompiles the body and not the specification of ACCOUNTING, Oracle7 does not invalidate dependent objects.

Related Topics

CREATE PACKAGE command on 4 – 198
CREATE PACKAGE BODY command on 4 – 202

ALTER PROCEDURE

Purpose

To recompile a stand–alone stored procedure.

Prerequisites

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

If you are using Trusted Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>── ALTER PROCEDURE ──┬───────────┬── package ── COMPILE ──────><
                       └─ schema. ─┘
```

Keywords and Parameters

schema — is the schema containing the procedure. If you omit *schema*, Oracle7 assumes the procedure is in your own schema.

procedure — is the name of the procedure to be recompiled.

COMPILE — causes Oracle7 to recompile the procedure. The COMPILE keyword is required.

Usage Notes

The ALTER PROCEDURE command and the ALTER FUNCTION command are quite similar. The following discussion of explicitly recompiling procedures also applies to functions.

You can use the ALTER PROCEDURE command to explicitly recompile a procedure that is invalid. Explicit recompilation eliminates the need for implicit runtime recompilation and prevents associated runtime compilation errors and performance overhead.

When you issue an ALTER PROCEDURE statement, Oracle7 recompiles the procedure regardless of whether it is valid or invalid.

You can only use the ALTER PROCEDURE command to recompile a stand–alone procedure. To recompile a procedure that is part of a package, you must recompile the entire package using the ALTER PACKAGE command.

When you recompile a procedure, Oracle7 first recompiles objects upon which the procedure depends, if any of these objects are invalid. Oracle7 also invalidates any local objects that depend upon the procedure, such as procedures that call the recompiled procedure or package bodies that define procedures that call the recompiled procedure. If Oracle7 recompiles the procedure successfully, the procedure becomes valid. If recompiling the procedure results in compilation errors, then Oracle7 returns an error and the procedure remains invalid. You can then debug procedures using the predefined package DBMS_OUTPUT. For information on debugging procedures, see the "Using Procedures and Packages" chapter of the *Oracle7 Server Application Developer's Guide*. For information on how Oracle7 maintains dependencies among schema objects, including remote objects, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

> **Note:** This command does not change the declaration or definition of an existing procedure. To re–declare or redefine a procedure, you must use the CREATE PROCEDURE command with the OR REPLACE option.

Example

To explicitly recompile the procedure CLOSE_ACCT owned by the user HENRY, issue the following statement:

```
ALTER PROCEDURE henry.close_acct
    COMPILE
```

If Oracle7 encounters no compilation errors while recompiling CLOSE_ACCT, CLOSE_ACCT becomes valid. Oracle7 can subsequently execute it without recompiling it at runtime. If recompiling CLOSE_ACCT results in compilation errors, Oracle7 returns an error and CLOSE_ACCT remains invalid.

Oracle7 also invalidates all dependent objects. These objects include any procedures, functions, and package bodies that call CLOSE_ACCT. If you subsequently reference one of these objects without first explicitly recompiling it, Oracle7 recompiles it implicitly at runtime.

Related Topics

ALTER FUNCTION command on 4 – 31
ALTER PACKAGE command on 4 – 39
CREATE PROCEDURE command on 4 – 206

ALTER PROFILE

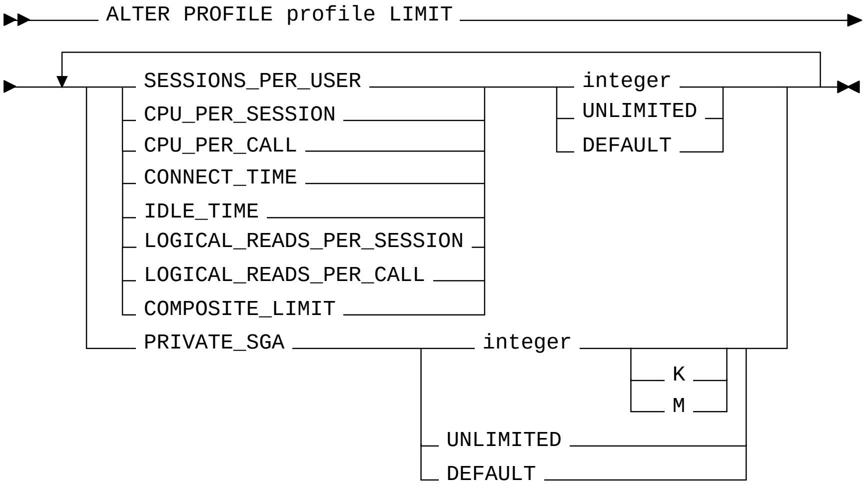
Purpose To add, modify, or remove a resource limit in a profile.

Prerequisites You must have ALTER PROFILE system privilege.

If you are using Trusted Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

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 on page 4 – 210.

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.

Usage Notes

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

You cannot remove a limit from the DEFAULT profile.

Example I

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

```
ALTER PROFILE engineer LIMIT SESSIONS_PER_USER  5
```

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

Example II

This statement defines unlimited idle time for the ENGINEER profile:

```
ALTER PROFILE engineer LIMIT IDLE_TIME UNLIMITED
```

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

Example III

This statement removes the IDLE_TIME limit from the ENGINEER profile:

```
ALTER PROFILE engineer LIMIT IDLE_TIME DEFAULT
```

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

Example IV

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

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

This IDLE_TIME limit applies to these users:

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

Related Topics

CREATE PROFILE command on 4 – 210

ALTER RESOURCE COST

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.

Prerequisites

You must have ALTER RESOURCE COST system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match DBLOW or you must have WRITEDOWN system privileges.

Syntax

```
ALTER RESOURCE COST  ┬ CPU_PER_SESSION integer ────────────┬
                     ├ CONNECT_TIME integer ───────────────┤
                     ├ LOGICAL_READS_PER_SESSION integer ──┤
                     └ PRIVATE_SGA integer ────────────────┘
```

Keywords and Parameters

integer is the weight of each resource.

Usage Notes

The ALTER RESOURCE COST command specifies the formula by which Oracle7 calculates the total resource cost used in a session. With this command, you can assign a weight to each of these resources:

CPU_PER_SESSION
: The amount of CPU time used by a session measured in hundredths of seconds.

CONNECT_TIME
: The amount of CPU time used by a session measured in hundredths of seconds.

CPU_PER_SESSION
: The elapsed time of a session measured in minutes.

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

PRIVATE_SGA
: 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.

Oracle7 calculates the total resource cost by multiplying the amount of each resource used in the session by the resource's weight and summing the products for all four resources. Both the products and the total cost are expressed in units called *service units*.

Although Oracle7 monitors the use of other resources, only these four can contribute to the total resource cost for a session. For information on all resources, see the CREATE PROFILE command on page 4 – 210.

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

Once you have specified a formula for the total resource cost, you can limit this cost for a session with the COMPOSITE_LIMIT parameter of the CREATE PROFILE command. If a session's cost exceeds the limit, Oracle7 aborts the session and returns an error. For information on establishing resource limits, see the CREATE PROFILE command on page 4 – 210. If you use the ALTER RESOURCE COST command to change the weight assigned to each resource, Oracle7 uses these new weights to calculate the total resource cost for all current and subsequent sessions.

Example

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

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

The weights establish this cost formula for a session:

$T = (100 * CPU) + CON$

where:

T	is the total resource cost for the session expressed in service units.
CPU	is the CPU time used by the session measured in hundredths of seconds.
CON	is the elapsed time of a session measured in minutes.

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

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

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

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

These new weights establish a new cost formula:

$T = (100 * CPU) + (2 * LOG)$

where:

T CPU	are the same as in the previous formula.
LOG	is the number of data blocks read during the session.

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

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

Related Topics

CREATE PROFILE command on 4 – 210

ALTER ROLE

Purpose

To change the authorization needed to enable a role.

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 Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>── ALTER ROLE role ──┬─ NOT IDENTIFIED ───────────────────┬──><
                       └─ IDENTIFIED ────┬─ BY password ──┤
                                         └─ EXTERNALLY ───┘
```

Keywords and Parameters

The keywords and parameters in the ALTER ROLE command all have the same meaning as in the CREATE ROLE command. For information on these keywords and parameters, see the CREATE ROLE command on page 4 – 215.

Example

This statement changes the password on the TELLER role to LETTER:

```
ALTER ROLE teller
   IDENTIFIED BY letter
```

Users granted the TELLER role must subsequently specify the new password to enable the role.

Related Topics

CREATE ROLE command on 4 – 215
SET ROLE command on 4 – 442

ALTER ROLLBACK SEGMENT

Purpose

To alter a rollback segment in one of these ways:

- by bringing it online
- by taking it offline
- by changing its storage characteristics
- by shrinking it to an optimal or given size

Prerequisites

You must have ALTER ROLLBACK SEGMENT system privilege.

If you are using Trusted Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>-- ALTER ROLLBACK SEGMENT rollback_segment --+- ONLINE --------------------+-><
                                               +- OFFLINE -------------------+
                                               +- STORAGE storage_clause ----+
                                               +- SHRINK --------------------+
                                                          +- TO integer -----+
                                                                       +- K -+
                                                                       +- M -+
```

Keywords and Parameters

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. See the STORAGE clause on page 4 – 449.
SHRINK	attempts to shrink the rollback segment to an optimal or given size.

Usage Notes

When you create a rollback segment, it is initially offline. An offline rollback segment is not available for transactions.

The ONLINE option brings the rollback segment online making it available for transactions by your instance. You can also bring a rollback segment online when you start your instance with the initialization parameter ROLLBACK_SEGMENTS.

The OFFLINE option takes the rollback segment offline. If the rollback segment does not contain information necessary to rollback any active transactions, Oracle7 takes it offline immediately. If the rollback segment does contain information for active transactions, Oracle7 makes the rollback segment unavailable for future transactions and takes it offline after all the active transactions are committed or rolled back. Once the rollback segment is offline, it can be brought online by any instance.

You cannot take the SYSTEM rollback segment offline.

You can tell whether a rollback segment is online or offline by querying the data dictionary view DBA_ROLLBACK_SEGS. Online rollback segments are indicated by a STATUS value of 'IN_USE'. Offline rollback segments are indicated by a STATUS value of 'AVAILABLE'.

For more information on making rollback segments available and unavailable, see the "Managing Rollback Segments" chapter of *Oracle7 Server Administrator's Guide*.

The STORAGE clause of the ALTER ROLLBACK SEGMENT command affects future space allocation in the rollback segment. You cannot change the values of the INITIAL and MINEXTENTS for an existing rollback segment.

The SHRINK clause of the ALTER ROLLBACK SEGMENT command initiates an attempt to reduce the specified rollback segment to an optimum size. If size is not specified, then the size defaults to the OPTIMAL value of the STORAGE clause of the CREATE ROLLBACK SEGMENT command that created the rollback segment. If the OPTIMAL value was not specified, then the size defaults to the MINEXTENTS value of the STORAGE clause. The specified size in a SHRINK is valid for the execution of the command; thereafter, OPTIMUM remains unchanged. Regardless of whether a size is specified or not, the rollback segment cannot shrink to less than two extents.

You can query the DBA_ROLLBACK_SEGS tables to determine the actual size of a rollback segment after attempting to shrink a rollback segment.

For a parallel server, you can only shrink rollback segments that are online to your instance.

The SHRINK option is an *attempt* to shrink the size of the rollback segment; the success and amount of shrinkage depends on the following:

- available free space in the rollback segment
- how active transactions are holding space in the rollback segment

Example I This statement brings the rollback segment RSONE online:

```
ALTER ROLLBACK SEGMENT rsone ONLINE
```

Example II This statement changes the STORAGE parameters for RSONE:

```
ALTER ROLLBACK SEGMENT rsone
   STORAGE (NEXT 1000 MAXEXTENTS 20)
```

Example III This statement attempts to resize a rollback segment to an optimum size of one hundred megabytes:

```
ALTER ROLLBACK SEGMENT rsone
   SHRINK TO 100 M
```

Related Topics

CREATE ROLLBACK SEGMENT command on 4 – 218
CREATE TABLESPACE command on 4 – 254
STORAGE clause on 4 – 449

ALTER SEQUENCE

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 sequence numbers must be ordered

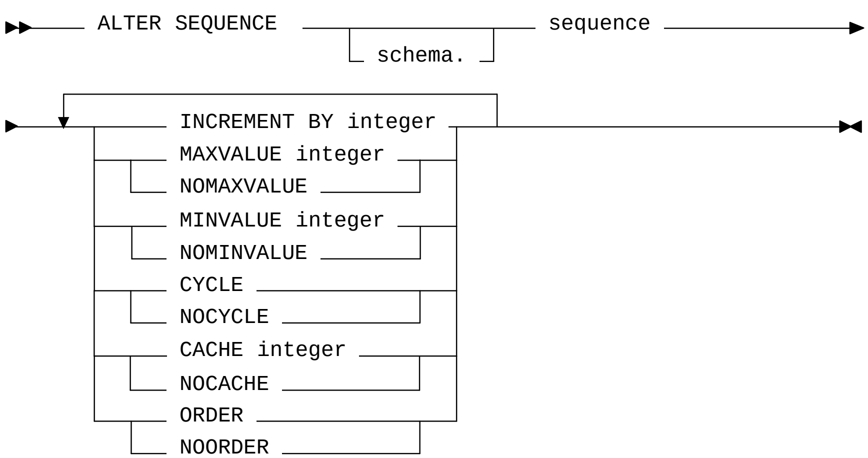
Prerequisites

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 Oracle7 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

The keywords and parameters in this command serve the same purpose that they do in the CREATE SEQUENCE command on page 4 – 224.

Usage Notes

The sequence must be dropped and recreated to restart the sequence at a different number. Only future sequence numbers are affected by the ALTER SEQUENCE command.

Some validations are performed. For example, a new MAXVALUE cannot be imposed that is less than the current sequence number.

Example I

This statement sets a new maximum value for the ESEQ sequence:

```
ALTER SEQUENCE eseq
    MAXVALUE 1500
```

Example II

This statement turns on CYCLE and CACHE for the ESEQ sequence:

```
ALTER SEQUENCE eseq
    CYCLE
    CACHE 5
```

Related Topics

CREATE SEQUENCE command on 4 – 224
DROP SEQUENCE command on 4 – 314

ALTER SESSION

Purpose

To alter your current session in one of the following:

- to enable or disable the SQL trace facility
- to enable or disable global name resolution
- to change the values of NLS parameters
- to change your DBMS session label in Trusted Oracle7
- to change the default label format for your session
- to specify the size of the cache used to hold frequently used cursors
- to enable or disable the closing of cached cursors on COMMIT or ROLLBACK
- in a parallel server, to indicate that the session must access database files as if the session was connected to another instance
- to enable, disable, and change the behavior of hash join operations
- to change the handling of remote procedure call dependencies
- to change transaction level handling
- to close a database link
- to send advice to remote databases for forcing an in–doubt distributed transaction
- to permit or prohibit stored procedures and functions from issuing COMMIT and ROLLBACK statements
- to change the goal of the cost–based optimization approach

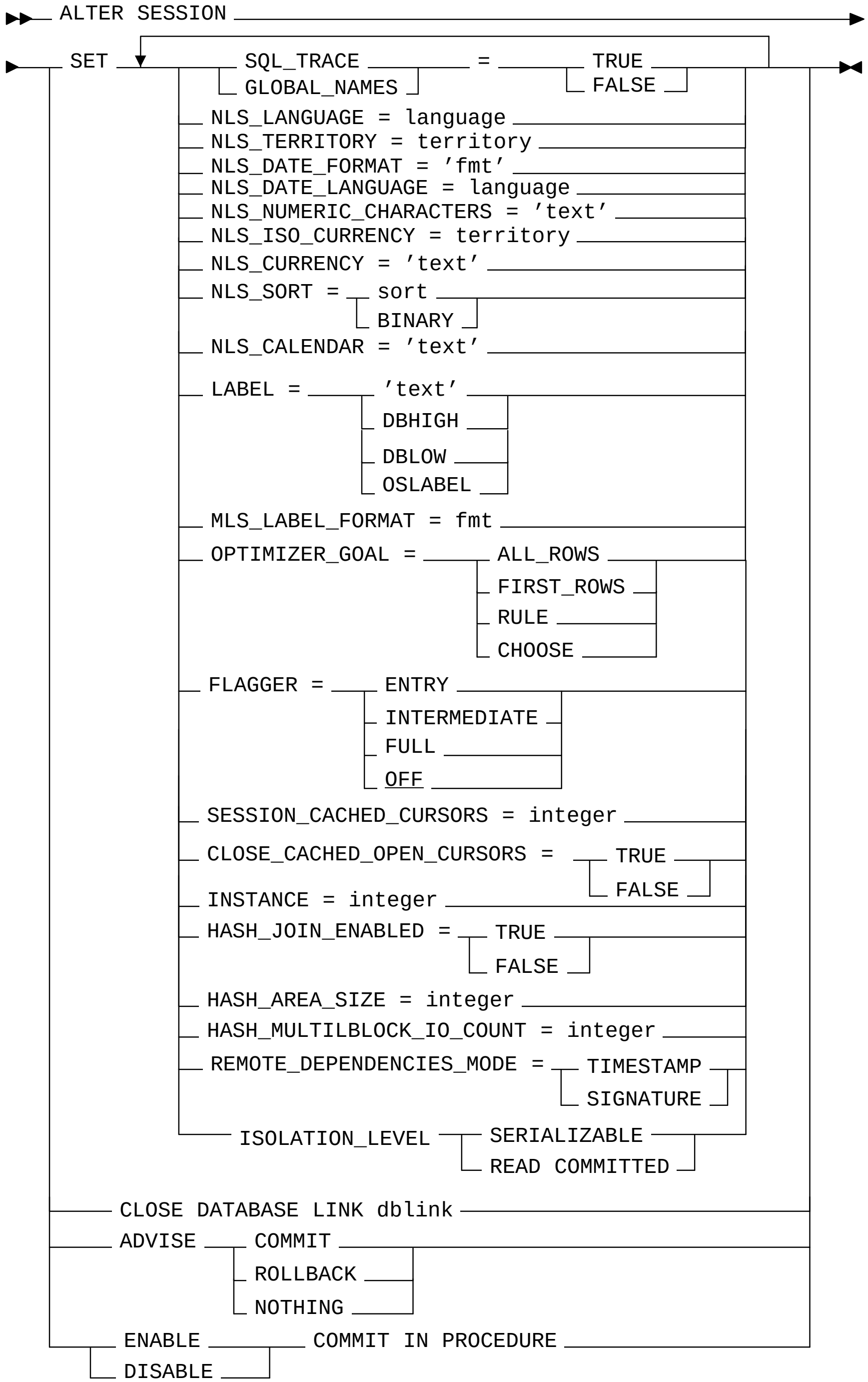
Prerequisites

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.

Syntax



ALTER SESSION
SET
SQL_TRACE
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
NLS_CALENDAR = 'text'
LABEL =
'text'
DBHIGH
DBLOW
OSLABEL
MLS_LABEL_FORMAT = fmt
OPTIMIZER_GOAL =
ALL_ROWS
FIRST_ROWS
RULE
CHOOSE
FLAGGER =
ENTRY
INTERMEDIATE
FULL
OFF
SESSION_CACHED_CURSORS = integer
CLOSE_CACHED_OPEN_CURSORS =
TRUE
FALSE
INSTANCE = integer
HASH_JOIN_ENABLED =
TRUE
FALSE
HASH_AREA_SIZE = integer
HASH_MULTILBLOCK_IO_COUNT = integer
REMOTE_DEPENDENCIES_MODE =
TIMESTAMP
SIGNATURE
ISOLATION_LEVEL
SERIALIZABLE
READ COMMITTED
CLOSE DATABASE LINK dblink
ADVISE
COMMIT
ROLLBACK
NOTHING
ENABLE
DISABLE
COMMIT IN PROCEDURE

Keywords and Parameters

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 on page 4 – 76.

NLS_LANGUAGE
: changes the language in which Oracle7 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
- B.C. and A.D. indicators
- A.M. and P.M. meridian indicators

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. The '*fmt*' value must be a date format model as specified in the section "Date Format" on page 3 – 64.

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 two different single–byte characters, and cannot be a numeric value or any of the following characters:

"+" plus
"–" minus (or hyphen)
"<" less–than
">" greater–than

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

NLS_CURRENCY
: explicitly specifies a new local currency symbol. The symbol cannot exceed 10 characters.

NLS_SORT changes the sequence into which Oracle7 sorts character values.

sort specifies the name of a linguistic sort sequence.

BINARY specifies a binary sort.

The default sort for all character sets is binary.

NLS_CALENDAR explicitly specifies a new calendar type.

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. For more information on this parameter, see the *Trusted Oracle7 Server Administrator's Guide.*

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 FIPS flagging.

ENTRY flags for SQL92 Entry level

INTERMEDIATE

flags for SQL92 Intermediate level

FULL flags for SQL92 Full level

OFF turns off flagging

SESSION_CACHED_CURSORS
specify the size of the session cache for holding frequently used cursors. *integer* specifies how many cursors can be retained in the cache.

CLOSE_OPEN_CACHED_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.

INSTANCE in a parallel server, accesses database files as if the session were connected to the instance specified by *integer.*

HASH_JOIN_ENABLED
enables or disables the use of the hash join operation in queries. The default is TRUE, which allows hash joins.

HASH_AREA_SIZE
specifies in bytes the amount of memory to use for hash join operations. The default is twice the value of the SORT_AREA_SIZE initialization parameter.

HASH_MULTIBLOCK_IO_COUNT

specifies the number of data blocks to read and write during a hash join operation. The value multiplied by the DB_BLOCK_SIZE initialization parameter should not exceed 64 kilobytes. The default value for this parameter is 1. If the multi–threaded server is used, the value is always 1, and any value given here is ignored.

REMOTE_DEPENDENCIES_MODE

specifies how dependencies of remote stored procedures are handled by the session. For more information, refer to "Remote Dependencies" in the *Oracle7 Server Application Developer's Guide*.

ISOLATION_LEVEL

specifies how transactions containing database modifications are handled.

SERIALIZABLE

transactions in the session use the serializable transaction isolation mode as specified in SQL92. That is, if a serializable transaction attempts to execute a DML statement that updates rows that are updated by another uncommitted transaction at the start of the serializable transaction, then the DML statement fails. A serializable transaction can see its own updates. The COMPATIBLE initialization parameter must be set to 7.3.0 or higher for SERIALIZABLE mode to work.

READ COMMITTED

transactions in the session will use the default Oracle transaction behavior. Thus, if the transaction contains DML that require row locks held by another transaction, then the DML statement will wait until the row locks are released.

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.

Enabling and Disabling the SQL Trace Facility

The SQL trace facility generates performance statistics for the processing of SQL statements. You can enable and disable the SQL trace facility for all sessions on an Oracle7 instance with the initialization parameter SQL_TRACE. When you begin a session, Oracle7 enables or disables the SQL trace facility based on the value of this parameter. You can subsequently enable or disable the SQL trace facility for your session with the SQL_TRACE option of the ALTER SESSION command.

For more information on the SQL trace facility, including how to format and interpret its output, see Appendix A "Performance Diagnostic Tools" of the *Oracle7 Server Tuning*.

Example I

To enable the SQL trace facility for your session, issue the following statement:

```
ALTER SESSION
    SET SQL_TRACE = TRUE
```

Using NLS Parameters

Oracle7 contains support for use in different nations and with different languages. When you start an instance, Oracle7 establishes support based on the values of initialization parameters that begin with "NLS". For information on these parameters, see *Oracle7 Server Reference*. You use the NLS clauses of the ALTER SESSION command to change NLS characteristics dynamically for your session. You can query the dynamic performance table V$NLS_PARAMETERS to see the current NLS attributes for your session.

Language for Error Messages

You can specify a new language for error messages with the NLS_LANGUAGE parameter. Note that this parameter also implicitly changes other language–related items. Oracle7 provides error messages in a wide range of languages on many platforms.

Example II

The following statement changes the language for error messages to the French:

```
ALTER SESSION
    SET NLS_LANGUAGE = French
```

Oracle7 returns error messages in French:

```
SELECT * FROM emp
ORA–00942: Table ou vue n'existe pas
```

Default Date Format

You can specify a new default date format either explicitly with the NLS_DATE_FORMAT parameter or implicitly with the NLS_TERRITORY parameter. For information on the default date format models, see the section "Date Format Models" on page 3 – 64.

Example III The following statement dynamically changes the default date format for your session to 'YYYY MM DD–HH24:MI:SS':

```
ALTER SESSION
   SET NLS_DATE_FORMAT = 'YYYY MM DD HH24:MI:SS'
```

Oracle7 uses the new default date format:

```
SELECT TO_CHAR(SYSDATE) Today
   FROM DUAL
TODAY
-------------------
1993 08 12 14:25:56
```

Language for Months and Days

You can specify a new language for names and abbreviations of months and days either explicitly with the NLS_DATE_LANGUAGE parameter or implicitly with the NLS_LANGUAGE parameter.

Example IV The following statement changes the language for date format elements to the French:

```
ALTER SESSION
   SET NLS_DATE_LANGUAGE = French

SELECT TO_CHAR(SYSDATE, 'Day DD Month YYYY') Today
   FROM DUAL

TODAY
---------------------------
Mardi    28 Février   1992
```

Decimal Character and Group Separator

You can specify new values for these number format elements either explicitly with the NLS_NUMERIC_CHARACTERS parameter or implicitly with the NLS_TERRITORY parameter:

D (decimal character)	is the character that separates the integer and decimal portions of a number.
G (group separator)	is the character that separates groups of digits in the integer portion of a number.

For information on how to use number format models, see the section "Number Format Models" on page 3 – 61.

The decimal character and the group separator can only be single–byte characters and cannot be the same character. If the decimal character is not a period (.), you must use single quotation marks to enclose all number values that appear in expressions in your SQL statements. When not using a period for the decimal point, you should always use the TO_NUMBER function to ensure that a valid number is retrieved.

Example V The following statement dynamically changes the decimal character to ',' and the group separator to '.':

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ',.'
```

Oracle7 returns these new characters when you use their number format elements:

```
SELECT TO_CHAR( SUM(sal), 'L999G999D99') Total FROM emp

TOTAL
-------------
  FF29.025,00
```

ISO Currency Symbol

You can specify a new value for the C number format element, the ISO currency symbol, either explicitly with the NLS_ISO_CURRENCY parameter or implicitly with the NLS_TERRITORY parameter. The value that you specify for these parameters is a territory whose ISO currency symbol becomes the value of the C number format element.

Example VI The following statement dynamically changes the ISO currency symbol to the ISO currency symbol for the territory America:

```
ALTER SESSION
    SET NLS_ISO_CURRENCY = America

SELECT TO_CHAR( SUM(sal), 'L999G999D99') Total
    FROM emp

TOTAL
-------------
 USD29,025.00
```

Local Currency Symbol

You can specify a new value for the L number format element, called the local currency symbol, either explicitly with the NLS_CURRENCY parameter or implicitly with the NLS_TERRITORY parameter.

Example VII The following statement dynamically changes the local currency symbol to 'DM':

```
ALTER SESSION
    SET NLS_CURRENCY = 'DM'

SELECT TO_CHAR( SUM(sal), 'L999G999D99') Total
    FROM emp

TOTAL
-------------
   DM29.025,00
```

Linguistic Sort Sequence

You can specify a new linguistic sort sequence or a binary sort either explicitly with the NLS_SORT parameter or implicitly with the NLS_LANGUAGE parameter.

Example VIII

The following statement dynamically changes the linguistic sort sequence to Spanish:

```
ALTER SESSION
    SET NLS_SORT = XSpanish
```

Oracle7 sorts character values based on their position in the Spanish linguistic sort sequence.

Changing the Optimization Approach and Goal

The Oracle7 optimizer can use either of these approaches to optimize a SQL statement:

rule–based	The optimizer optimizes a SQL statement based on the indexes and clusters associated with the accessed tables, the syntactic constructs of the statement, and a heuristically ranked list of these constructs.
cost–based	The optimizer optimizes a SQL statement by considering statistics describing the tables, indexes, and clusters accessed by the statement as well as the information considered with the rule–based approach.

With the cost–based approach, the optimizer can optimize a SQL statement with one of these goals:

best throughput	or the minimal time necessary to return all rows accessed by the statement
best response time	or the minimal time necessary to return the first row accessed by the statement

When you start your instance, the optimization approach is established by the initialization parameter OPTIMIZER_MODE. If this parameter establishes the cost–based approach, the default goal is best throughput. You can subsequently change the optimization approach or the goal of the cost–based optimization approach for your session with the OPTIMIZER_GOAL parameter.

Example IX The following statement changes the goal of the cost–based approach to best response time:

```
ALTER SESSION
    SET OPTIMIZER_GOAL = FIRST_ROWS
```

For information on how to choose a goal for the cost–based approach based on the characteristics of your application, see the *Oracle7 Server Tuning*.

FIPS Flagging

FIPS flagging causes an error message to be generated when a SQL statement is issued that is an extension of ANSI SQL92. In Oracle7, Release 7.3, there is currently no difference between Entry, Intermediate, or Full level flagging. Once flagging is set in a session, a subsequent ALTER SESSION SET FLAGGER commands will work, but generates the message, ORA–00097. This allows FIPS flagging to be altered without disconnecting the session.

Caching Session Cursors

If an application repeatedly issues parse calls on the same set of SQL statements, the reopening of the session cursors can affect performance. The ALTER SESSION SET SESSION_CACHED_CURSORS command allows frequently used session cursors to be stored in a session cache even if they are closed. This is particularly useful for some Oracle7 tools. For example, Oracle Forms applications close all session cursors associated with a form when switching to another form; in this case, frequently used cursors would not have to be reparsed.

Oracle7 uses the shared SQL area to determine if more than three parse requests were issued on a given statement. If so, Oracle7 moves the cursor into the session cursor cache. Subsequent requests to parse that SQL statement by the same session will find the cursor in the session cursor cache.

Session cursors are automatically cached if the initialization parameter, SESSION_CACHED_CURSORS is set to a positive value. This parameter specifies the maximum number of session cursors to be kept in the cache. A least recently used algorithm ages out entries in the cache to make room for new entries when needed. You use the ALTER SESSION SET SESSION_CACHED_CURSORS command to dynamically enable session cursor caching.

For more information on session cursor caching, see the *Oracle7 Server Tuning*.

Accessing the Database as if Connected to Another Instance in a Parallel Server

For optimum performance, each instance of a parallel server uses its own private rollback segments, freelist groups, and so on. A database is usually designed for a parallel server such that users connect to a particular instance and access data that is partitioned primarily for their use. If the users for that instance must connect to another instance, the data partitioning can be lost. The ALTER SESSION SET INSTANCE command allows users to access an instance as if they were connected to their usual instance.

Closing Database Links

A database link allows you to access a remote database in DELETE, INSERT, LOCK TABLE, SELECT, and UPDATE statements. When you issue a statement that uses a database link, Oracle7 creates a session for you on the remote database using the database link. The connection remains open until you end your local session or until the number of database links for your session exceeds the value of the initialization parameter OPEN_LINKS.

You can use the CLOSE DATABASE LINK clause of the ALTER SESSION command to explicitly close a database link if you do not plan to use it again in your session. You may want to explicitly close a database link if the network overhead associated with leaving it open is costly. Before closing a database link, you must first close all cursors that use the link and then end your current transaction if it uses the link.

Example X

This example updates the employee table on the SALES database using a database link, commits the transaction, and explicitly closes the database link:

```
UPDATE emp@sales
    SET sal = sal + 200
    WHERE empno = 9001
COMMIT
ALTER SESSION
    CLOSE DATABASE LINK sales
```

Offering Advice for Forcing In–doubt Distributed Transactions

If a network or machine failure occurs during the commit process for a distributed transaction, the state of the transaction may be unknown, or in–doubt. The transaction can be manually committed or rolled back on each database involved in the transaction with the FORCE clause of the COMMIT or ROLLBACK commands.

Before committing a distributed transaction, you can use the ADVISE clause of the ALTER SESSION command to send advice to a remote database in the event a distributed transaction becomes in–doubt. If the transaction becomes in–doubt, the advice appears in the ADVICE column of the DBA_2PC_PENDING view on the remote database. The administrator of that database can then use this advice to decide whether to commit or roll back the transaction on the remote database. For more information on distributed transactions and how to decide whether to commit or roll back in–doubt distributed transactions, see the “Database Administration” chapter of *Oracle7 Server Distributed Systems, Volume I*.

You issue multiple ALTER SESSION statements with the ADVISE clause in a single transaction. Each such statement sends advice to the databases referenced in the following statements in the transaction until another such statement is issued. This allows you to send different advice to different databases.

Example XI

This transaction inserts an employee record into the EMP table on the database identified by the database link SITE1 and deletes an employee record from the EMP table on the database identified by SITE2:

```
ALTER SESSION
    ADVISE COMMIT
INSERT INTO emp@site1
    VALUES (8002, 'FERNANDEZ', 'ANALYST', 7566,
            TO_DATE('04-OCT-1992', 'DD-MON-YYYY'), 3000, NULL, 20)
ALTER SESSION
    ADVISE ROLLBACK
DELETE FROM emp@site2
    WHERE empno = 8002
COMMIT
```

This transaction has two ALTER SESSION statements with the ADVISE clause. If the transaction becomes in–doubt, SITE1 is sent the advice ’COMMIT’ by virtue of the first ALTER SESSION statement and SITE2 is sent the advice ’ROLLBACK’ by virtue of the second.

Enabling and Disabling Transaction Control in Procedures and Stored Functions

Since procedures and stored functions are written in PL/SQL, they can issue COMMIT and ROLLBACK statements. If your application performs record management that would be disrupted by a COMMIT or ROLLBACK statement not issued directly by the application itself, you may want to prevent procedures and stored functions called during your session from issuing these statements. You can do this with the following statement:

```
ALTER SESSION DISABLE COMMIT IN PROCEDURE
```

If you subsequently call a procedure or a stored function that issues a COMMIT or ROLLBACK statement, Oracle7 returns an error and does not commit or roll back the transaction. SQL*Forms automatically prohibits COMMIT and ROLLBACK statements in procedures and stored functions.

You can subsequently allow procedures and stored functions to issue COMMIT and ROLLBACK statements in your session by issuing the following statement:

```
ALTER SESSION ENABLE COMMIT IN PROCEDURE
```

This command does not apply to database triggers. Triggers can never issue COMMIT or ROLLBACK statements.

Related Topics

"Tuning SQL Statements" and "Performance Diagnostic Tools" of the *Oracle7 Server Tuning Guide*.

ALTER SNAPSHOT

Purpose

To alter a snapshot in one of the following ways:

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

Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the snapshot's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

To change the storage characteristics of the internal table that Oracle7 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 on page 4 – 89.

Syntax

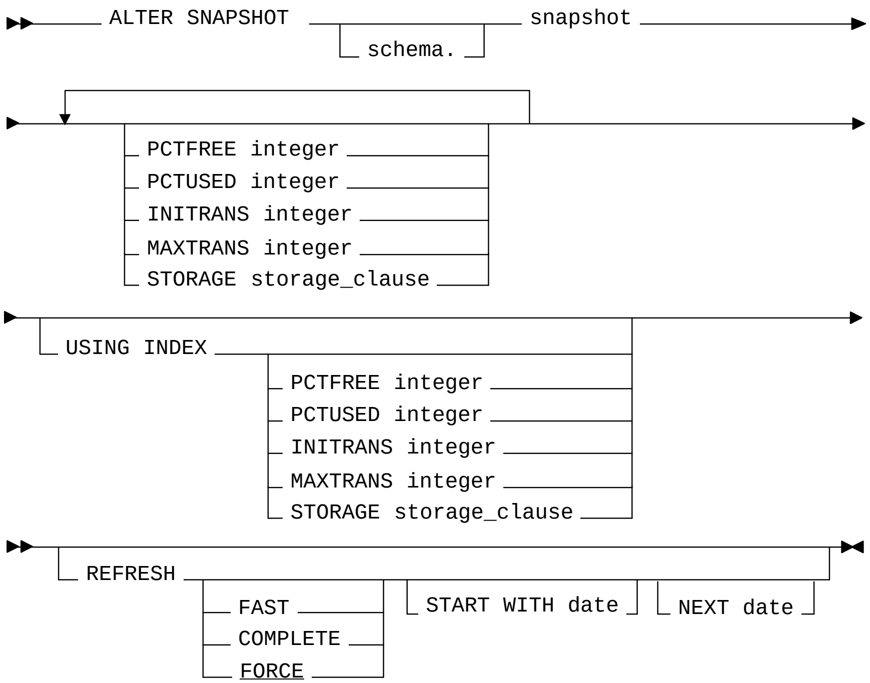
Keywords and Parameters

schema — is the schema containing the snapshot. If you omit *schema*, Oracle7 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 Oracle7 uses to maintain the snapshot's data. For information on the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters, see the CREATE TABLE command on page 4 – 245.

STORAGE — changes the storage characteristics of the internal table Oracle7 uses to maintain the snapshot's data. See the STORAGE clause on page 4 – 449.



USING INDEX — changes the value of INITRANS, MAXTRANS, and STORAGE parameters for the index Oracle7 uses to maintain the snapshot's data. If USING INDEX is not specified then the index is written to the user's default tablespace.

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 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. Oracle7 decides whether a fast refresh is possible at refresh time.

If you omit the FAST, COMPLETE, and FORCE options, Oracle7 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.

Usage Notes

For more information on snapshots, including refreshing snapshots, see the CREATE SNAPSHOT command on page 4 – 230.

Example I

The following statement changes the automatic refresh mode for the HQ_EMP snapshot to FAST:

```
ALTER SNAPSHOT hq_emp
    REFRESH FAST
```

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

Because the REFRESH clause does not specify START WITH or NEXT values, the refresh intervals established by the REFRESH clause when the HQ_EMP snapshot was created or last altered are still used.

Example II

The following statement stores a new interval between automatic refreshes for the BRANCH_EMP snapshot:

```
ALTER SNAPSHOT branch_emp
    REFRESH NEXT SYSDATE+7
```

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

At the time of the next automatic refresh, Oracle7 refreshes the snapshot, evaluates the NEXT expression SYSDATE+7 to determine the next automatic refresh time, and continues to automatically refresh the snapshot once a week.

Because the REFRESH clause does not explicitly specify a refresh mode, Oracle7 continues to use the refresh mode specified by the REFRESH clause of a previous CREATE SNAPSHOT or ALTER SNAPSHOT statement.

Example III The following statement specifies a new refresh mode, next refresh time, and new interval between automatic refreshes of the SF_EMP snapshot:

```
ALTER SNAPSHOT sf_emp
    REFRESH COMPLETE
    START WITH TRUNC(SYSDATE+1) + 9/24
    NEXT SYSDATE+7
```

The START WITH value establishes the next automatic refresh for the snapshot to be 9:00am tomorrow. At that point, Oracle7 performs a fast refresh of the snapshot, evaluates the NEXT expression, and subsequently refreshes the snapshot every week.

Related Topics

CREATE SNAPSHOT command on 4 – 230
DROP SNAPSHOT command on 4 – 315

ALTER SNAPSHOT LOG

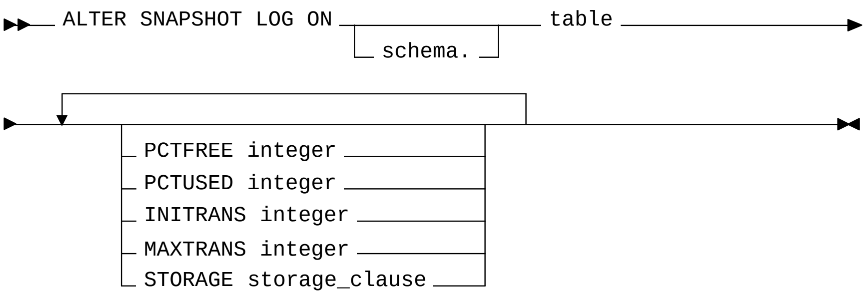
Purpose

Changes the storage characteristics of a 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 later in this chapter.

Syntax



Keywords and Parameters

schema	is the schema containing the snapshot log and its master table. If you omit *schema*, Oracle7 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 on page 4 – 245.
STORAGE	changes the storage characteristics of the snapshot log. See the STORAGE clause on page 4 – 449.

Usage Notes

For more information on snapshot logs, see the CREATE SNAPSHOT LOG command on page 4 – 238.

Example The following statement changes the MAXEXTENTS value of a snapshot log:

```
ALTER SNAPSHOT LOG dept   STORAGE MAXEXTENTS 50
```

Related Topics

CREATE SNAPSHOT command on 4 – 230
CREATE SNAPSHOT LOG command on 4 – 238
DROP SNAPSHOT LOG command on 4 – 316

ALTER SYSTEM

Purpose

To dynamically alter your Oracle7 instance in one of the following ways:

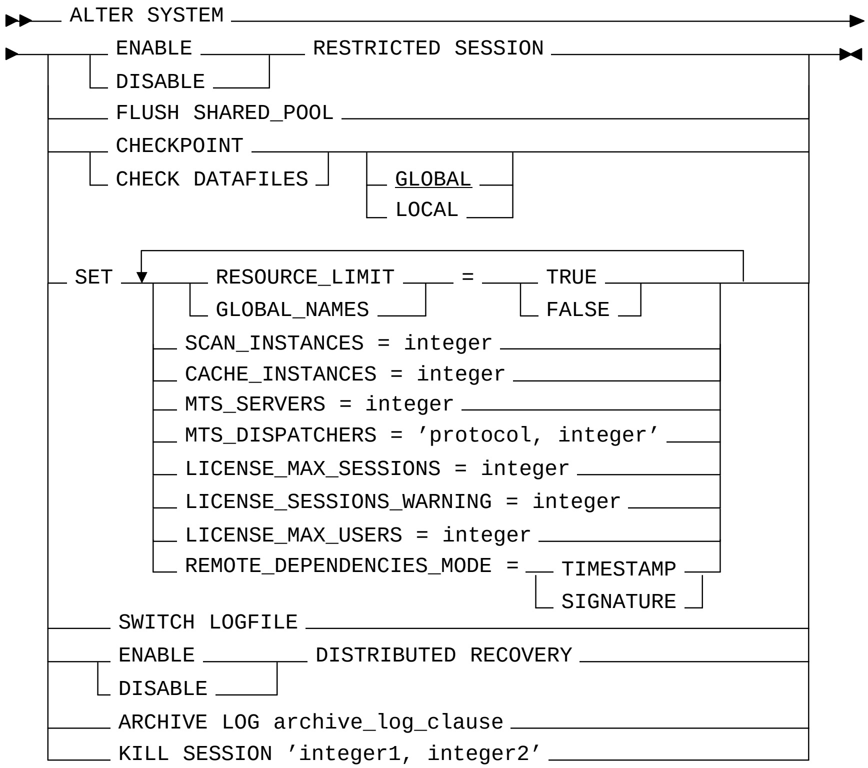
- to restrict logons to Oracle7 to only those users with RESTRICTED SESSION system privilege
- to clear all data from the shared pool in the System Global Area (SGA)
- to explicitly perform a checkpoint
- to verify access to data files
- to enable or disable resource limits
- to enable or disable global name resolution
- to manage shared server processes or dispatcher processes for the multi–threaded server architecture
- to dynamically change or disable limits or thresholds for concurrent usage licensing and named user licensing
- to explicitly switch redo log file groups
- 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 terminate a session

Prerequisites

You must have ALTER SYSTEM system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must be the equivalent of DBHIGH.

Syntax



Keywords and Parameters

You can use the following 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 Oracle7.

DISABLE RESTRICTED SESSION

reverses the effect of the ENABLE RESTRICTED SESSION option, allowing all users with CREATE SESSION system privilege to logon to Oracle7.

FLUSH SHARED_POOL

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

You can use the following 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, Oracle7 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, Oracle7 uses GLOBAL by default.

You can only use the following 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.

SCAN_INSTANCES

in a parallel server, specify the number of instances to participate in parallelized operations.

CACHE_INSTANCES

in a parallel server, specify the number of instances that will cache a table.

For more information on parallel operations, see the "Parallel Query Option" chapter of *Oracle7 Server Tuning*.

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 Oracle7 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 concurrent users on your database. A value of 0 disables the limit.

REMOTE_DEPENDENCIES_MODE
specifies how dependencies of remote stored procedures are handled by the server. For more information, refer to "Remote Dependencies" in the *Oracle7 Server Application Developer's Guide.*

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. See the ARCHIVE LOG clause on page 4 – 124.

KILL SESSION
terminates a session. You must identify the session with both of the following values from the V$SESSION view:

integer1 is the value of the SID column.

integer2 is the value of the SERIAL# column.

Restricting Logons

By default, any user granted CREATE SESSION system privilege can log on to Oracle7. The ENABLE RESTRICTED SESSION option of the ALTER SYSTEM command prevents logons by all users except those having RESTRICTED SESSION system privilege. Existing sessions are not terminated.

You may want to restrict logons if you are performing application maintenance and you want only application developers with RESTRICTED SESSION system privilege to log on. To restrict logons, issue the following statement:

```
ALTER SYSTEM
   ENABLE RESTRICTED SESSION
```

You can then terminate any existing sessions using the KILL SESSION clause of the ALTER SYSTEM command.

After performing maintenance on your application, issue the following statement to allow any user with CREATE SESSION system privilege to log on:

```
ALTER SYSTEM
   DISABLE RESTRICTED SESSION
```

Clearing the Shared Pool

The FLUSH SHARED_POOL option of the ALTER SYSTEM command clears all information from the shared pool in the System Global Area (SGA). The shared pool stores this information:

- cached data dictionary information
- shared SQL and PL/SQL areas for SQL statements, stored procedures, functions, packages, and triggers

You might want to clear the shared pool before beginning performance analysis. To clear the shared pool, issue the following statement:

```
ALTER SYSTEM
   FLUSH SHARED_POOL
```

The above statement does not clear shared SQL and PL/SQL areas for SQL statements, stored procedures, functions, packages, or triggers that are currently being executed or for SQL SELECT statements for which all rows have not yet been fetched.

Performing a Checkpoint

The CHECKPOINT clause of the ALTER SYSTEM command explicitly forces Oracle7 to perform a checkpoint. You can force a checkpoint if you want to ensure that all changes made by committed transactions are written to the data files on disk. For more information on checkpoints, see the "Recovery Structures" chapter of *Oracle7 Server Concepts*. If you are using Oracle7 with the Parallel Server option in parallel mode, you can specify either the GLOBAL option to perform a checkpoint on all instances that have opened the database or the LOCAL option to perform a checkpoint on only your instance.

The following statement forces a checkpoint:

```
ALTER SYSTEM
   CHECKPOINT
```

Oracle7 does not return control to you until the checkpoint is complete.

Checking Data Files

The CHECK DATAFILES clause of the ALTER SYSTEM command verifies access to all online data files. If any data file is not accessible, Oracle7 writes a message to an ALERT file. You may want to perform this operation after fixing a hardware problem that prevented an instance from accessing a data file. For more information on using this clause, see *Oracle7 Parallel Server Concepts & Administration*.

The following statement verifies that all instances that have opened the database can access all online data files:

```
ALTER SYSTEM
    CHECK DATAFILES GLOBAL
```

Using Resource Limits

When you start an instance, Oracle7 enables or disables resource limits based on the value of the initialization parameter RESOURCE_LIMIT. You can issue an ALTER SYSTEM statement with the RESOURCE_LIMIT option to enable or disable resource limits for subsequent sessions.

Enabling resource limits only causes Oracle7 to enforce the resource limits assigned to users. To choose resource limit values for a user, you must create a *profile*, or a set of limits, and assign that profile to the user. For more information on this process, see the CREATE PROFILE command on page 4 – 210 and the CREATE USER command on page 4 – 267.

This ALTER SYSTEM statement dynamically enables resource limits:

```
ALTER SYSTEM
    SET RESOURCE_LIMIT = TRUE
```

Enabling and Disabling Global Name Resolution

When you start an instance, Oracle7 determines whether to enforce global name resolution for remote objects accessed in SQL statements based on the value of the initialization parameter GLOBAL_NAMES. You can subsequently enable or disable global names resolution while your instance is running with the GLOBAL_NAMES parameter of the ALTER SYSTEM command. You can also enable or disable global name resolution for your session with the GLOBAL_NAMES parameter of the ALTER SESSION command discussed earlier in this chapter.

It is recommended that you enable global name resolution. For more information on global name resolution and how Oracle7 enforces it, see section "Referring to Objects in Remote Databases" on page 2 – 11 and *Oracle7 Server Distributed Systems, Volume I.*

Managing Processes for the Multi–Threaded Server

When you start your instance, Oracle7 creates shared server processes and dispatcher processes for the multi–threaded server architecture based on the values of the following initialization parameters:

MTS_SERVERS
: This parameter specifies the initial and minimum number of shared server processes. Oracle7 may automatically change the number of shared server processes if the load on the existing processes changes. While your instance is running, the number of shared server processes can vary between the values of the initialization parameters MTS_SERVERS and MTS_MAX_SERVERS.

MTS_DISPATCHERS
: This parameter specifies one or more network protocols and the number of dispatcher processes for each protocol.

For more information on the multi–threaded server architecture, see *Oracle7 Server Concepts*.

You can subsequently use the MTS_SERVERS and MTS_DISPATCHERS parameters of the ALTER SYSTEM command to perform one of the following operations while the instance is running:

To create additional shared server processes:

You can cause Oracle7 to create additional shared server processes by increasing the minimum number of shared server processes.

To terminate existing shared server processes:

Oracle7 terminates the shared server processes after finishing processing their current calls,unless the load on the server processes is so high that it cannot be managed by the remaining processes.

To create more dispatcher processes for a specific protocol:

You can create additional dispatcher processes up to a maximum across all protocols specified by the initialization parameter MTS_MAX_DISPATCHERS.

You cannot use this command to create dispatcher processes for network protocols that are not specified by the initialization parameter MTS_DISPATCHERS. To create dispatcher processes for a new protocol, you must change the value of the initialization parameter.

To terminate existing dispatcher processes for a specific protocol:

Oracle7 terminates the dispatcher processes only after their current user processes disconnect from the instance.

Example I The following statement changes the minimum number of shared server processes to 25:

```
ALTER SYSTEM
    SET MTS_SERVERS = 25
```

If there are currently fewer than 25 shared server processes, Oracle7 creates more. If there are currently more than 25, Oracle7 terminates some of them when they are finished processing their current calls if the load could be managed by the remaining 25.

Example II The following statement dynamically changes the number of dispatcher processes for the TCP/IP protocol to 5 and the number of dispatcher processes for the DECNET protocol to 10:

```
ALTER SYSTEM
    SET MTS_DISPATCHERS = 'TCP, 5'
        MTS_DISPATCHERS = 'DECnet, 10'
```

If there are currently fewer than 5 dispatcher processes for TCP, Oracle7 creates new ones. If there are currently more than 5, Oracle7 terminates some of them after the connected users disconnect.

If there are currently fewer than 10 dispatcher processes for DECnet, Oracle7 creates new ones. If there are currently more than 10, Oracle7 terminates some of them after the connected users disconnect.

If there are currently existing dispatchers for another protocol, the above statement does not affect the number of dispatchers for this protocol.

Using Licensing Limits

Oracle7 enforces concurrent usage licensing and named user licensing limits specified by your Oracle7 license. When you start your instance, Oracle7 establishes the licensing limits based on the values of the following initialization parameters:

LICENSE_MAX_SESSIONS
: This parameter establishes the concurrent usage licensing limit, or the limit for concurrent sessions. Once this limit is reached, only users with RESTRICTED SESSION system privilege can connect.

LICENSE_SESSIONS_WARNING
: This parameter establishes a warning threshold for concurrent usage. Once this threshold is reached, Oracle7 writes a warning message to the database ALERT file for each subsequent session. Also, users with RESTRICTED SESSION system privilege receive warning messages when they begin subsequent sessions.

LICENSE_MAX_USERS
: This parameter establishes the limit for users connected to your database. Once this limit for users is reached, more users cannot connect.

You can subsequently use the LICENSE_MAX_SESSIONS, LICENSE_SESSIONS_WARNING, and LICENSE_MAX_USERS parameters of the ALTER SYSTEM command to dynamically change or disable limits or thresholds while your instance is running. Do not disable or raise session or user limits unless you have appropriately upgraded your Oracle7 license. For information on upgrading your license, contact your Oracle sales representative.

New limits apply only to future sessions and users:

- If you reduce the limit on sessions below the current number of sessions, Oracle7 does not end existing sessions to enforce the new limit. Users without RESTRICTED SESSION system privilege can only begin new sessions when the number of sessions falls below the new limit.
- If you reduce the warning threshold for sessions below the current number of sessions, Oracle7 writes a message to the ALERT file for all subsequent sessions.
- You cannot reduce the limit on users below the current number of users created for the database.

Example III The following statement dynamically changes the limit on sessions for your instance to 64 and the warning threshold for sessions on your instance to 54:

```
ALTER SYSTEM
    SET LICENSE_MAX_SESSIONS = 64
       LICENSE_SESSIONS_WARNING = 54
```

If the number of sessions reaches 54, Oracle7 writes a warning message to the ALERT file for each subsequent session. Also, users with RESTRICTED SESSION system privilege receive warning messages when they begin subsequent sessions.

If the number of sessions reaches 64, only users with RESTRICTED SESSION system privilege can begin new sessions until the number of sessions falls below 64 again.

Example IV The following statement dynamically disables the limit for sessions on your instance:

```
ALTER SYSTEM
    SET LICENSE_MAX_SESSIONS = 0
```

After you issue the above statement, Oracle7 no longer limits the number of sessions on your instance.

Example V The following statement dynamically changes the limit on the number of users in the database to 200:

```
ALTER SYSTEM
    SET LICENSE_MAX_USERS = 200
```

After you issue the above statement, Oracle7 prevents the number of users in the database from exceeding 200.

Switching Redo Log File Groups

The SWITCH LOGFILE option of the ALTER SYSTEM command explicitly forces Oracle7 to begin writing to a new redo log file group, regardless of whether the files in the current redo log file group are full. You may want to force a log switch to drop or rename the current redo log file group or one of its members, since you cannot drop or rename a file while Oracle7 is writing to it. The forced log switch only affects your instance’s redo log thread. Note that when you force a log switch, Oracle7 begins to perform a checkpoint. Oracle7 returns control to you immediately rather than when the associated checkpoint is complete.

The following statement forces a log switch:

```
ALTER SYSTEM
    SWITCH LOGFILE
```

Enabling Distributed Recovery

Oracle7 allows you to perform distributed transactions, or transactions that modify data on multiple databases. If a network or machine failure occurs during the commit process for a distributed transaction, the state of the transaction may be unknown, or *in–doubt*. Once the failure has been corrected and the network and its nodes are back online, Oracle7 recovers the transaction.

If you are using Oracle7 in multiple–process mode, this distributed recovery is performed automatically. If you are using Oracle7 in single–process (single user) mode, such as on the MS–DOS operating system, you must explicitly initiate distributed recovery with the following statement.

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY
```

You may need to issue the above statement more than once to recover an in–doubt transaction, especially if the remote node involved in the transaction is not accessible. In–doubt transactions appear in the data dictionary view DBA_2PC_PENDING. You can tell that the transaction is recovered when it no longer appears in DBA_2PC_PENDING. For more information about distributed transactions and distributed recovery, see *Oracle7 Server Distributed Systems, Volume I*.

Disabling Distributed Recovery

You can use the following statement to disable distributed recovery in both single–process and multiprocess mode:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY
```

You may want to disable distributed recovery for demonstration purposes. You can then enable distributed recovery again by issuing an ALTER SYSTEM statement with the ENABLE DISTRIBUTED RECOVERY clause.

Terminating a Session

The KILL SESSION clause of the ALTER SYSTEM command terminates a session, immediately performing the following tasks:

- rolling back its current transactions
- releasing all of its locks
- freeing all of its resources

You may want to kill the session of a user that is holding resources needed by other users. The user receives an error message indicating that the session has been killed and can no longer make calls to the database without beginning a new session. You can only kill a session on the same instance as your current session.

If you try to kill a session that is performing some activity that must be completed, such as waiting for a reply from a remote database or rolling back a transaction, Oracle7 waits for this activity to complete, kills the session, and then returns control to you. If the waiting lasts as long as a minute, Oracle7 marks the session to be killed and returns control to you with a message indicating that the session is marked to be killed. Oracle7 then kills the session when the activity is complete.

Example VI Consider this data from the V$SESSION dynamic performance table:

```
SELECT sid, serial#, username
    FROM v$session

        SID    SERIAL# USERNAME
---------- ---------- ------------------------------
         1          1
         2          1
         3          1
         4          1
         5          1
         7          1
         8         28 OPS$BQUIGLEY
        10        211 OPS$SWIFT
        11         39 OPS$OBRIEN
        12         13 SYSTEM
        13          8 SCOTT
```

The following statement kills the session of the user SCOTT using the SID and SERIAL# values from V$SESSION:

```
ALTER SYSTEM
    KILL SESSION '13, 8'
```

Related Topics

ALTER SESSION command on 4 – 55
CREATE PROFILE command on 4 – 210
CREATE USER command on 4 – 267

ALTER TABLE

Purpose

To alter the definition of a table in one of the following 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
- to explicitly deallocate the unused space of a table
- to allow or disallow writing to a table
- to modify the degree of parallelism for a table

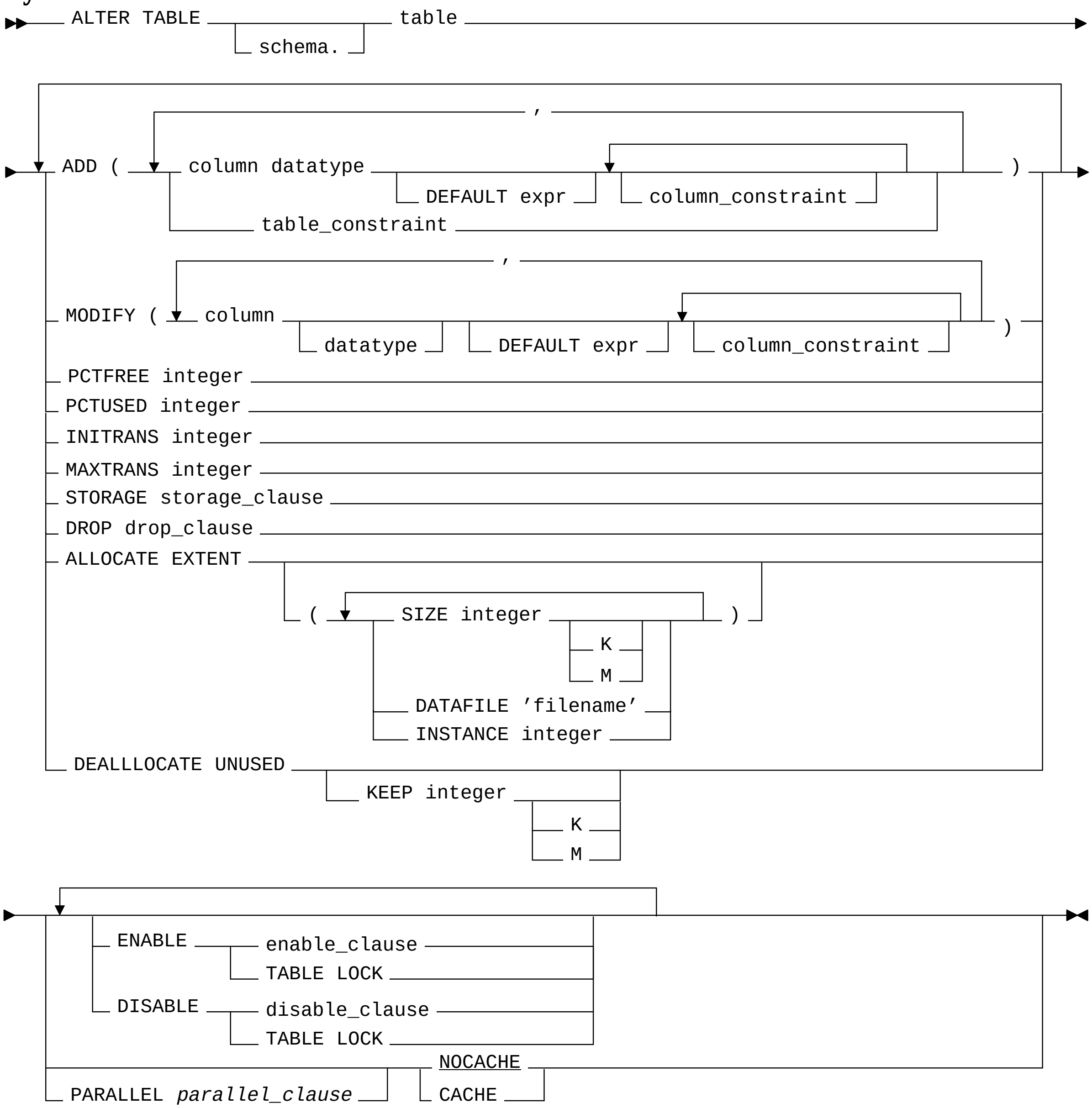
Prerequisites

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 Oracle7 in DBMS MAC mode, your DBMS label must match the table's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax


ALTER TABLE
schema.
table
ADD (
,
column datatype
DEFAULT expr
column_constraint
table_constraint
)
MODIFY (
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
)
DEALLLOCATE UNUSED
KEEP integer
K
M
ENABLE
enable_clause
TABLE LOCK
DISABLE
disable_clause
TABLE LOCK
NOCACHE
PARALLEL parallel_clause
CACHE

Keywords and Parameters

schema	is the schema containing the table. If you omit *schema*, Oracle7 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 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. You can only omit the datatype if the statement also designates the column as part of the foreign key of a referential integrity constraint. Oracle7 automatically assigns the column the same datatype as the corresponding column of the referenced key of the referential integrity constraint.
DEFAULT	specifies a default value for a new column or a new default for an existing column. Oracle7 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. The column must also be long enough to hold the default value. 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. See the syntax of *column_constraint* on page 4 – 152.
table_constraint	adds an integrity constraint to the table. See the syntax of *table_constraint* on page 4 – 152.
PCTFREE PCTUSED INITRANS MAXTRANS	changes the value of specified parameters for the table. See the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters of the CREATE TABLE command on page 4 – 245.

STORAGE — changes the storage characteristics of the table. See the STORAGE clause beginning on page 4 – 449.

DROP — drops an integrity constraint. See the DROP clause on page 4 – 299.

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, Oracle7 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, Oracle7 chooses the data file.

INSTANCE — makes the new extent available to the freelist group associated with the specified instance. If the instance number exceeds the maximum number of freelist groups, the former is divided by the latter, and the remainder is used to identify the freelist group to be used. An instance is identified by the value of its initialization parameter INSTANCE_NUMBER. If you omit this parameter, the space is allocated to the table, but is not drawn from any particular freelist group. Rather the master freelist is used, and space is allocated as needed. For more information, see *Oracle7 Server Concepts*. Only use this parameter if you are using Oracle7 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.

DEALLOCATE UNUSED

explicitly deallocate unused space at the end of the table and make the freed space available for other segments. You can free only unused space above the high–water mark. If KEEP is omitted, all unused space is freed. For more information, see the *deallocate_clause.*

KEEP

specifies the number of bytes above the high–water mark that the table will have after deallocation. If the number of remaining extents are less than MINEXTENTS, then MINEXTENTS is set to the current number of extents. If the initial extent becomes smaller than INITIAL, then INITIAL is set to the value of the current initial extent.

ENABLE *enable_clause*

enables a single integrity constraint or all triggers associated with the table. See the ENABLE clause on page 4 – 326.

ENABLE TABLE LOCK

enables DML and DDL locks on a table in a parallel server environment. For more information, see *Oracle7 Parallel Server Concepts & Administration*.

DISABLE *disable_clause*

disables a single integrity constraint or all triggers associated with the table. See the DISABLE clause on page 4 – 295.

Integrity constraints specified in DISABLE 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, Oracle7 enables it by default.

DISABLE TABLE LOCK

disables DML and DDL locks on a table to improve performance in a parallel server environment. For more information, see *Oracle7 Parallel Server Concepts & Administration*.

PARALLEL

specifies the degree of parallelism for the table. See the *parallel_clause* on page 4 – 378.

CACHE

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

NOCACHE

Specifies that the blocks retrieved for this table are placed at the least recently used end of the LRU list in the buffer cache when a full table scan is performed. This is the default behavior.

Adding Columns

If you use the ADD clause to add a new column to the table, then the initial value of each row for the new column is null. You can add a column with a NOT NULL constraint only to a table that contains no rows.

If you create a view with a query that uses the asterisk (*) in the select list to select all columns from the base table and you subsequently add columns to the base table, Oracle7 will not automatically add the new column to the view. To add the new column to the view, you can re–create the view using the CREATE VIEW command with the OR REPLACE option.

Operations performed by the ALTER TABLE command can cause Oracle7 to invalidate procedures and stored functions that access the table. For information on how and when Oracle7 invalidates such objects, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

Modifying Column Definitions

You can use the MODIFY clause to change any of the following parts of a column definition:

- datatype
- size
- default value
- NOT NULL column constraint

The MODIFY clause need only specify the column name and the modified part of the definition, rather than the entire column definition.

Datatypes and Sizes

You can change a CHAR column to VARCHAR2 (or VARCHAR) and a VARCHAR2 (or VARCHAR) to CHAR only if the column contains nulls in all rows or if you do not attempt to change the column size. You can change any column's datatype or decrease any column's size if all rows for the column contain nulls. However, you can always increase the size of a character or raw column or the precision of a numeric column.

Default Values

A change to a column's default value only affects rows subsequently inserted into the table. Such a change does not change default values previously inserted.

Integrity Constraints

The only type of integrity constraint that you can add to an existing column using the MODIFY clause with the column constraint syntax is a NOT NULL constraint. However, you can define other types of integrity constraints (UNIQUE, PRIMARY KEY, referential integrity, and CHECK constraints) on existing columns using the ADD clause and the table constraint syntax.

You can define a NOT NULL constraint on an existing column only if the column contains no nulls.

Example I The following statement adds a column named THRIFTPLAN of datatype NUMBER with a maximum of seven digits and two decimal places and a column named LOANCODE of datatype CHAR with a size of one and a NOT NULL integrity constraint:

```
ALTER TABLE emp
    ADD (thriftplan NUMBER(7,2),
         loancode CHAR(1) NOT NULL)
```

Example II The following statement increases the size of the THRIFTPLAN column to nine digits:

```
ALTER TABLE emp
    MODIFY (thriftplan NUMBER(9,2))
```

Because the MODIFY clause contains only one column definition, the parentheses around the definition are optional.

Example III The following statement changes the values of the PCTFREE and PCTUSED parameters for the EMP table to 30 and 60, respectively:

```
ALTER TABLE emp
    PCTFREE 30
    PCTUSED 60
```

Example IV The following statement allocates an extent of 5 kilobytes for the EMP table and makes it available to instance 4:

```
ALTER TABLE emp
    ALLOCATE EXTENT (SIZE 5K INSTANCE 4)
```

Because this command omits the DATAFILE parameter, Oracle7 allocates the extent in one of the data files belonging to the tablespace containing the table.

Example V

This example modifies the BAL column of the ACCOUNTS table so that it has a default value of 0:

```
ALTER TABLE accounts
   MODIFY (bal  DEFAULT 0)
```

If you subsequently add a new row to the ACCOUNTS table and do not specify a value for the BAL column, the value of the BAL column is automatically 0:

```
INSERT INTO accounts(accno, accname)
   VALUES (accseq.nextval, 'LEWIS')
 SELECT *
   FROM accounts
   WHERE accname = 'LEWIS'
  ACCNO ACCNAME BAL
------ ------- ---
815234 LEWIS     0
```

Other Examples

For examples of defining integrity constraints with the ALTER TABLE command, see the CONSTRAINT clause beginning on page 4 – 152.

For examples of enabling, disabling, and dropping integrity constraints and triggers with the ALTER TABLE command, see the ENABLE clause on page 4 – 326, the DISABLE clause on page 4 – 295, and DROP clause on page 4 – 299.

For examples of changing the value of a table's storage parameters, see the STORAGE clause on page 4 – 449.

Related Topics

CREATE TABLE command on 4 – 245
CONSTRAINT clause on 4 – 149
DISABLE clause on 4 – 295
DROP clause on 4 – 299
ENABLE clause on 4 – 326
STORAGE clause on 4 – 449

ALTER TABLESPACE

Purpose

To alter an existing tablespace in one of the following ways:

- to add datafile(s)
- to rename datafiles
- to change default storage parameters
- to take the tablespace online or offline
- to begin or end a backup
- to allow or disallow writing to a tablespace

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 the following operations:

- to take the tablespace online or offline
- to begin or end a backup
- make the tablespace read–only or read–write

Before you can make a tablespace read–only, the following conditions must be met. It may be easiest to meet these restrictions by performing this function in restricted mode, so that only users with the RESTRICTED SESSION system privilege can be logged on.

- The tablespace must be online.
- There must not be any active transactions in the entire database.

  This is necessary to ensure that there is no undo information that needs to be applied to the tablespace.
- The tablespace must not contain any active rollback segments.

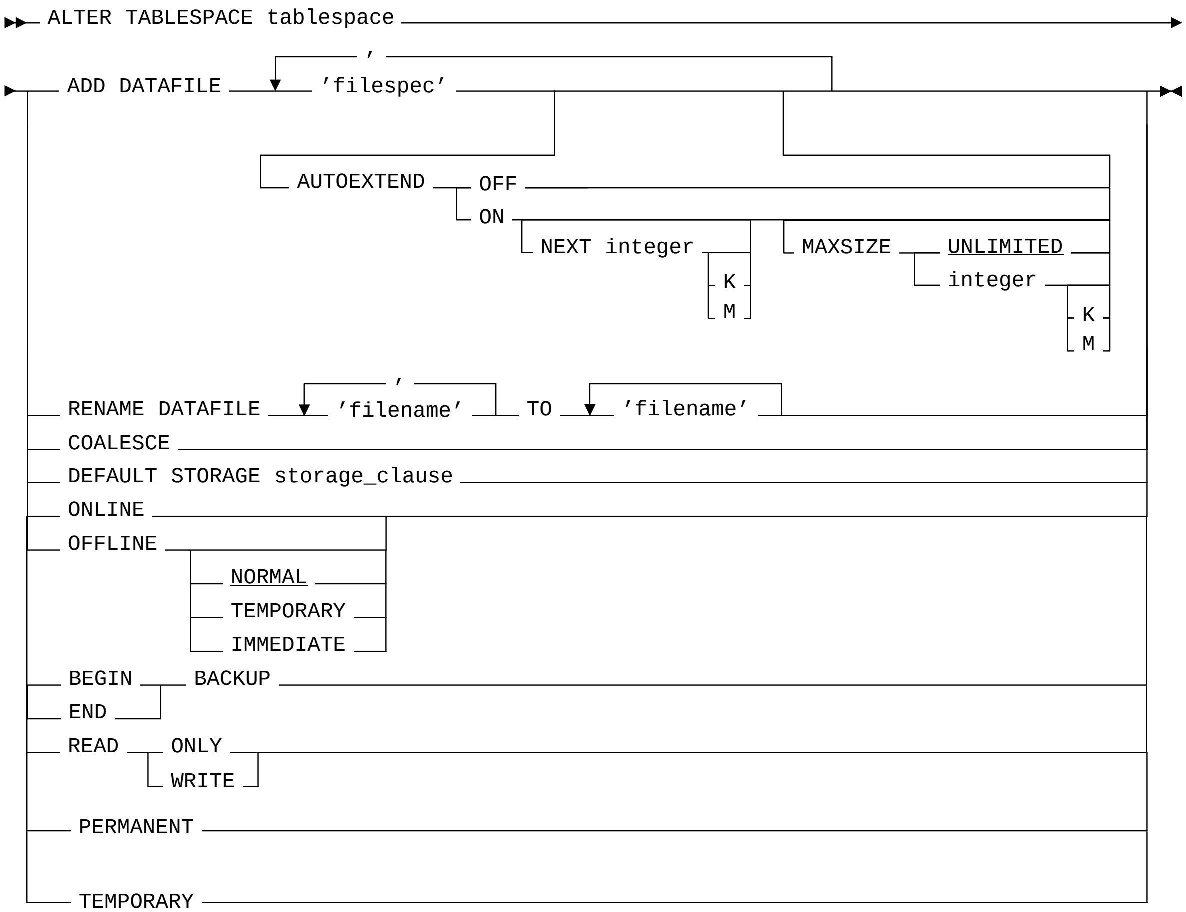
  For this reason, the SYSTEM tablespace can never be made read–only, since it contains the SYSTEM rollback segment. Additionally, because the rollback segments of a read–only tablespace are not accessible, it is recommended that you drop the rollback segments before you make a tablespace read–only.
- The tablespace must not be involved in an online backup, since the end of a backup updates the header file of all datafiles in the tablespace.
- The COMPATIBLE initialization parameter must be set to 7.1.0 or greater.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the tablespace's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

If you are using Trusted Oracle7 in DBMS MAC mode, to add a datafile, your operating system process label must be the equivalent of DBHIGH.

Syntax



Keywords and Parameters

tablespace
is the name of the tablespace to be altered.

ADD DATAFILE
adds the datafile specified by *filespec* to the tablespace. See the syntax description of *filespec*. You can add a datafile while the tablespace is online or offline. Be sure that the datafile is not already in use by another database.

AUTOEXTEND
enables or disables the autoextending of the size of the datafile in the tablespace.

OFF
disable autoextend if it is turned on. NEXT and MAXSIZE are set to zero. Values for NEXT and MAXSIZE must be respecified in further ALTER TABLESPACE AUTOEXTEND commands.

ON
enable autoextend.

NEXT
the size in bytes of the next increment of disk space to be automatically allocated to the datafile when more extents are required. You can also use K or M to specify this size in kilobytes or megabytes. The default is one data block.

MAXSIZE
maximum disk space allowed for automatic extension of the datafile.

UNLIMITED
set no limit on allocating disk space to the datafile.

RENAME DATAFILE

renames one or more of the tablespace's datafiles. Take the tablespace offline before renaming the datafile. Each '*filename*' must fully specify a datafile 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.

COALESCE

for each datafile in the tablespace, coalesce all contiguous free extents into larger contiguous extents.

COALESCE cannot be specified with any other command option.

DEFAULT STORAGE

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

ONLINE

brings the tablespace online.

OFFLINE

takes the tablespace offline and prevents further access to its segments.

NORMAL

performs a checkpoint for all datafiles in the tablespace. All of these datafiles 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 datafiles 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.

Suggestion: Before taking a tablespace offline for a long 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.

BEGIN BACKUP signifies that an online backup is to be performed on the datafiles that comprise this tablespace. This option does not prevent users from accessing the tablespace. You must use this option before beginning an online backup. You cannot use this option on a read–only tablespace.

While the backup is in progress, you cannot:

- 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. You cannot use this option on a read–only tablespace.

READ ONLY signifies that no further write operations are allowed on the tablespace.

READ WRITE signifies that write operations are allowed on a previously read only tablespace.

PERMANENT specifies that the tablespace is to be converted from a temporary to a permanent one. A permanent tablespace is one wherein permanent database objects can be stored. This is the default when a tablespace is created.

TEMPORARY	specifies that the tablespace is to be converted from a permanent to a temporary one. A temporary tablespace is one wherein no permanent database objects can be stored.

Usage Notes

If you are using Trusted Oracle7, datafiles that you add to a tablespace are labelled with the operating system equivalent of DBHIGH.

Before taking a tablespace offline for a long 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.

Once a tablespace is read–only, you can copy its files to read–only media. You must then rename the datafiles in the control file to point to the new location by using the SQL command ALTER DATABASE RENAME.

If you forget to indicate the end of an online tablespace backup, and an instance failure or SHUTDOWN ABORT occurs, Oracle assumes that media recovery (possibly requiring archived redo log) is necessary at the next instance start up. To restart the database without media recovery, see *Oracle7 Server Administrator's Guide*.

Example I

The following statement signals to the database that a backup is about to begin:

```
ALTER TABLESPACE accounting
    BEGIN BACKUP
```

Example II

The following statement signals to the database that the backup is finished:

```
ALTER TABLESPACE accounting
    END BACKUP
```

Example III

This example moves and renames a datafile associated with the ACCOUNTING tablespace from 'DISKA:PAY1.DAT' to 'DISKB:RECEIVE1.DAT':

1. Take the tablespace offline using an ALTER TABLESPACE statement with the OFFLINE option:

   ```
   ALTER TABLESPACE accounting OFFLINE NORMAL
   ```

2. Copy the file from 'DISKA:PAY1.DAT' to 'DISKB:RECEIVE1.DAT' using your operating system's commands.

3. Rename the datafile using the ALTER TABLESPACE command with the RENAME DATAFILE clause:

```
ALTER TABLESPACE accounting
    RENAME  DATAFILE ’diska:pay1.dbf’
    TO      ’diskb:receive1.dbf’
```

4. Bring the tablespace back online using an ALTER TABLESPACE statement with the ONLINE option:

```
ALTER TABLESPACE accounting ONLINE
```

Example IV The following statement adds a datafile to the tablespace; when more space is needed new extents of size 10 kilobytes will be added up to a maximum of 100 kilobytes:

```
ALTER TABLESPACE accounting
    ADD DATAFILE ’disk3:pay3.dbf’
    AUTOEXTEND ON
    NEXT 10 K
    MAXSIZE 100 K
```

Related Topics

CREATE TABLESPACE command on 4 – 254
CREATE DATABASE command on 4 – 178
DROP TABLESPACE command on 4 – 320
STORAGE clause on 4 – 449

ALTER TRIGGER

Purpose

To enable, disable, or compile a database trigger:

Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the trigger's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>-- ALTER TRIGGER ----------------------- trigger ---- ENABLE -------><
                      |__ schema. __|               |-- DISABLE --|
                                                    |__ COMPILE __|
```

Keywords and Parameters

schema	is the schema containing the trigger. If you omit *schema*, Oracle7 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.
COMPILE	compiles the trigger.

Usage Notes

You can use the ALTER TRIGGER command to explicitly recompile a trigger that is invalid. Explicit recompilation eliminates the need for implicit runtime recompilation and prevents associated runtime compilation errors and performance overhead.

When you issue an ALTER TRIGGER statement, Oracle7 recompiles the trigger regardless of whether it is valid or invalid.

When you recompile a trigger, Oracle7 first recompiles objects upon which the trigger depends, if any of these objects are invalid. If Oracle7 recompiles the trigger successfully, the trigger becomes valid. If recompiling the trigger results in compilation errors, then Oracle7 returns an error and the trigger remains invalid. You can then debug triggers using the predefined package DBMS_OUTPUT. For information on debugging procedures, see the "Using Procedures and Packages" chapter of the *Oracle7 Server Application Developer's Guide.* For information on how Oracle7 maintains dependencies among schema objects, including remote objects, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

> **Note:** This command does not change the declaration or definition of an existing trigger. To redeclare or redefine a trigger, you must use the CREATE TRIGGER command with the OR REPLACE option.

Enabling and Disabling Triggers

A database trigger is always in one of the following states:

enabled — If a trigger is enabled, Oracle7 fires the trigger when a triggering statement is issued.

disabled — If the trigger is disabled, Oracle7 does not fire the trigger when a triggering statement is issued.

When you create a trigger, Oracle7 enables it automatically. You can use the ENABLE and DISABLE options of the ALTER TRIGGER command to enable and disable a trigger.

You can also use the ENABLE and DISABLE clauses of the ALTER TABLE command to enable and disable all triggers associated with a table.

> **Note:** The ALTER TRIGGER command does not change the definition of an existing trigger. To redefine a trigger, you must use the CREATE TRIGGER command with the OR REPLACE option.

Example

Consider a trigger named REORDER created on the INVENTORY table that is fired whenever an UPDATE statement reduces the number of a particular part on hand below the part's reorder point. The trigger inserts into a table of pending orders a row that contains the part number, a reorder quantity, and the current date.

When this trigger is created, Oracle7 enables it automatically. You can subsequently disable the trigger with the following statement:

```
ALTER TRIGGER reorder
   DISABLE
```

When the trigger is disabled, Oracle7 does not fire the trigger when an UPDATE statement causes the part's inventory to fall below its reorder point.

After disabling the trigger, you can subsequently enable it with the following statement:

```
ALTER TRIGGER reorder
   ENABLE
```

After you reenable the trigger, Oracle7 fires the trigger whenever a part's inventory falls below its reorder point as a result of an UPDATE statement. Note that a part's inventory may have fallen below its reorder point while the trigger was disabled. When you reenable the trigger, Oracle7 does not automatically fire the trigger for this part.

Related Topics

CREATE TRIGGER command on 4 – 257
DROP TRIGGER command on 4 – 322
DISABLE clause on 4 – 295
ENABLE clause on 4 – 326

ALTER USER

Purpose

To change any of the following 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

Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the user's creation label or you must satisfy one of the following criteria:

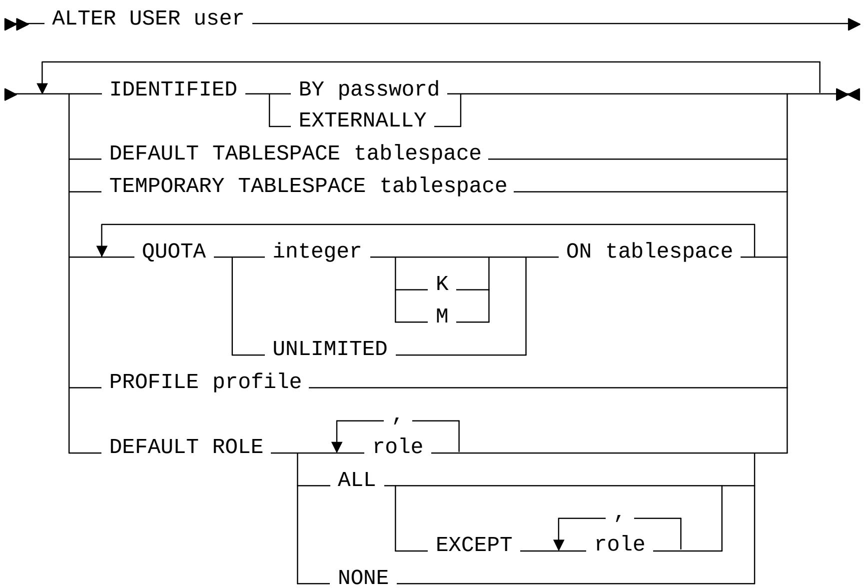
- If the user's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
- If the user's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
- If the user's creation label and your DBMS label are not, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

You can only change a user's default roles if your DBMS label matches the creation label of the user. Your DBMS label must also dominate the role's creation label or you must have READUP system privilege.

You can only establish a default or temporary tablespace if both your DBMS label and the user's creation label dominates the tablespace's creation label or if both you and the user have READUP system privilege.

You can only change a user's profile if both your DBMS label and the user's creation label dominate the profile's creation label or if both you and the user have READUP system privilege.

Syntax



Keywords and Parameters

user is the user to be altered.

IDENTIFIED indicates how Oracle7 permits user access.

BY specifies a new password for the user. The *password* is not usually quoted and must also follow the rules described in the section "Object Naming Rules" on page 2 – 3. A password can only contain single–byte characters from your database character set regardless of whether your character set also contains multi–byte characters.

EXTERNALLY indicates that Oracle7 verifies user access with the operating system, rather than with a password. See the CREATE USER command on page 4 – 267.

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 on page 4 – 267.

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. Oracle7 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.

Establishing Default Roles

The DEFAULT ROLE clause can only contain roles that have been granted directly to the user with a GRANT statement. You cannot use the DEFAULTROLE clause to enable:

- roles not granted to the user
- roles granted through other roles
- roles managed by the operating system

Note that Oracle7 enables default roles at logon without requiring the user to specify their passwords.

Example I

The following statement changes the user SCOTT's password to LION and default tablespace to the tablespace TSTEST:

```
ALTER USER scott
    IDENTIFIED BY lion
    DEFAULT TABLESPACE tstest
```

Example II

The following statement assigns the CLERK profile to SCOTT:

```
ALTER USER scott
PROFILE clerk
```

In subsequent sessions, SCOTT is restricted by limits in the CLERK profile.

Example III

The following statement makes all roles granted directly to SCOTT default roles, except the AGENT role:

```
ALTER USER scott
DEFAULT ROLE ALL EXCEPT agent
```

At the beginning of SCOTT's next session, Oracle7 enables all roles granted directly to SCOTT except the AGENT role.

Related Topics

CREATE PROFILE command on 4 – 210
CREATE ROLE command on 4 – 215
CREATE USER command on 4 – 267
CREATE TABLESPACE command on 4 – 254

ALTER VIEW

Purpose

To recompile a view.

Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the view's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>__ ALTER VIEW ____________________ view __ COMPILE _______________><
                  |__ schema. __|
```

Keywords and Parameters

schema — is the schema containing the view. If you omit *schema*, Oracle7 assumes the view is in your own schema.

view — is the name of the view to be recompiled.

COMPILE — causes Oracle7 to recompile the view. The COMPILE keyword is required.

Usage Notes

You can use the ALTER VIEW command to explicitly recompile a view that is invalid. Explicit recompilation allows you to locate recompilation errors before runtime. You may want to explicitly recompile a view after altering one of its base tables to ensure that the alteration does not affect the view or other objects that depend on it.

When you issue an ALTER VIEW statement, Oracle7 recompiles the view regardless of whether it is valid or invalid. Oracle7 also invalidates any local objects that depend on the view. For more information, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

> **Note:** This command does not change the definition of an existing view. To redefine a view, you must use the CREATE VIEW command with the OR REPLACE option.

Example

To recompile the view CUSTOMER_VIEW, issue the following statement:

```
ALTER VIEW customer_view
    COMPILE
```

If Oracle7 encounters no compilation errors while recompiling CUSTOMER_VIEW, CUSTOMER_VIEW becomes valid. If recompiling results in compilation errors, Oracle7 returns an error and CUSTOMER_VIEW remains invalid.

Oracle7 also invalidates all dependent objects. These objects include any procedures, functions, package bodies, and views that reference CUSTOMER_VIEW. If you subsequently reference one of these objects without first explicitly recompiling it, Oracle7 recompiles it implicitly at runtime.

Related Topics

CREATE VIEW command on 4 – 271

ANALYZE

Purpose

To perform one of the following 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

Prerequisites

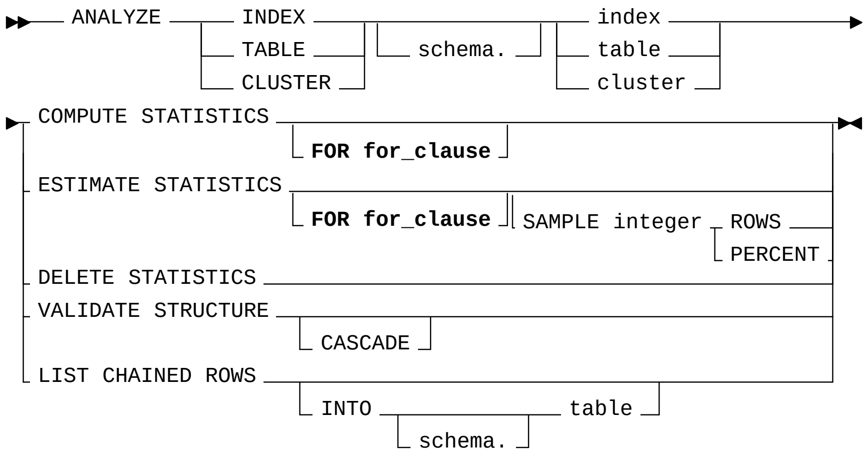
The object to be analyzed must be in your own schema or you must have the ANALYZE ANY system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the creation label of the object to be analyzed or you must satisfy one of the following criteria:

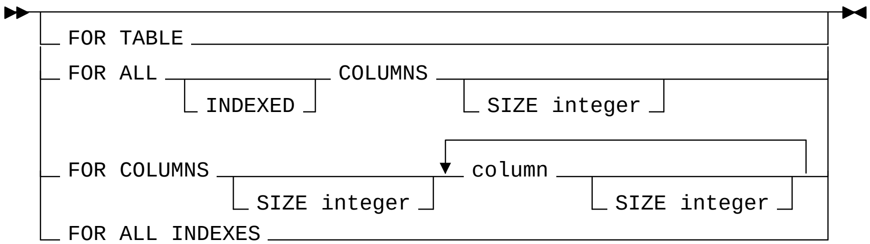
- If the object's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges
- If the object's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
- If the object's creation label and your DBMS label are not comparable, you must 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 Oracle7 in DBMS MAC mode, the list table must also meet the criteria for the analyzed object described above.

Syntax



for_clause ::=



Keywords and Parameters

INDEX
: identifies an index to be analyzed (if no FOR clause is used). If you omit *schema*, Oracle7 assumes the index is in your own schema.

TABLE
: identifies a table to be analyzed. If you omit *schema*, Oracle7 assumes the table is in your own schema. When you collect statistics for a table, Oracle7 also automatically collects the statistics for each of the table's indexes, provided that no FOR clauses are used.

CLUSTER
: identifies a cluster to be analyzed. If you omit *schema*, Oracle7 assumes the cluster is in your own schema. When you collect statistics for a cluster, Oracle7 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 Oracle7 samples to estimate statistics. If you omit this parameter, Oracle7 samples 1064 rows. If you specify more than half of the data, Oracle7 reads all the data and computes the statistics.

ROWS

causes Oracle7 to sample *integer* rows of the table or cluster or *integer* entries from the index. The *integer* must be at least 1.

PERCENT

causes Oracle7 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.

Histogram statistics are described in *Oracle7 Server Tuning*. The following clauses only apply to the ANALYZE TABLE version of this command:

FOR TABLE

collect table statistics for the table.

FOR ALL COLUMNS

collect column statistics for all columns in the table.

FOR ALL INDEXED COLUMNS

collect column statistics for all indexed columns in the table.

FOR COLUMNS
collect column statistics for the specified columns.

FOR ALL INDEXES
all indexes associated with the table will be analyzed.

SIZE
specifies the maximum number of partitions in the histogram. The default value is 75, minimum value is 1, and maximum value is 254.

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, Oracle7 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, Oracle7 also validates the table's indexes. If you use this option when validating a cluster, Oracle7 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 Oracle7 lists the migrated and chained rows. If you omit *schema*, Oracle7 assumes the list table is in your own schema. If you omit this clause altogether, Oracle7 assumes that the table is named CHAINED_ROWS. The list table must be on your local database.

Collecting Statistics

You can collect statistics about the physical storage characteristics and data distribution of an index, table, column, or cluster and store them in the data dictionary. For computing or estimating statistics

- Computation always provides exact values, but can take longer than estimation.
- Estimation is often much faster than computation and the results are usually nearly exact.

Use estimation, rather than computation, unless you feel you need exact values. Some statistics are always computed exactly, regardless of whether you specify computation or estimation. If you choose estimation and the time saved by estimating a statistic is negligible, Oracle7 computes the statistic exactly.

If the data dictionary already contains statistics for the analyzed object, Oracle7 updates the existing statistics with the new ones.

The statistics are used by the Oracle7 optimizer to choose the execution plan for SQL statements that access analyzed objects. These statistics may also be useful to application developers who write such statements. For information on how these statistics are used, see *Oracle7 Server Tuning*.

The following sections list the statistics for indexes, tables, columns, and clusters.

Indexes

For an index, Oracle7 collects the following statistics:

- depth of the index from its root block to its leaf blocks*
- number of leaf blocks
- number of distinct index values
- average number of leaf blocks per index value
- average number of data blocks per index value (for an index on a table)
- clustering factor (how well ordered are the rows about the indexed values)

The statistics marked with asterisks (*) are always computed exactly.

Index statistics appear in the data dictionary views USER_INDEXES, ALL_INDEXES, and DBA_INDEXES.

Tables

For a table, Oracle7 collects the following statistics:

- number of rows
- number of data blocks currently containing data *
- number of data blocks allocated to the table that have never been used *
- average available free space in each data block in bytes
- number of chained rows
- average row length, including the row's overhead, in bytes

The statistics marked with asterisks (*) are always computed exactly.

Table statistics appear in the data dictionary views USER_TABLES, ALL_TABLES, and DBA_TABLES.

Columns

Column statistics can be based on the entire column or can use a histogram. A *histogram* partitions the values in the column into bands, so that all column values in a band fall within the same range In some cases, it is useful to see how many values fall in various ranges. Oracle's histograms are height balanced as opposed to width balanced. This means that the column values are divided into bands so that each band contains approximately the same number of values. The useful information the histogram provides, then, is where in the range of values the endpoints fall. Width–balanced histograms, on the other hand, divide the data into a number of ranges, all of which are the same size, and then count the number of values falling into each range.

The *size* parameter specifies how many bands the column should be divided into. A size of 1 treats the entire column as a single band, which is equivalent to not using histograms at all.

The column statistics that Oracle7 collects are the following:

- number of distinct values in the column as a whole
- maximum and minimum values in each band

When to use Histograms

For uniformly distributed data, the cost–based approach makes fairly accurate guesses at the cost of executing a particular statement. For non–uniformly distributed data, Oracle allows you to store histograms describing the data distribution of a particular column. These histograms are stored in the dictionary and can be used by the cost–based optimizer.

Since they are persistent objects, there is a maintenance and space cost for using histograms. You should only compute histograms for columns that you know have highly–skewed data distribution. Also, be aware that histograms, as well as all optimizer statistics, are static. If the data distribution of a column changes frequently, you must reissue the ANALYZE command to recompute the histogram for that column.

Histograms are not useful for columns with the following characteristics:

- all predicates on the column use bind variables
- the column data is uniformly distributed
- the column is not used in WHERE clauses of queries
- the column is unique and is used only with equality predicates

Create histograms on columns that are frequently used in WHERE clauses of queries and have a highly–skewed data distribution. You create a histogram by using the ANALYZE TABLE option of this command. For example, if you want to create a 10–band histogram on the SAL column of the EMP table, issue the following statement:

```
ANALYZE TABLE emp
   COMPUTE STATISTICS FOR COLUMNS sal SIZE 10;
```

Column statistics appear in the data dictionary views USER_TAB_COLUMNS, ALL_TAB_COLUMNS, and DBA_TAB_COLUMNS. Histograms appear in the data dictionary views USER_HISTOGRAMS, DBA_HISTOGRAMS, and ALL_HISTOGRAMS.

Clusters

For an indexed cluster, Oracle7 collects the average number of data blocks taken up by a single cluster key value and all of its rows. For a hash clusters, Oracle7 collects the average number of data blocks taken up by a single hash key value and all of its rows. These statistics appear in the data dictionary views USER_CLUSTERS and DBA_CLUSTERS.

Example I

The following statement estimates statistics for the CUST_HISTORY table and all of its indexes:

```
ANALYZE TABLE cust_history
   ESTIMATE STATISTICS
```

Deleting Statistics

With the DELETE STATISTICS option of the ANALYZE command, you can remove existing statistics about an object from the data dictionary. You may want to remove statistics if you no longer want the Oracle7 optimizer to use them.

When you use the DELETE STATISTICS option on a table, Oracle7 also automatically removes statistics for all the table's indexes. When you use the DELETE STATISTICS option on a cluster, Oracle7 also automatically removes statistics for all the cluster's tables and all their indexes, including the cluster index.

Example II The following statement deletes statistics about the CUST_HISTORY table and all its indexes from the data dictionary:

```
ANALYZE TABLE cust_history
   DELETE STATISTICS
```

Validating Structures

With the VALIDATE STRUCTURE option of the ANALYZE command, you can verify the integrity of the structure of an index, table, or cluster. If Oracle7 successfully validates the structure, a message confirming its validation is returned to you. If Oracle7 encounters corruption in the structure of the object, an error message is returned to you. In this case, drop and recreate the object.

Since the validating the structure of a object prevents SELECT, INSERT, UPDATE, and DELETE statements from concurrently accessing the object, do not use this option on the tables, clusters, and indexes of your production applications during periods of high database activity.

Indexes

For an index, the VALIDATE STRUCTURE option verifies the integrity of each data block in the index and checks for block corruption. Note that this option does not confirm that each row in the table has an index entry or that each index entry points to a row in the table. You can perform these operations by validating the structure of the table.

When you use the VALIDATE STRUCTURE option on an index, Oracle7 also collects statistics about the index and stores them in the data dictionary view INDEX_STATS. Oracle7 overwrites any existing statistics about previously validated indexes. At any time, INDEX_STATS can contain only one row describing only one index. The INDEX_STATS view is described in the *Oracle7 Server Reference*.

The statistics collected by this option are not used by the Oracle7 optimizer. Do not confuse these statistics with the statistics collected by the COMPUTE STATISTICS and ESTIMATE STATISTICS options.

Example III The following statement validates the structure of the index PARTS_INDEX:

```
ANALYZE INDEX parts_index
    VALIDATE STRUCTURE
```

Tables

For a table, the VALIDATE STRUCTURE option verifies the integrity of each of the table's data blocks and rows. You can use the CASCADE option to also validate the structure of all indexes on the table and to perform cross–referencing between the table and each of its indexes. For each index, the cross–referencing involves the following validations:

- Each value of the tables' indexed column must match the indexed column value of an index entry. The matching index entry must also identify the row in the table by the correct ROWID.
- Each entry in the index identifies a row in the table. The indexed column value in the index entry must match that of the identified row.

Example IV The following statement analyzes the EMP table and all of its indexes:

```
ANALYZE TABLE emp
    VALIDATE STRUCTURE CASCADE
```

Clusters

For a cluster, the VALIDATE STRUCTURE option verifies the integrity of each row in the cluster and automatically validates the structure of each of the cluster's tables. You can use the CASCADE option to also validate the structure of all indexes on the cluster's tables, including the cluster index.

Example V The following statement analyzes the ORDER_CUSTS cluster, all of its tables, and all of their indexes, including the cluster index:

```
ANALYZE CLUSTER order_custs
    VALIDATE STRUCTURE CASCADE
```

Listing Chained Rows

With the LIST option of the ANALYZE command, you can collect information about the migrated and chained rows in a table or cluster. A *migrated row* is one that has been moved from one data block to another. For example, Oracle7 migrates a row in a cluster if its cluster key value is updated. A *chained row* is one that is contained in more than one data block. For example, Oracle7 chains a row of a table or cluster if the row is too long to fit in a single data block. Migrated and chained rows may cause excessive I/O. You may want to identify such rows to eliminate them. For information on eliminating migrated and chained rows, see *Oracle7 Server Tuning*.

You can use the INTO clause to specify an output table into which Oracle7 places this information. The definition of a sample output table CHAINED_ROWS is provided in a SQL script available on your distribution media. Your list table must have the same column names, types, and sizes as the CHAINED_ROWS table. On many operating systems, the name of this script is UTLCHAIN.SQL. The actual name and location of this script may vary depending on your operating system.

Example VI The following statement collects information about all the chained rows of the table ORDER_HIST:

```
ANALYZE TABLE order_hist
    LIST CHAINED ROWS INTO cr
```

The preceding statement places the information into the table CR.

You can then examine the rows with this query:

```
SELECT *
    FROM cr

OWNER_NAME  TABLE_NAME  CLUSTER_NAME  HEAD_ROWID         TIMESTAMP
----------  ----------  ------------  ------------------ ---------
SCOTT       ORDER_HIST                0000346A.000C.0003 15-MAR-93
```

Related Topics

Oracle7 Server Tuning

ARCHIVE LOG clause

Purpose

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

Prerequisites

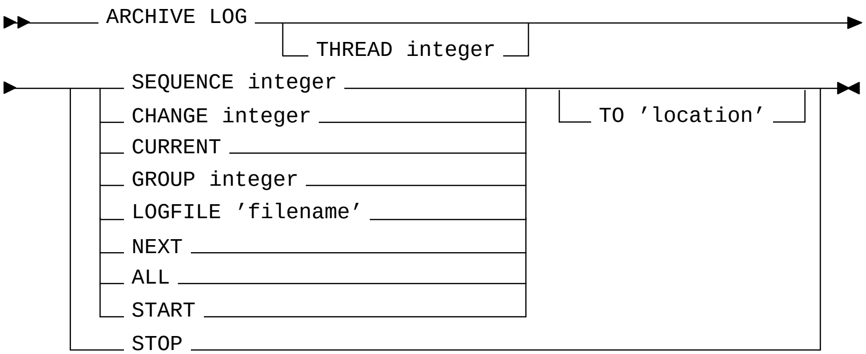
The ARCHIVE LOG clause must appear in an ALTER SYSTEM command. You must have the privileges necessary to issue this statement. For information on these privileges, see the ALTER SYSTEM command on page 4 – 76.

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 Oracle7 in DBMS MAC mode, your DBMS label must be the equivalent of DBHIGH.

Syntax



Keywords and Parameters

THREAD

specifies thread containing the redo log file group to be archived. You only need to specify this parameter if you are using Oracle7 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, Oracle7 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, Oracle7 performs a log switch. If you omit the THREAD parameter, Oracle7 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, Oracle7 archives all redo log file groups from all enabled threads, including logs previous to current logs. 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, Oracle7 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, Oracle7 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, Oracle7 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.

Usage Notes

You must archive redo log file groups in the order in which they are filled. If you specify a redo log file group for archiving with these or LOGFILE parameter and earlier redo log file groups are not yet archived, Oracle7 returns an error. If you specify a redo log file group for archiving with the CHANGE parameter or CURRENT option and earlier redo log file groups are not yet archived, Oracle7 archives all unarchived groups up to and including the specified group.

You can also manually archive redo log file groups with the ARCHIVE LOG Server Manager command. For information on this command, see the *Oracle Server Manager User's Guide.*

You can also choose to have Oracle7 archive redo log files groups automatically. For information on automatic archiving, see the "Archiving Redo Information" chapter of the *Oracle7 Server Administrator's Guide.* Note that you can always manually archive redo log file groups regardless of whether automatic archiving is enabled.

Example I The following statement manually archives the redo log file group with the log sequence number 4 in thread number 3:

```
ALTER SYSTEM ARCHIVE LOG THREAD 3 SEQ 4
```

Example II The following statement manually archives the redo log file group containing the redo log entry with the SCN 9356083:

```
ALTER SYSTEM ARCHIVE LOG CHANGE 9356083
```

Example III The following statement manually archives the redo log file group containing a member named 'DISKL:LOG6.LOG' to an archived redo log file in the location 'DISKA:[ARCH$]':

```
ALTER SYSTEM ARCHIVE LOG
    LOGFILE 'diskl:log6.log'
    TO 'diska:[arch$]'
```

Related Topics

ALTER SYSTEM command on 4 – 76

AUDIT (SQL Statements)

Purpose

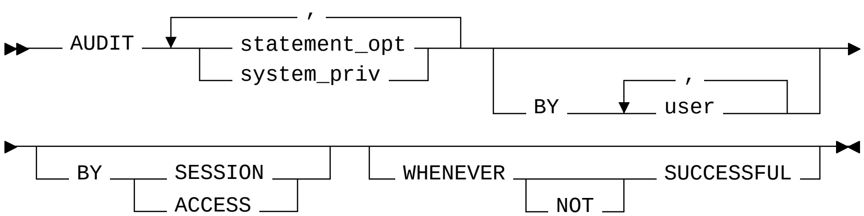
To choose specific SQL statements for auditing in subsequent user sessions. To choose particular schema objects for auditing, use the AUDIT command (Schema Objects).

Prerequisites

You must have AUDIT SYSTEM system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must dominate the creation label of the users whose SQL statements you are auditing.

Syntax



Keywords and Parameters

statement_opt chooses specific SQL statements for auditing. For a list of these statement options and the SQL statements they audit, see Table 4 – 7 on page 4 – 130 and Table 4 – 8 on page 4 – 132.

system_priv chooses SQL statements that are authorized by the specified system privilege for auditing. For a list of all system privileges and the SQL statements that they authorize, see Table 4 – 11 on page 4 – 351.

BY *user* chooses only SQL statements issued by specified users for auditing. If you omit this clause, Oracle7 audits all users' statements.

BY SESSION causes Oracle7 to write a single record for all SQL statements of the same type issued in the same session.

BY ACCESS causes Oracle7 to write one record for each audited statement.

If you specify statement options or system privileges that audit Data Definition Language statements, Oracle7 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, Oracle7 audits SQL statements regardless of success or failure.

Auditing

Auditing keeps track of operations performed by database users. For each audited operation, Oracle7 produces an audit record containing this information:

- user performing the operation
- type of operation
- object involved in the operation
- date and time of the operation

Oracle7 writes audit records to the audit trail. The audit trail is a database table that contains audit records. You can review database activity by examining the audit trail through data dictionary views. For information on these views, see the "Data Dictionary" chapter of *Oracle7 Server Reference*.

How to Audit

To generate audit records, you must perform the following steps:

Enable auditing: You must enable auditing with the initialization parameter AUDIT_TRAIL.

Specify auditing options: To specify auditing options, you must use the AUDIT command. Auditing options choose which SQL commands, operations, database objects, and users Oracle7 audits. After you specify auditing options, they appear in the data dictionary. For more information on data dictionary views containing auditing options see the "Data Dictionary" chapter of *Oracle7 Server Reference*.

You can specify auditing options regardless of whether auditing is enabled. However, Oracle7 does not generate audit records until you enable auditing.

Auditing options specified by the AUDIT command (SQL Statements) apply only to subsequent sessions, rather than to current sessions.

Statement Options

Table 4 – 7 lists the statement options and the statements that they audit.

Statement Option	SQL Statements and Operations
CLUSTER	CREATE CLUSTER AUDIT CLUSTER DROP CLUSTER TRUNCATE CLUSTER
DATABASE LINK	CREATE DATABASE LINK DROP DATABASE LINK
EXISTS	All SQL statements that fail because an object, part of an object, or values already exists in the database. This option is only available with Trusted Oracle.
INDEX	CREATE INDEX ALTER INDEX DROP INDEX
NOT EXISTS	All SQL statements that fail because a specified object does not exist.
PROCEDURE	CREATE FUNCTION CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP PACKAGE DROP PROCEDURE
PROFILE	CREATE PROFILE ALTER PROFILE DROP PROFILE
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM DROP PUBLIC SYNONYM
ROLE	CREATE ROLE ALTER ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	CREATE ROLLBACK SEGMENT ALTER ROLLBACK SEGMENT DROP ROLLBACK SEGMENT

Table 4 – 7 Statement Auditing Options

Statement Option	SQL Statements and Operations
SEQUENCE	CREATE SEQUENCE DROP SEQUENCE
SESSION	Logons
SYNONYM	CREATE SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT (SQL Statements) NOAUDIT (SQL Statements)
SYSTEM GRANT	GRANT (System Privileges and Roles) REVOKE (System Privileges and Roles)
TABLE	CREATE TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE
TRIGGER	CREATE TRIGGER ALTER TRIGGER with ENABLE and DISABLE options DROP TRIGGER ALTER TABLE with ENABLE ALL TRIGGERS and DISABLE ALL TRIGGERS clauses
USER	CREATE USER ALTER USER DROP USER
VIEW	CREATE VIEW DROP VIEW

Table 4 – 7 (continued) Statement Auditing Options

Short Cuts for System Privileges and Statement Options

Oracle7 provides short cuts for specifying system privileges and statement options. With these shortcuts, you can specify auditing for multiple system privileges and statement options at once:

CONNECT
: This short cut is equivalent to specifying the CREATE SESSION system privilege.

RESOURCE
: This short cut is equivalent to specifying the following system privileges:

- ALTER SYSTEM
- CREATE CLUSTER
- CREATE DATABASE LINK
- CREATE PROCEDURE
- CREATE ROLLBACK SEGMENT
- CREATE SEQUENCE
- CREATE SYNONYM
- CREATE TABLE
- CREATE TABLESPACE
- CREATE VIEW

DBA
: This short cut is equivalent to the SYSTEM GRANT statement option and the following system privileges:

- AUDIT SYSTEM
- CREATE PUBLIC DATABASE LINK
- CREATE PUBLIC SYNONYM
- CREATE ROLE
- CREATE USER

ALL
: This short cut is equivalent to specifying all statement options shown in Table 4 – 7, but not the additional statement options shown in Table 4 – 8.

ALL PRIVILEGES
: This short cut is equivalent to specifying all system privileges.

Oracle Corporation encourages you to choose individual system privileges and statement options for auditing, rather than these short cuts. These short cuts may not be supported in future versions of Oracle.

Additional Statement Options

Table 4 – 8 lists additional statement options and the SQL statements and operations that they audit. Note that these statement options are not included in the ALL short cut.

Statement Option	SQL Statements and Operations
ALTER SEQUENCE	ALTER SEQUENCE
ALTER TABLE	ALTER TABLE
COMMENT TABLE	COMMENT ON TABLE table, view, snapshot COMMENT ON COLUMN table.column, view.column, snapshot.column
DELETE TABLE	DELETE FROM table, view
EXECUTE PROCEDURE	Execution of any procedure or function or access to any variable or cursor inside a package.
GRANT PROCEDURE	GRANT privilege ON procedure, function, package REVOKE privilege ON procedure, function, package
GRANT SEQUENCE	GRANT privilege ON sequence REVOKE privilege ON sequence
GRANT TABLE	GRANT privilege ON table, view, snapshot. REVOKE privilege ON table, view, snapshot
INSERT TABLE	INSERT INTO table, view
LOCK TABLE	LOCK TABLE table, view
SELECT SEQUENCE	Any statement containing sequence.CURRVAL or sequence.NEXTVAL
SELECT TABLE	SELECT FROM table, view, snapshot
UPDATE TABLE	UPDATE table, view

Table 4 – 8 Additional Statement Auditing Options

Example I To choose auditing for every SQL statement that creates, alters, drops, or sets a role, regardless of whether the statement completes successfully, issue the following statement:

```
AUDIT ROLE
```

To choose auditing for every statement that successfully creates, alters, drops, or sets a role, issue the following statement:

```
AUDIT ROLE
    WHENEVER SUCCESSFUL
```

To choose auditing for every CREATE ROLE, ALTER ROLE, DROP ROLE, or SET ROLE statement that results in an Oracle7 error, issue the following statement:

```
AUDIT ROLE
    WHENEVER NOT SUCCESSFUL
```

Example II To choose auditing for any statement that queries or updates any table, issue the following statement:

```
AUDIT SELECT TABLE, UPDATE TABLE
```

To choose auditing for statements issued by the users SCOTT and BLAKE that query or update a table or view, issue the following statement:

```
AUDIT SELECT TABLE, UPDATE TABLE
    BY scott, blake
```

Example III To choose auditing for statements issued using the DELETE ANY TABLE system privilege, issue the following statement:

```
AUDIT DELETE ANY TABLE
```

Related Topics

AUDIT (Schema Objects) command on 4 – 134
NOAUDIT (SQL Statements) command on 4 – 372

AUDIT (Schema Objects)

Purpose

To choose a specific schema object for auditing. To choose particular SQL commands for auditing, use the AUDIT command (SQL Statements) described in the previous section of this chapter.

Prerequisites

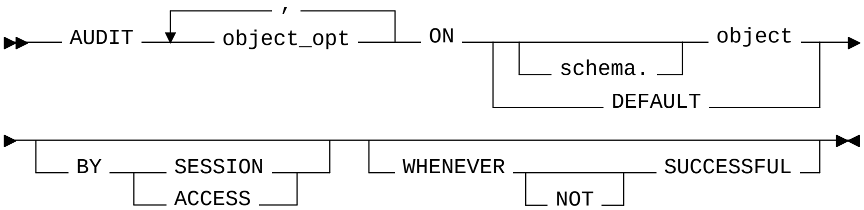
The object you choose for auditing must be in your own schema or you must have AUDIT ANY system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the object's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

object_opt specifies a particular operation for auditing. Table 4 – 9 shows each object option and the types of objects for which it applies.

schema is the schema containing the object chosen for auditing. If you omit *schema*, Oracle7 assumes the object is in your own schema.

object identifies the object chosen for auditing. The object must be one of the following 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.

If you omit both of the following options, Oracle7 audits by session.

BY SESSION means that Oracle7 writes a single record for all operations of the same type on the same object issued in the same session.

BY ACCESS means that Oracle7 writes one record for each audited operation.

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, Oracle7 audits all SQL statements, regardless of success or failure.

Auditing

Auditing keeps track of operations performed by database users. Fora brief conceptual overview of auditing including how to enable auditing, see the AUDIT command (SQL Statements) described on page 4 – 127. Note that auditing options established by the AUDIT command (Schema Objects) apply to current sessions as well as to subsequent sessions.

Object Options

Table 4 – 9 shows the object options you can choose for each type of object.

Object Option	Tables	Views	Sequences	Procedures Functions Packages	Snapshots
ALTER	3		3		3
AUDIT	3	3	3	3	3
COMMENT	3	3			3
DELETE	3	3			3
EXECUTE				3	
GRANT	3	3	3	3	3
INDEX	3				3
INSERT	3	3			3
LOCK	3	3			3
RENAME	3	3		3	3
SELECT	3	3	3		3
UPDATE	3	3			3

Table 4 – 9 Object Auditing Options

The name of each object option specifies a command to be audited. For example, if you choose to audit a table with the ALTER option, Oracle7 audits all ALTER TABLE statements issued against the table. If you choose to audit a sequence with the SELECT option, Oracle7 audits all statements that use any of the sequence's values.

Short Cuts for Object Options

Oracle7 provides a short cut for specifying object auditing options:

ALL This short cut is equivalent to specifying all object options applicable for the type of object. You can use this short cut rather than explicitly specifying all options for an object.

Default Auditing

You can use the DEFAULT option of the AUDIT command to specify auditing options for objects that have not yet been created. Once you have established these default auditing options, any subsequently created object is automatically audited with those options. Note that the default auditing options for a view are always the union of the auditing options for the view's base tables.

If you change the default auditing options, the auditing options for previously–created objects remain the same. You can only change the auditing options for an existing object by specifying the object in the ON clause of the AUDIT command.

Example I

To choose auditing for every SQL statement that queries the EMP table in the schema SCOTT, issue the following statement:

```
AUDIT SELECT
    ON scott.emp
```

To choose auditing for every statement that successfully queries the EMP table in the schema SCOTT, issue the following statement:

```
AUDIT SELECT
    ON scott.emp
    WHENEVER SUCCESSFUL
```

To choose auditing for every statement that queries the EMP table in the schema SCOTT and results in an Oracle7 error, issue the following statement:

```
AUDIT SELECT
    ON scott.emp
    WHENEVER NOT SUCCESSFUL
```

Example II

To choose auditing for every statement that inserts or updates a row in the DEPT table in the schema BLAKE, issue the following statement:

```
AUDIT INSERT, UPDATE
    ON blake.dept
```

Example III

To choose auditing for every statement that performs any operation on the ORDER sequence in the schema ADAMS, issue the following statement:

```
AUDIT ALL
    ON adams.order
```

The above statement uses the ALL short cut to choose auditing for the following statements that operate on the sequence:

- ALTER SEQUENCE
- AUDIT
- GRANT
- any statement that accesses the sequence's values using the pseudocolumns CURRVAL or NEXTVAL

Example IV The following statement specifies default auditing options for objects created in the future:

```
AUDIT ALTER, GRANT, INSERT, UPDATE, DELETE
    ON DEFAULT
```

Any objects created later are automatically audited with the specified options that apply to them, provided that auditing has been enabled:

- If you create a table, Oracle7 automatically audits any ALTER, INSERT, UPDATE, or DELETE statements issued against the table.
- If you create a view, Oracle7 automatically audits any INSERT, UPDATE, or DELETE statements issued against the view.
- If you create a sequence, Oracle7 automatically audits any ALTER statements issued against the sequence.
- If you create a procedure, package, or function, Oracle7 automatically audits any ALTER statements issued against it.

Related Topics

AUDIT (SQL Statements) command on 4 – 127
NOAUDIT (Schema Objects) command on 4 – 374

CLOSE (Embedded SQL)

Purpose

To disable a cursor, freeing the resources acquired by opening the cursor, and releasing parse locks.

Prerequisites

The cursor must be already open.

Syntax

```
►►—— EXEC SQL CLOSE cursor ——————————————————————►◄
```

Keywords and Parameters

cursor is the cursor to be closed. The cursor must currently be open.

Usage Notes

Rows cannot be fetched from a closed cursor. A cursor need not be closed to be reopened. The HOLD_CURSOR and RELEASE_CURSOR precompiler options alter the effect of the CLOSE command. For information on these options, see *Programmer's Guide to the Oracle Precompilers.*

Example

This example illustrates the use of the CLOSE command:

```
EXEC SQL CLOSE emp_cursor
```

Related Topics

PREPARE command on 4 – 381
DECLARE CURSOR command on 4 – 280
OPEN command on 4 – 376

COMMENT

Purpose

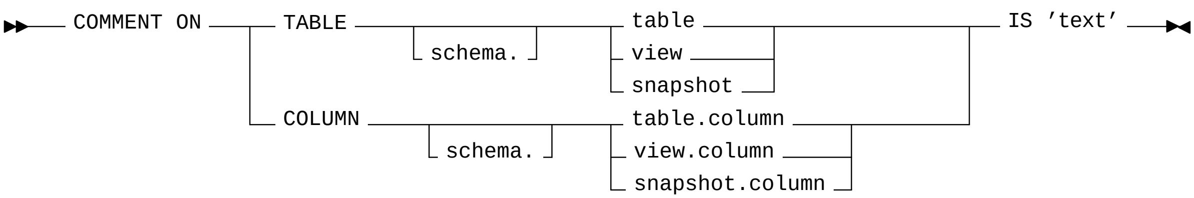
To add a comment about a table, view, snapshot, or column into the data dictionary.

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 Oracle7 in DBMS MAC mode, your DBMS label must match the creation label of the table, view, snapshot, or column.

Syntax



Keywords and Parameters

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*, Oracle7 assumes the table, view, or snapshot is in your own schema.

IS *'text'* is the text of the comment. See the syntax description of *'text'* on page 2 – 15.

Usage Notes

You can effectively drop a comment from the database by setting it to the empty string ''. For information on the data dictionary views that contain comments, see Appendix B "Data Dictionary Reference" of *Oracle7 Server Reference*.

Example

To insert an explanatory remark on the NOTES column of the SHIPPING table, you might issue the following statement:

```
COMMENT ON COLUMN shipping.notes
    IS 'Special packing or shipping instructions'
```

To drop this comment from the database, issue the following statement:

```
COMMENT ON COLUMN shipping.notes IS ''
```

Related Topics

The section "Comments" on page 2 – 43.

COMMIT

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.

Prerequisites

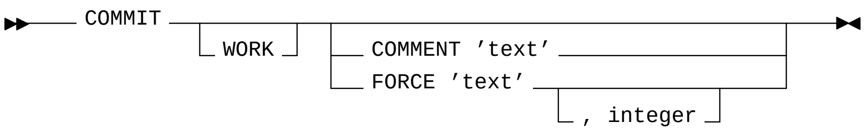
You need no privileges to commit your current transaction.

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.

If you are using Trusted Oracle7 in DBMS MAC mode, you can only commit an in–doubt transaction if your DBMS label matches the label the transaction’s label and the creation label of the user who originally committed the transaction or if you satisfy one of the following criteria:

- If the transaction’s label or the user’s creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
- If the transaction’s label or the user’s creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
- If the transaction’s label or the user’s creation label is not comparable with your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

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 Oracle7 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.

Usage Notes

It is recommended that you explicitly end every transaction in your application programs with a COMMIT or ROLLBACK statement, including the last transaction, before disconnecting from Oracle7. If you do not explicitly commit the transaction and the program terminates abnormally, the last uncommitted transaction is automatically rolled back.

A normal exit from most Oracle7 utilities and tools causes the current transaction to be committed. A normal exit from an Oracle Precompiler program does not commit the transaction and relies on Oracle7 to rollback the current transaction. See the COMMIT command (Embedded SQL) on page 4 – 141.

Transactions

A transaction (or a logical unit of work) is a sequence of SQL statements that Oracle7 treats as a single unit. A transaction begins with the first executable SQL statement after a COMMIT, ROLLBACK or connection to the database. A transaction ends with a COMMIT, ROLLBACK or disconnection (intentional or unintentional) from the database. Note that Oracle7 issues an implicit COMMIT before and after any Data Definition Language statement.

You can also use a COMMIT or ROLLBACK statement to terminate a read only transaction begun by a SET TRANSACTION statement.

Example I This example inserts a row into the DEPT table and commits this change:

```
INSERT INTO dept   VALUES (50, 'MARKETING', 'TAMPA')
COMMIT WORK
```

Example II The following statement commits the current transaction and associates a comment with it:

```
COMMIT WORK
   COMMENT 'In–doubt transaction Code 36, Call (415) 555–2637'
```

If a network or machine failure prevents this distributed transaction from committing properly, Oracle7 stores the comment in the data dictionary along with the transaction ID. The comment indicates the part of the application in which the failure occurred and provides information for contacting the administrator of the database where the transaction was committed.

Distributed Transactions

Oracle7 with the distributed option allows you to perform distributed transactions, or transactions that modify data on multiple databases. To commit a distributed transaction, you need only issue a COMMIT statement as you would to commit any other transaction. Each component of the distributed transaction is then committed on each database.

If a network or machine failure during the commit process for a distributed transaction, the state of the transaction may be unknown, or *in–doubt*. After consultation with the administrators of the other databases involved in the transaction, you may decide to manually commit or roll back the transaction on your local database. You can manually commit the transaction on your local database by using the FORCE clause of the COMMIT command. For more information on these topics, see the "Database Administration" chapter of *Oracle7 Server Distributed Systems, Volume I.*

Note that a COMMIT statement with a FORCE clause only commits the specified transaction. Such a statement does not affect your current transaction.

Example III The following statement manually commits an in–doubt distributed transaction:

```
COMMIT FORCE '22.57.53'
```

Related Topics

COMMIT (Embedded SQL) command on 4 – 141
ROLLBACK command on 4 – 397
SAVEPOINT command on 4 – 404
SET TRANSACTION command on 4 – 445

COMMIT (Embedded SQL)

Purpose

To end your current transaction, making permanent all its changes to the database and optionally freeing all resources and disconnecting from Oracle7.

Prerequisites

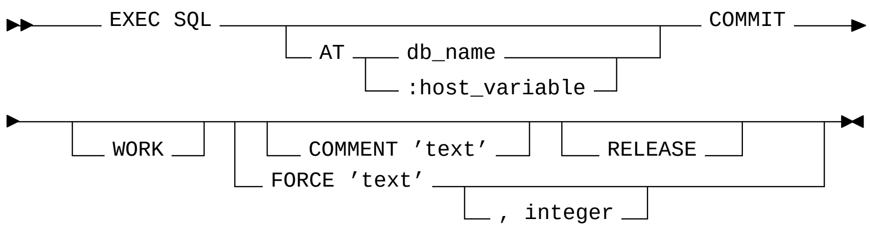
To commit your current transaction, no privileges are necessary.

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.

If you are using Trusted Oracle7 in DBMS MAC mode, you can only commit an in–doubt transaction if your DBMS label matches the label the transaction's label and the creation label of the user who originally committed the transaction or if you satisfy one of the following criteria:

- If the transaction's label or the user's creation label is higher than your DBMS label, you must have READUP and WRITEUP system privileges.
- If the transaction's label or the user's creation label is lower than your DBMS label, you must have WRITEDOWN system privilege.
- If the transaction's label or the user's creation label is not comparable with your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keyword and Parameters

AT
: identifies the database to which the COMMIT statement is issued. The database can be identified by either:

  db_name
  : is a database identifier declared in a previous DECLARE DATABASE statement.

  :*host_variable*
  : is a host variable whose value is a previously declared *db_name*.

  If you omit this clause, Oracle7 issues the statement to your default database.

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 Oracle7 stores in the data dictionary view DBA_2PC_PENDING along with the transaction ID if the transaction becomes in–doubt.

RELEASE
: frees all resources and disconnects you from Oracle7.

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 optional *integer* to explicitly assign the transaction a system change number (SCN). If you omit the *integer*, the transaction is committed using the current SCN.

Usage Notes

Always explicitly commit or rollback the last transaction in your program by using the COMMIT or ROLLBACK command and the RELEASE option. Oracle7 automatically rolls back changes if the program terminates abnormally.

The COMMIT command has no effect on host variables or on the flow of control in the program.

For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example This example illustrates the use of the embedded SQL COMMIT command:

```
EXEC SQL AT sales_db COMMIT RELEASE
```

Related Topics

COMMIT command on 4 – 141
ROLLBACK command on 4 – 397
SAVEPOINT command on 4 – 404
SET TRANSACTION command on 4 – 445

CONNECT (Embedded SQL)

Purpose

To log on to an Oracle7 database.

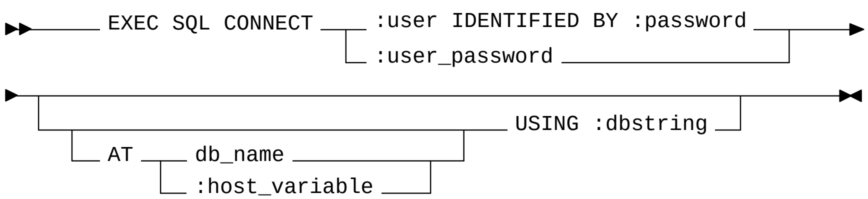
Prerequisites

You must have CREATE SESSION system privilege in the specified database.

If you are using Trusted Oracle7 in DBMS MAC mode, your operating system label must dominate both your creation label and the label at which you were granted CREATE SESSION system privilege. Your operating system label must also fall between the operating system equivalents of DBHIGH and DBLOW, inclusive.

If you are using Trusted Oracle7 in OS MAC mode, your operating system label must match the label of the database to which you are connecting.

Syntax



Keyword and Parameters

:*user*
:*password*
: specifies your username and password separately.

:*user_password*
: is a single host variable containing the Oracle7 username and password separated by a slash (/).

To allow Oracle7 to verify your connection through your operating system, specify a :*user_password* value of '/'.

AT
: identifies the database to which the connection is made. The database can be identified by either:

db_name
: is a database identifier declared in a previous DECLARE DATABASE statement.

:*host_variable*
: is a host variable whose value is a previously declared *db_name.*

USING — specifies the SQL*Net database specification string used to connect to a non–default database. If you omit this clause, you are connected to your default database.

Usage Notes

A program can have multiple connections, but can only connect once to your default database. For more information on this command, the *Programmer's Guide to the Oracle Precompilers.*

Example

The following example illustrate the use of CONNECT:

```
EXEC SQL CONNECT :username
   IDENTIFIED BY :password
```

You can also use this statement in which the value of :*userid* is the value of :*username* and :*password* separated by a "/" such as 'SCOTT/TIGER':

```
EXEC SQL CONNECT :userid
```

Related Topics

COMMIT command on 4 – 141
DECLARE DATABASE command on 4 – 282
ROLLBACK command on 4 – 397

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.

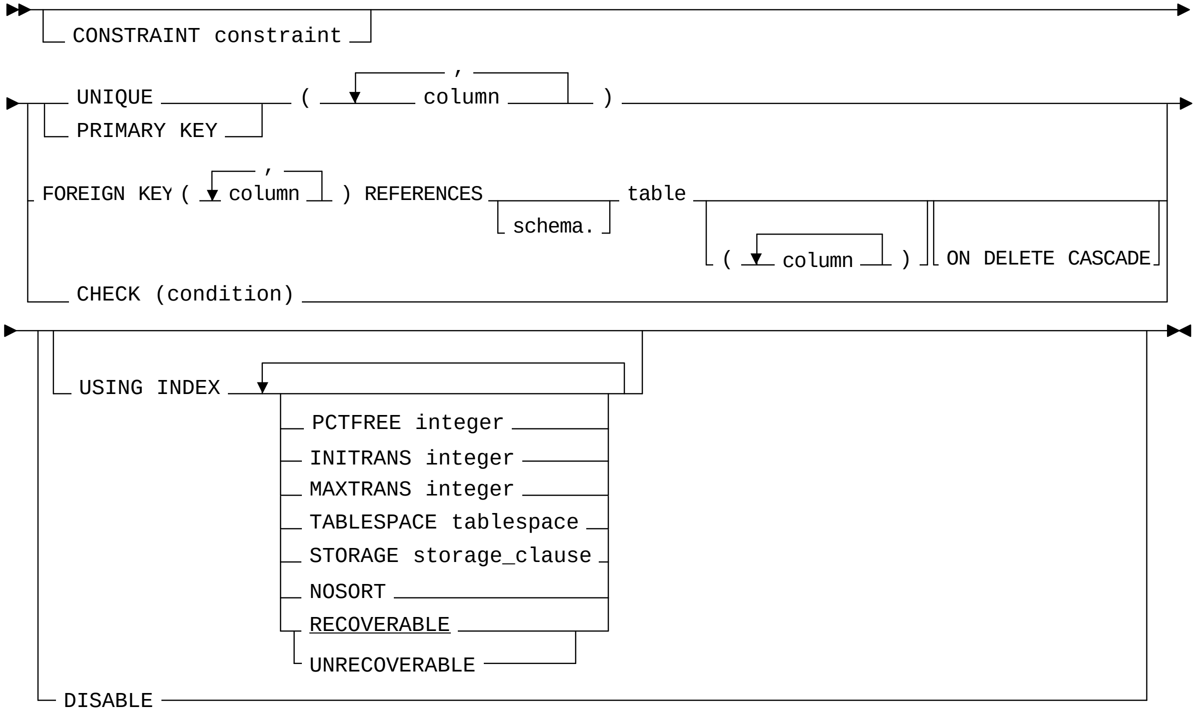
Prerequisites

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 command on page 4 – 245 and the ALTER TABLE command on page 4 – 89.

Defining a constraint may also require additional privileges or preconditions that depend on the type of constraint. For information on these privileges, see the descriptions of each type of integrity constraint beginning on page 4 – 152.

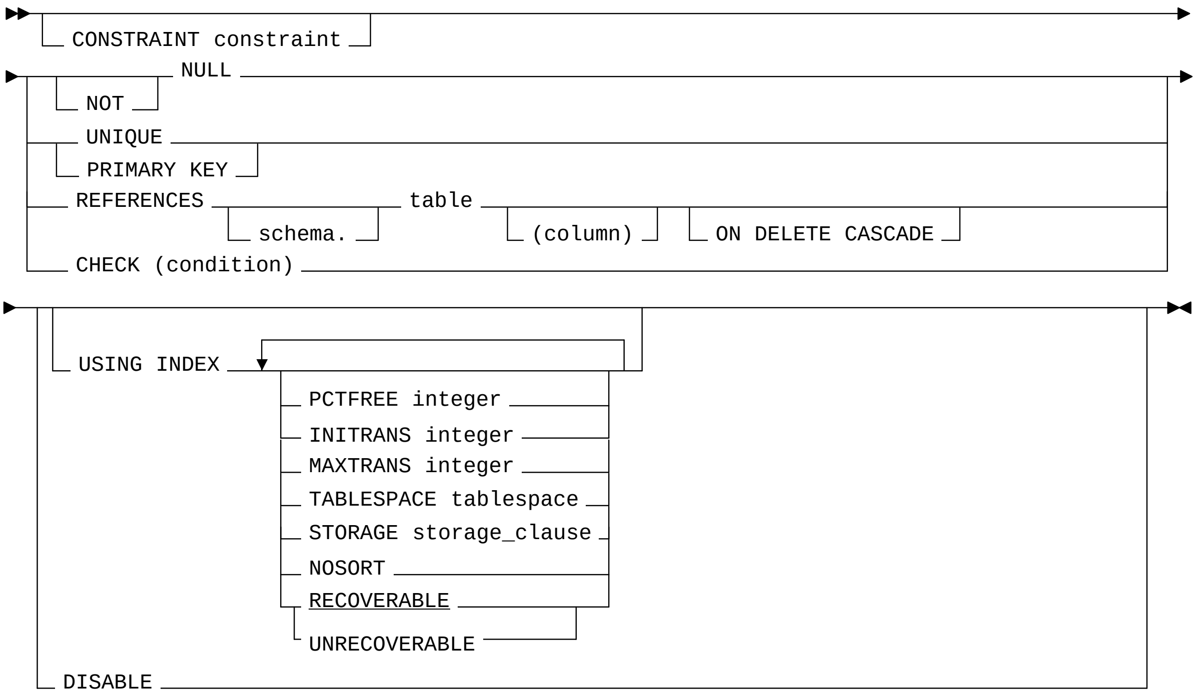
Syntax

table_constraint ::=



Syntax

column_constraint ::=



Keywords and Parameters

CONSTRAINT — identifies the integrity constraint by the name *constraint*. Oracle7 stores this name in the data dictionary along with the definition of the integrity constraint. If you omit this identifier, Oracle7 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. For information on data dictionary views that contain constraints, see the "Data Dictionary Reference" chapter of *Oracle7 Server Reference*.

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 Oracle7 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 Oracle7 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, PCTFREE, RECOVERABLE, and UNRECOVERABLE parameters for the index. For information on these parameters, see the CREATE TABLE command on page 4 – 245.

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

NOSORT — indicates that the rows are stored in the database in ascending order and therefore Oracle7 does not have to sort the rows when creating the index.

DISABLE — disables the integrity constraint. If an integrity constraint is disabled, Oracle7 does not enforce it. If you do not specify this option, Oracle7 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. See the ENABLE clause on page 4 – 326 and DISABLE clause on pages 4 – 295.

Defining Integrity Constraints

To define an integrity constraint, include a CONSTRAINT clause in CREATE TABLE or ALTER TABLE statement. The CONSTRAINT clause has two syntactic forms:

table_constraint
: The *table_constraint* syntax is part of the table definition. An integrity constraint defined with this syntax can impose rules on any columns in the table.

  The *table_constraint* syntax can appear in a CREATE TABLE or ALTER TABLE statement. This syntax can define any type of integrity constraint except a NOT NULL constraint.

column_constraint
: The *column_constraint* syntax is part of a column definition. Usually, an integrity constraint defined with this syntax can only impose rules on the column in which it is defined.

  The *column_constraint* syntax that appears in a CREATE TABLE statement can define any type of integrity constraint. *Column_constraint* syntax that appears in an ALTER TABLE statement can only define or remove a NOT NULL constraint.

The *table_constraint* syntax and the *column_constraint* syntax are simply different syntactic means of defining integrity constraints. A constraint that references more than one column must be defined as a table constraint. There is no other functional difference between an integrity constraint defined with *table_constraint* syntax and the same constraint defined with *column_constraint* syntax.

NOT NULL Constraints

The NOT NULL constraint specifies that a column cannot contain nulls. To satisfy this constraint, every row in the table must contain a value for the column.

The NULL keyword indicates that a column can contain nulls. It does not actually define an integrity constraint. If you do not specify either NOT NULL or NULL, the column can contain nulls by default.

You can only specify NOT NULL or NULL with *column_constraint* syntax in a CREATE TABLE or ALTER TABLE statement, not with *table_constraint* syntax.

Example I

The following statement alters the EMP table and defines and enables a NOT NULL constraint on the SAL column:

```
ALTER TABLE emp
   MODIFY (sal  NUMBER  CONSTRAINT nn_sal NOT NULL)
```

NN_SAL ensures that no employee in the table has a null salary.

UNIQUE Constraints

The UNIQUE constraint designates a column or combination of columns as a unique key. To satisfy a UNIQUE constraint, no two rows in the table can have the same value for the unique key. However, the unique key made up of a single column can contain nulls.

A unique key column cannot be of datatype LONG or LONG RAW. You cannot designate the same column or combination of columns as both a unique key and a primary key or as both a unique key and a cluster key. However, you can designate the same column or combination of columns as both a unique key and a foreign key.

Defining Unique Keys

You can define a unique key on a single column with *column_constraint* syntax.

Example II The following statement creates the DEPT table and defines and enables a unique key on the DNAME column:

```
CREATE TABLE dept
    (deptno  NUMBER(2),
     dname   VARCHAR2(9)  CONSTRAINT unq_dname UNIQUE,
     loc     VARCHAR2(10) )
```

The constraint UNQ_DNAME identifies the DNAME column as a unique key. This constraint ensures that no two departments in the table have the same name. However, the constraint does allow departments without names.

Alternatively, you can define and enable this constraint with the *table_constraint* syntax:

```
CREATE TABLE dept
    (deptno  NUMBER(2),
     dname   VARCHAR2(9),
     loc     VARCHAR2(10),
        CONSTRAINT unq_dname
        UNIQUE (dname)
    USING INDEX PCTFREE 20
        TABLESPACE user_x
        STORAGE (INITIAL 8K  NEXT 6K) )
```

The above statement also uses the USING INDEX option to specify storage characteristics for the index that Oracle7 creates to enforce the constraint.

Defining Composite Unique Keys

A composite unique key is a unique key made up of a combination of columns. Since Oracle7 creates an index on the columns of a unique key, a composite unique key can contain a maximum of 16 columns. To define a composite unique key, you must use *table_constraint* syntax, rather than *column_constraint* syntax.

To satisfy a constraint that designates a composite unique key, no two rows in the table can have the same combination of values in the key columns. Also, any row that contains nulls in all key columns automatically satisfies the constraint. However, two rows that contain nulls for one or more key columns and the same combination of values for the other key columns violate the constraint.

Example III

The following statement defines and enables a composite unique key on the combination of the CITY and STATE columns of the CENSUS table:

```
ALTER TABLE census
    ADD CONSTRAINT unq_city_state
    UNIQUE (city, state)
    USING INDEX PCTFREE 5
        TABLESPACE user_y
    EXCEPTIONS INTO bad_keys_in_ship_cont
```

The UNQ_CITY_STATE constraint ensures that the same combination of CITY and STATE values does not appear in the table more than once.

The CONSTRAINT clause also specifies other properties of the constraint:

- The USING INDEX option specifies storage characteristics for the index Oracle7 creates to enforce the constraint.
- The EXCEPTIONS option causes Oracle7 to write information to the BAD_KEYS_IN_SHIP_CONT table about any rows currently in the SHIP_CONT table that violate the constraint.

PRIMARY KEY Constraints

A PRIMARY KEY constraint designates a column or combination of columns as the table's primary key. To satisfy a PRIMARY KEY constraint, both of the following conditions must be true:

- No primary key value can appear in more than one row in the table.
- No column that is part of the primary key can contain a null.

A table can have only one primary key.

A primary key column cannot be of datatype LONG or LONG RAW. You cannot designate the same column or combination of columns as both a primary key and a unique key or as both a primary key and a cluster key. However, you can designate the same column or combination of columns as both a primary key and a foreign key.

Defining Primary Keys

You can use the *column_constraint* syntax to define a primary key on a single column.

Example IV The following statement creates the DEPT table and defines and enables a primary key on the DEPTNO column:

```
CREATE TABLE dept
    (deptno  NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,
     dname   VARCHAR2(9),
     loc     VARCHAR2(10) )
```

The PK_DEPT constraint identifies the DEPTNO column as the primary key of the DEPTNO table. This constraint ensures that no two departments in the table have the same department number and that no department number is NULL.

Alternatively, you can define and enable this constraint with *table_constraint* syntax:

```
CREATE TABLE dept
    (deptno  NUMBER(2),
     dname   VARCHAR2(9),
     loc   VARCHAR2(10),
     CONSTRAINT pk_dept PRIMARY KEY (deptno) )
```

Defining Composite Primary Keys

A composite primary key is a primary key made up of a combination of columns. Because Oracle7 creates an index on the columns of a primary key, a composite primary key can contain a maximum of 16 columns. To define a composite primary key, you must use the *table_constraint* syntax, rather than the *column_constraint* syntax.

Example V The following statement defines a composite primary key on the combination of the SHIP_NO and CONTAINER_NO columns of the SHIP_CONT table:

```
ALTER TABLE ship_cont
    ADD PRIMARY KEY (ship_no, container_no) DISABLE
```

This constraint identifies the combination of the SHIP_NO and CONTAINER_NO columns as the primary key of the SHIP_CONTAINER. The constraint ensures that no two rows in the table have the same values for both the SHIP_NO column and the CONTAINER_NO column.

The CONSTRAINT clause also specifies the following properties of the constraint:

- Since the constraint definition does not include a constraint name, Oracle7 generates a name for the constraint.
- The DISABLE option causes Oracle7 to define the constraint but not enforce it.

Referential Integrity Constraints

A referential integrity constraint designates a column or combination of columns as a foreign key and establishes a relationship between that foreign key and a specified primary or unique key, called the referenced key. In this relationship, the table containing the foreign key is called the *child table* and the table containing the referenced key is called the *parent table*. Note the following caveats:

- The child and parent tables must be on the same database. They cannot be on different nodes of a distributed database. Oracle7 allows you to enforce referential integrity across nodes of a distributed database with database triggers. For information on how to use database triggers for this purpose, see the “Using Database Triggers” chapter of the *Oracle7 Server Application Developer’s Guide*.
- The foreign key and the referenced key can be in the same table. In this case, the parent and child tables are the same.

To satisfy a referential integrity constraint, each row of the child table must meet one of the following conditions:

- The value of the row’s foreign key must appear as a referenced key value in one of the parent table’s rows. The row in the child table is said to depend on the referenced key in the parent table.
- The value of one of the columns that makes up the foreign key must be null.

A referential integrity constraint is defined in the child table. A referential integrity constraint definition can include any of the following keywords:

FOREIGN KEY
: identifies the column or combination of columns in the child table that makes up of the foreign key. Only use this keyword when you define a foreign key with a table constraint clause.

REFERENCES
: identifies the parent table and the column or combination of columns that make up the referenced key.

If you only identify the parent table and omit the column names, the foreign key automatically references the primary key of the parent table.

The corresponding columns of the referenced key and the foreign key must match in number and datatypes.

ON DELETE CASCADE — allows deletion of referenced key values in the parent table that have dependent rows in the child table and causes Oracle7 to automatically delete dependent rows from the child table to maintain referential integrity.

If you omit this option, Oracle7 forbids deletions of referenced key values in the parent table that have dependent rows in the child table.

Before you define a referential integrity constraint in the child table, the referenced UNIQUE or PRIMARY KEY constraint on the parent table must already be defined. Also, the parent table must be in your own schema or you must have REFERENCES privilege on the columns of the referenced key in the parent table. Before you enable a referential integrity constraint, its referenced constraint must be enabled.

You cannot define a referential integrity constraint in a CREATE TABLE statement that contains an AS clause. Instead, you can create the table without the constraint and then add it later with an ALTER TABLE statement.

A foreign key column cannot be of datatype LONG or LONG RAW. You can designate the same column or combination of columns as both a foreign key and a primary or unique key. You can also designate the same column or combination of columns as both a foreign key and a cluster key.

You can define multiple foreign keys in a table. Also, a single column can be part of more than one foreign key.

Defining Referential Integrity Constraints

You can use *column_constraint* syntax to define a referential integrity constraint in which the foreign key is made up of a single column.

Example VI

The following statement creates the EMP table and defines and enables a foreign key on the DEPTNO column that references the primary key on the DEPTNO column of the DEPT table:

```
CREATE TABLE emp
    (empno     NUMBER(4),
     ename     VARCHAR2(10),
     job       VARCHAR2(9),
     mgr       NUMBER(4),
     hiredate  DATE,
     sal       NUMBER(7,2),
     comm      NUMBER(7,2),
     deptno    CONSTRAINT fk_deptno REFERENCES dept(deptno) )
```

The constraint FK_DEPTNO ensures that all departments given for employees in the EMP table are present in the DEPT table. However, employees can have null department numbers, meaning they are not assigned to any department. If you wished to prevent the latter, you could create a NOT NULL constraint on the *deptno* column in the EMP table, in addition to the REFERENCES constraint.

Before you define and enable this constraint, you must define and enable a constraint that designates the DEPTNO column of the DEPT table as a primary or unique key. For the definition of such a constraint, see Example IV on page 4 – 155.

Note that the referential integrity constraint definition does not use the FOREIGN KEY keyword to identify the columns that make up the foreign key. Because the constraint is defined with a column constraint clause on the DEPTNO column, the foreign key is automatically on the DEPTNO column.

Note that the constraint definition identifies both the parent table and the columns of the referenced key. Because the referenced key is the parent table's primary key, the referenced key column names are optional.

Note that the above statement omits the DEPTNO column's datatype. Because this column is a foreign key, Oracle7 automatically assigns it the datatype of the DEPT.DEPTNO column to which the foreign key refers.

Alternatively, you can define a referential integrity constraint with *table_constraint* syntax:

```
CREATE TABLE emp
  (empno     NUMBER(4),
   ename     VARCHAR2(10),
   job       VARCHAR2(9),
   mgr       NUMBER(4),
   hiredate  DATE,
   sal       NUMBER(7,2),
   comm      NUMBER(7,2),
   deptno,
   CONSTRAINT fk_deptno
      FOREIGN KEY (deptno)
   REFERENCES dept(deptno) )
```

Note that the foreign key definitions in both of the above statements omit the ON DELETE CASCADE option, causing Oracle7 to forbid the deletion of a department if any employee works in that department.

Maintaining Referential Integrity with the ON DELETE CASCADE Option

If you use the ON DELETE CASCADE option, Oracle7 permits deletions of referenced key values in the parent table and automatically deletes dependent rows in the child table to maintain referential integrity.

Example VII

This example creates the EMP table, defines and enables the referential integrity constraint FK_DEPTNO, and uses the ON DELETE CASCADE option:

```
CREATE TABLE emp
   (empno     NUMBER(4),
   ename      VARCHAR2(10),
   job    VARCHAR2(9),
   mgr    NUMBER(4),
   hiredate   DATE,
   sal    NUMBER(7,2),
   comm       NUMBER(7,2),
   deptno     NUMBER(2)  CONSTRAINT fk_deptno
                         REFERENCES dept(deptno)
                         ON DELETE CASCADE )
```

Because of the ON DELETE CASCADE option, Oracle7 cascades any deletion of a DEPTNO value in the DEPT table to the DEPTNO values of its dependent rows of the EMP table. For example, if department 20 is deleted from the DEPT table, Oracle7 deletes the department's employees from the EMP table.

Referential Integrity Constraints with Composite Keys

A composite foreign key is a foreign key made up of a combination of columns. A composite foreign key can contain as many as 16 columns. To define a referential integrity constraint with a composite foreign key, you must use *table_constraint* syntax. You cannot use *column_constraint* syntax because this syntax can only impose rules on a single column. A composite foreign key must refer to a composite unique key or a composite primary key.

To satisfy a referential integrity constraint involving composite keys, each row in the child table must satisfy one of the following conditions:

- The values of the foreign key columns must match the values of the referenced key columns in a row in the parent table.
- The value of at least one of the columns of the foreign key must be null.

Example VIII The following statement defines and enables a foreign key on the combination of the AREACO and PHONENO columns of the PHONE_CALLS table:

```
ALTER TABLE phone_calls
   ADD CONSTRAINT fk_areaco_phoneno
      FOREIGN KEY (areaco, phoneno)
      REFERENCES customers(areaco, phoneno)
      EXCEPTIONS INTO wrong_numbers
```

The constraint FK_AREACO_PHONENO ensures that all the calls in the PHONE_CALLS table are made from phone numbers that are listed in the CUSTOMERS table. Before you define and enable this constraint, you must define and enable a constraint that designates the combination of the AREACO and PHONENO columns of the CUSTOMERS table as a primary or unique key.

The EXCEPTIONS option causes Oracle7 to write information to the WRONG_NUMBERS about any rows in the PHONE_CALLS table that violate the constraint.

CHECK Constraints

The CHECK constraint explicitly defines a condition. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null). For information on conditions, see the syntax description of *condition* on page 3 – 78. The condition of a CHECK constraint can refer to any column in the table, but it cannot refer to columns of other tables. CHECK constraint conditions cannot contain the following constructs:

- queries to refer to values in other rows
- calls to the functions SYSDATE, UID, USER, or USERENV
- the pseudocolumns CURRVAL, NEXTVAL, LEVEL, or ROWNUM
- date constants that are not fully specified

Whenever Oracle7 evaluates a CHECK constraint condition for a particular row, any column names in the condition refer to the column values in that row.

If you create multiple CHECK constraints for a column, design them carefully so their purposes do not conflict. Oracle7 does not verify that CHECK conditions are not mutually exclusive.

Example IX The following statement creates the DEPT table and defines a CHECK constraint in each of the table's columns:

```
CREATE TABLE dept  (deptno NUMBER CONSTRAINT check_deptno
                   CHECK (deptno BETWEEN 10 AND 99)
                   DISABLE,
dname VARCHAR2(9)  CONSTRAINT check_dname
                   CHECK (dname = UPPER(dname))
                   DISABLE,
loc VARCHAR2(10)   CONSTRAINT check_loc
                   CHECK (loc IN ('DALLAS','BOSTON',
                   'NEW YORK','CHICAGO'))
                   DISABLE)
```

Each constraint restricts the values of the column in which it is defined:

CHECK_DEPTNO ensures that no department numbers are less than 10 or greater than 99.

CHECK_DNAME ensures that all department names are in uppercase.

CHECK_LOC restricts department locations to Dallas, Boston, New York, or Chicago.

Unlike other types of constraints, a CHECK constraint defined with *column_constraint* syntax can impose rules on any column in the table, rather than only on the column in which it is defined.

Because each CONSTRAINT clause contains the DISABLE option, Oracle7 only defines the constraints and does not enforce them.

Example X The following statement creates the EMP table and uses a table constraint clause to define and enable a CHECK constraint:

```
CREATE TABLE emp
  (empno     NUMBER(4),
   ename     VARCHAR2(10),
   job       VARCHAR2(9),
   mgr       NUMBER(4),
   hiredate  DATE,
   sal       NUMBER(7,2),
   comm      NUMBER(7,2),
   deptno    NUMBER(2),
   CHECK (sal + comm <= 5000) )
```

This constraint uses an inequality condition to limit an employee's total compensation, the sum of salary and commission, to $5000:

- If an employee has non–null values for both salary and commission, the sum of these values must not be more than $5000 to satisfy the constraint.
- If an employee has a null salary or commission, the result of the condition is unknown and the employee automatically satisfies the constraint.

Because the CONSTRAINT clause in this example does not supply a constraint name, Oracle7 generates a name for the constraint.

Example XI The following statement defines and enables a PRIMARY KEY constraint, two referential integrity constraints, a NOT NULL constraint, and two CHECK constraints:

```
CREATE TABLE order_detail
    (CONSTRAINT pk_od PRIMARY KEY (order_id, part_no),
     order_id NUMBER
              CONSTRAINT fk_oid REFERENCES scott.order (order_id),
     part_no  NUMBER
              CONSTRAINT fk_pno REFERENCES scott.part (part_no),
     quantity NUMBER
              CONSTRAINT nn_qty NOT NULL
              CONSTRAINT check_qty_low CHECK (quantity > 0),
     cost     NUMBER
              CONSTRAINT check_cost CHECK (cost > 0) )
```

The constraints enforce the following rules on table data:

PK_OD identifies the combination of the ORDER_ID and PART_NO columns as the primary key of the table. To satisfy this constraint, the following conditions must be true:

- No two rows in the table can contain the same combination of values in the ORDER_ID and the PART_NO columns.
- No row in the table can have a null in either the ORDER_ID column or the PART_NO column.

FK_OID identifies the ORDER_ID column as a foreign key that references the ORDER_ID column in the ORDER table in SCOTT's schema. All new values added to the column ORDER_DETAIL.ORDER_ID must already appear in the column SCOTT.ORDER.ORDER_ID.

FK_PNO	identifies the PART_NO column as a foreign key that references the PART_NO column in the PART table owned by SCOTT. All new values added to the column ORDER_DETAIL.PART_NO must already appear in the column SCOTT.PART.PART_NO.
NN_QTY	forbids nulls in the QUANTITY column.
CHECK_QTY	ensures that values in the QUANTITY column are always greater than 0.
CHECK_COST	ensures the values in the COST column are always greater than 0.

This example also illustrates the following points about constraint clauses and column definitions:

- *Table_constraint* syntax and column definitions can appear in any order. In this example, note that the *table_constraint* syntax that defines the PK_OD constraint precedes the column definitions. In Example IV in this section, the *table_constraint* syntax defining the table's primary key follows the column definitions.
- A column definition can use *column_constraint* syntax multiple times. In this example, the definition of the QUANTITY column contains the definitions of both the NN_QTY and CHECK_QTY constraints.
- A table can have multiple CHECK constraints. Multiple CHECK constraints, each with a simple condition enforcing a single business rule is better than a single CHECK constraint with a complicated condition enforcing multiple business rules. When a constraint is violated, Oracle7 returns an error message identifying the constraint. Such an error message more precisely identifies the violated business rule if the identified constraint enforces a single business rule.

Related Topics

CREATE TABLE command on 4 – 245
ALTER TABLE command on 4 – 89
ENABLE clause on 4 – 326
DISABLE clauses on 4 – 295

CREATE CLUSTER

Purpose

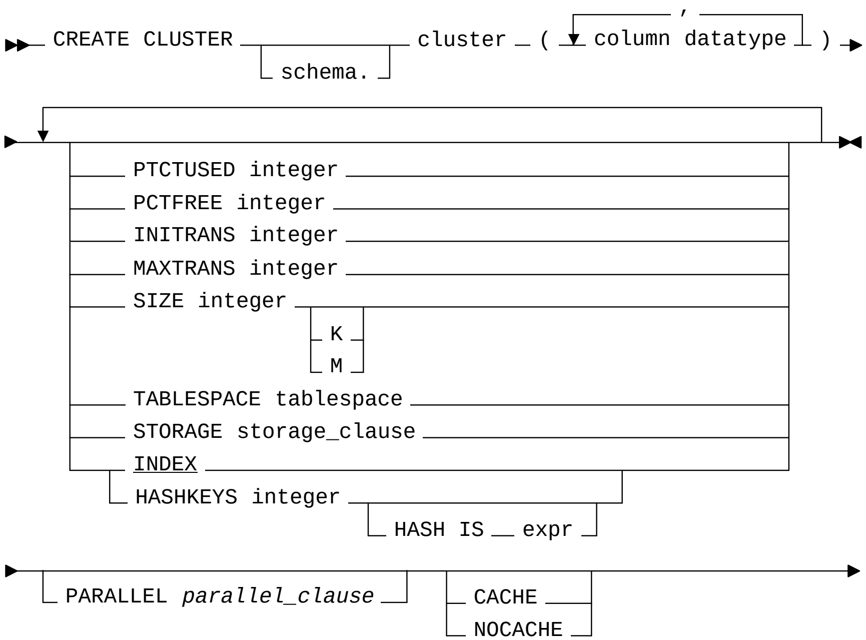
To create a cluster. A *cluster* is a schema object that contains one or more tables that all have one or more columns in common.

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 Oracle7 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.

Syntax



Keywords and Parameters

schema — is the schema to contain the cluster. If you omit *schema*, Oracle7 creates the cluster in your current schema.

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

column — is the name of a column in the cluster key.

datatype — is the datatype of a cluster key column. A cluster key column can have any datatype except LONG or LONG RAW. You cannot use the HASH IS clause if any column datatype is not INTEGER or NUMBER with scale 0. For information on datatypes, see the section "Datatypes" on page 2 – 18.

PCTUSED — specifies the limit that Oracle7 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 command on page 4 – 245.

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. If you omit this parameter, Oracle7 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. See the STORAGE clause on page 4 – 449.

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

HASHKEYS — creates a hash cluster and specifies the number of hash values for a hash cluster. Oracle7 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, Oracle7 creates an indexed cluster by default.

HASH IS — specifies a expression to be used as the hash function for the hash cluster.

The expression must:

- evaluate to a positive value
- contain one or more columns of datatype INTEGER or datatype NUMBER with scale 0.

The expression:

- cannot reference user defined PL/SQL functions
- cannot reference the following: SYSDATE, USERENV, TO_DATE, UID, USER, LEVEL, ROWNUM
- cannot evaluate to a constant value
- cannot contain a subquery
- cannot contain columns qualified with a schema or object name (other than the cluster name)

If you omit the HASH IS clause, Oracle7 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.

PARALLEL — specifies the degree of parallelism to use when creating the cluster and the default degree of parallelism to use when querying the cluster after creation. See the *parallel_clause* on page 4 – 378.

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

NOCACHE — specifies that the blocks retrieved for this table are placed at the least recently used end of the LRU list in the buffer cache when a full table scan is performed. This is the default behavior.

Usage Notes

A *cluster* is a schema object that contains one or more tables that all have one or more columns in common. Rows of one or more tables that share the same value in these common columns are physically stored together within the database.

Clustering provides more control over the physical storage of rows within the database. Clustering can reduce both the time it takes to access clustered tables and the space needed to store the table. After you create a cluster and add tables to it, the cluster is transparent. You can access clustered tables with SQL statements just as you can non–clustered tables.

If you cannot fit all rows for one hash value into a data block, do not use hash clusters. Performance is very poor in this circumstance because an insert or update of a row in a hash cluster with a size exceeding the data block size fills the block and row chaining to contain the rest of the row.

Generally, you should only cluster tables that are frequently joined on the cluster key columns in SQL statements. While clustering multiple tables improves the performance of joins, it is likely to reduce the performance of full table scans, INSERT statements, and UPDATE statements that modify cluster key values. Before clustering, consider its benefits and tradeoffs in light of the operations you plan to perform on your data. For more information on the performance implications of clustering, see the "Tuning SQL Statements" chapter of *Oracle7 Server Tuning*.

When you create a cluster in Trusted Oracle7, it is labeled with your DBMS label.

Cluster Keys

The columns defined by the CREATE CLUSTER command make up the *cluster key*. These cluster columns must correspond in both datatype and size to columns in each of the clustered tables, although they need not correspond in name.

You cannot specify integrity constraints as part of the definition of a cluster key column. Instead, you can associate integrity constraints with the tables that belong to the cluster.

Types of Clusters

A cluster can be one of the following types:

- indexed cluster
- hash cluster

Indexed Clusters

In an *indexed cluster*, Oracle7 stores rows having the same cluster key value together. Each distinct cluster key value is stored only once in each data block, regardless of the number of tables and rows in which it occurs. This saves disk space and improves performance for many operations.

You may want to use indexed clusters in the following cases:

- Your queries retrieve rows over a range of cluster key values.
- Your clustered tables may grow unpredictably.

After you create an indexed cluster, you must create an index on the cluster key before you can issue any Data Manipulation Language statements against a table in the cluster. This index is called the *cluster index*. For information on creating a cluster index, see the CREATE INDEX command on page 4 – 192. As with the columns of any index, the order of the columns in the cluster key affects the structure of the cluster index.

A cluster index provides quick access to rows within a cluster based on the cluster key. If you issue a SQL statement that searches for a row in the cluster based on its cluster key value, Oracle7 searches the cluster index for the cluster key value and then locates the row in the cluster based on its ROWID.

Hash Clusters

In a *hash cluster*, Oracle7 stores together rows that have the same hash key value. The *hash value* for a row is the value returned by the cluster's hash function. When you create a hash cluster, you can either specify a hash function or use the Oracle7 internal hash function. Hash values are not actually stored in the cluster, although cluster key values are stored for every row in the cluster.

You may want to use hash clusters in the following cases:

- Your queries retrieve rows based on equality conditions involving all cluster key columns.
- Your clustered tables are static or you can determine the maximum number of rows and the maximum amount of space required by the cluster when you create the cluster.

The hash function provides access to rows in the table based on the cluster key value. If you issue a SQL statement that locates a row in the cluster based on its cluster key value, Oracle7 applies the hash function to the given cluster key value and uses the resulting hash value to locate the matching rows. Because multiple cluster key values can map to the same hash value, Oracle7 must also check the row's cluster key value. Note that this process often results in less I/O than the process for the indexed cluster because the index search is not required.

Oracle7's internal hash function returns values ranging from 0 to the value of HASHKEYS – 1. If you specify a column with the HASH IS clause, the column values need not fall into this range. Oracle7 divides the column value by the HASHKEYS value and uses the remainder as the hash value. The hash value for null is HASHKEYS – 1. Oracle7 also rounds the HASHKEYS value up to the nearest prime number to obtain the actual number of hash values. This rounding reduces the likelihood of *hash collisions*, or multiple cluster key values having the same hash value.

You cannot create a cluster index for a hash cluster, and you need not create an index on a hash cluster key.

Cluster Size

Oracle7 uses the value of the SIZE parameter to determine the space reserved for rows corresponding to one cluster key value or one hash value. This space then determines the maximum number of cluster or hash values stored in a data block. If the SIZE value is not a divisor of the data block size, Oracle7 uses the next largest divisor. If the SIZE value is larger than the data block size, Oracle7 uses the operating system block size, reserving at least one data block per cluster or hash value.

Oracle7 also considers the length of the cluster key when determining how much space to reserve for the rows having a cluster key value. Larger cluster keys require larger sizes. To see the actual size, query the KEY_SIZE column of the USER_CLUSTERS data dictionary view. This does not apply to hash clusters because hash values are not actually stored in the cluster.

Although the maximum number of cluster and hash key values per data block is fixed on a per cluster basis, Oracle7 does not reserve an equal amount of space for each cluster or hash key value. Varying this space stores data more efficiently because the data stored per cluster or hash key value is rarely fixed.

A SIZE value smaller than the space needed by the average cluster or hash key value may require the data for one cluster key or hash key value to occupy multiple data blocks. A SIZE value much larger results in wasted space.

When you create a hash cluster, Oracle7 immediately allocates space for the cluster based on the values of the SIZE and HASHKEYS parameters. For more information on how Oracle7 allocates space for clusters, see the “Schema Objects” chapter of *Oracle7 Server Concepts*.

Adding Tables to a Cluster

You can add tables to an existing cluster by issuing a CREATE TABLE statement with the CLUSTER clause. A cluster can contain as many as 32 tables, although the performance gains of clustering are often negated in clusters of more than four or five tables.

All tables in the cluster have the cluster’s storage characteristics as specified by the PCTUSED, PCTFREE, INITRANS, MAXTRANS, TABLESPACE, and STORAGE parameters.

Example I The following statement creates an indexed cluster named PERSONNEL with the cluster key column DEPARTMENT_NUMBER, a cluster size of 512 bytes, and storage parameter values:

```
CREATE CLUSTER personnel
    ( department_number  NUMBER(2) )
    SIZE 512
    STORAGE (INITIAL 100K NEXT 50K PCTINCREASE 10)
```

The following statements add the EMP and DEPT tables to the cluster:

```
CREATE TABLE emp
    (empno     NUMBER         PRIMARY KEY,
     ename     VARCHAR2(10)  NOT NULL
                              CHECK (ename = UPPER(ename)),
     job       VARCHAR2(9),
     mgr       NUMBER         REFERENCES scott.emp(empno),
     hiredate  DATE           CHECK (hiredate >= SYSDATE),
     sal       NUMBER(10,2)  CHECK (sal > 500),
     comm      NUMBER(9,0)   DEFAULT NULL,
    deptno   NUMBER(2)    NOT NULL )
    CLUSTER personnel (deptno)

CREATE TABLE dept
    (deptno  NUMBER(2),
     dname   VARCHAR2(9),
     loc     VARCHAR2(9))
    CLUSTER personnel (deptno)
```

The following statement creates the cluster index on the cluster key of PERSONNEL:

```
CREATE INDEX idx_personnel ON CLUSTER personnel
```

After creating the cluster index, you can insert rows into either the EMP or DEPT tables.

Example II The following statement creates a hash cluster named PERSONNEL with the cluster key column DEPARTMENT_NUMBER, a maximum of 503 hash key values, each of size 512 bytes, and storage parameter values:

```
CREATE CLUSTER personnel
    ( department_number  NUMBER )
    SIZE 512  HASHKEYS 500
    STORAGE (INITIAL 100K  NEXT 50K  PCTINCREASE 10)
```

Because the above statement omits the HASH IS clause, Oracle7 uses the internal hash function for the cluster.

Example III The following statement creates a hash cluster named PERSONNEL with the cluster key comprised of the columns HOME_AREA_CODE and HOME_PREFIX, and uses a SQL expression containing these columns for the hash function:

```
CREATE CLUSTER personnel
 ( home_area_code  NUMBER,
   home_prefix     NUMBER )
  HASHKEYS 20
  HASH IS MOD(home_area_code + home_prefix, 101)
```

Related Topics

CREATE INDEX command on 4 – 192
CREATE TABLE command on 4 – 245
STORAGE clause on 4 – 449

CREATE CONTROLFILE

Purpose

To recreate a control file in one of the following 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.

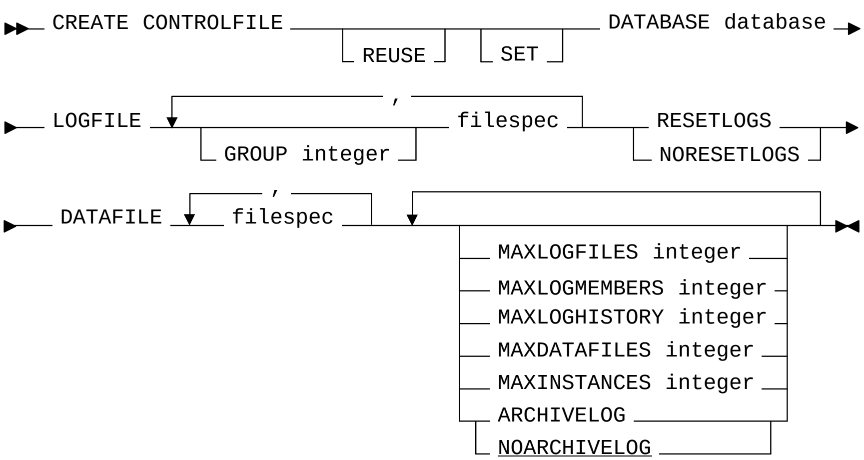
Warning: It is recommended that you perform a full backup of all files in the database before using this command.

Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your operating system label must be the equivalent of DBHIGH.

Syntax



Keywords and Parameters

REUSE	specifies that existing control files identified by the initialization parameter CONTROL_FILES can be reused, thus ignoring and overwriting any information they may currently contain. If you omit this option and any of these control files already exist, Oracle7 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. See the syntax description of *filespec* on page 4 – 343.
RESETLOGS	ignores the contents of the files listed in the LOGFILE clause. These files do not have to exist. Each *filespec* in the LOGFILE clause must specify the SIZE parameter. Oracle7 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 exit and must be the current redo log files rather than restored backups. Oracle7 reassigns the redo log file groups to the threads to which they were previously assigned and re–enables the threads as they were previously enabled. If you specify GROUP values, Oracle7 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. See the syntax description of *filespec* on page 4 – 343.

MAXLOGFILES specifies the maximum number of redo log file groups that can ever be created for the database. Oracle7 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.

OSDoc

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. Oracle7 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 Oracle7 Parallel Server. Oracle7 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 Oracle7 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, Oracle7 chooses noarchivelog mode by default. After creating the control file, you can change between archivelog mode and noarchivelog mode with the ALTER DATABASE command.

Usage Notes

It is recommended that you take a full backup of all files in the database before issuing a CREATE CONTROLFILE statement.

When you issue a CREATE CONTROLFILE statement, Oracle7 creates a new control file based on the information you specify in the statement. If you omit any of the options from the statement, Oracle7 uses the default options, rather than the options for the previous control file. After successfully creating the control file, Oracle7 mounts the database in exclusive mode. You then must perform media recovery before opening the database. It is recommended that you then shutdown the instance and take a full backup of all files in the database.

For more information on using this command, see the “Recovering a Database” chapter of *Oracle7 Server Administrator’s Guide.*

When you create a control file in Trusted Oracle7, it is labeled with your DBMS label. The control file cannot be used unless it is labeled at the operating system equivalent of DBHIGH. If you issue a CREATE CONTROLFILE statement in DBMS MAC mode, Trusted Oracle7 automatically switches to OS MAC mode. You can then return to DBMS MAC mode by issuing an ALTER DATABASE statement with the SET DBMAC ON clause.

Example This example recreates a control file:

```
CREATE CONTROLFILE REUSE
    SET DATABASE orders_2
    LOGFILE GROUP 1 ('diskb:log1.log', 'diskc:log1.log') SIZE 50K,
            GROUP 2 ('diskb:log2.log', 'diskc:log2.log') SIZE 50K
    NORESETLOGS
    DATAFILE 'diska:dbone.dat' SIZE 2M
    MAXLOGFILES 5
    MAXLOGHISTORY 100
    MAXDATAFILES 10
    MAXINSTANCES 2
    ARCHIVELOG
```

Related Topics

CREATE DATABASE command on 4 – 178

CREATE DATABASE

Purpose

To create a database, making it available for general use, with the following 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

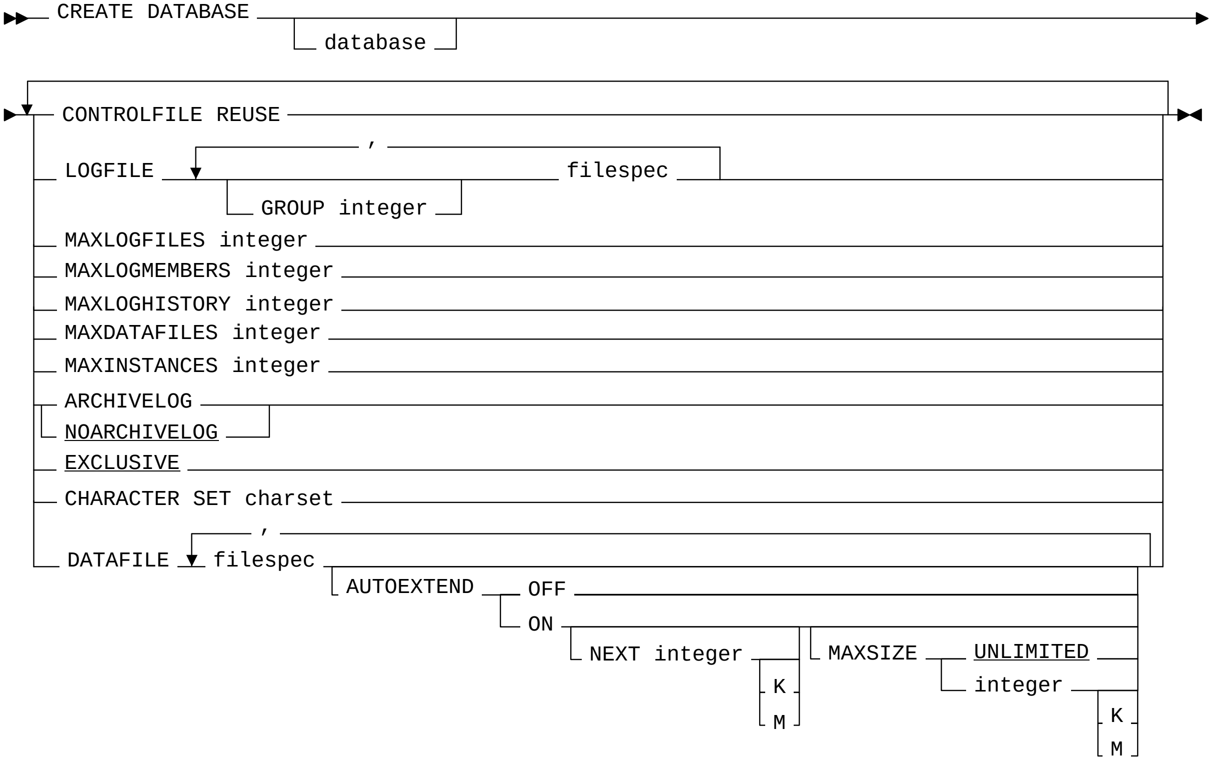
Warning: This command prepares a database for initial use and erases any data currently in the specified files. Only use this command when you understand its ramifications.

Prerequisites

You must have the OSDBA role enabled.

If you are using Trusted Oracle7 and you plan to use the database in DBMS MAC mode, your operating system label should be the equivalent of DBLOW.

Syntax



Keyword and Parameters

database

is the name of the database to be created and can be up to eight bytes long. Oracle7 writes this name into the control file. If you subsequently issue an ALTER DATABASE statement and that explicitly specifies a database name, Oracle7 verifies that name with the name in the control file. Database names should adhere to the rules described in section, "Object Naming Rules," on page 2 – 3.

Note: You cannot use special characters from European or Asian character sets in a database name. For example, the umlaut is not allowed.

The *database* cannot be a Server Manager reserved word as documented in the *Oracle Server Manager Manual*. If you omit the database name from a CREATE DATABASE statement, the name specified by the initialization parameter DB_NAME is used.

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, Oracle7 returns an error message.

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. See the syntax description of *filespec* on page 4 – 343. 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, Oracle7 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, Oracle7 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. Oracle7 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. Oracle7 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 Oracle7 with the Parallel Server option. Oracle7 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 Oracle7 with the Parallel Server option in 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 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. Oracle7 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 on page 4 – 16.

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.

DATAFILE

specifies one or more files to be used as data files. See the syntax description of *filespec* on page 4 – 343. These files all become part of the SYSTEM tablespace. If you omit this clause, Oracle7 creates one data file by default. The name and size of this default file depends on your operating system.

AUTOEXTEND	enables or disables the automatic extension of a datafile.	
	OFF	disable autoextend if it is turned on. NEXT and MAXSIZE are set to zero. Values for NEXT and MAXSIZE must be respecified in ALTER DATABASE AUTOEXTEND or ALTER TABLESPACE AUTOEXTEND commands.
	ON	enable autoextend.
	NEXT	the size in bytes of the next increment of disk space to be automatically allocated to the datafile when more extents are required. You can also use K or M to specify this size in kilobytes or megabytes. The default is one data block.
	MAXSIZE	maximum disk space allowed for automatic extension of the datafile.
	UNLIMITED	set no limit on allocating disk space to the datafile.

Usage Notes

This command erases all data in any specified data files that already exist to prepare them for initial database use. If you use the command on an existing database, all data in the data files is lost.

After creating the database, this command mounts it in exclusive mode and opens it, making it available for normal use.

If you create a database using Trusted Oracle7, it is labeled with your operating system label and is created in OS MAC mode. If you plan to use the database in DBMS MAC mode, be sure you set values for DBHIGH and DBLOW. For more information on creating Trusted Oracle7 databases, see *Trusted Oracle7 Server Administrator's Guide.*

Example The following statement creates a small database using defaults for all arguments:

```
CREATE DATABASE
```

The following statement creates a database and fully specifies each argument:

```
CREATE DATABASE newtest
    CONTROLFILE REUSE
    LOGFILE
        GROUP 1 ('diskb:log1.log', 'diskc:log1.log') SIZE 50K,
        GROUP 2 ('diskb:log2.log', 'diskc:log2.log') SIZE 50K
    MAXLOGFILES 5
    MAXLOGHISTORY 100
    DATAFILE 'diska:dbone.dat' SIZE 2M
    MAXDATAFILES 10
    MAXINSTANCES 2
    ARCHIVELOG
    EXCLUSIVE
    CHARACTER SET US7ASCII
    DATAFILE
        'disk1:df1.dbf' AUTOEXTEND ON
        'disk2:df2.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
```

Related Topics

ALTER DATABASE command on 4 – 16
CREATE ROLLBACK SEGMENT command on 4 – 218
CREATE TABLESPACE command on 4 – 254
STARTUP and SHUTDOWN commands in *Oracle Server Manager User's Guide.*

CREATE DATABASE LINK

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 Oracle7 or a non–Oracle7 database.

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.

Syntax

```
>>── CREATE ──┬────────┬── DATABASE LINK dblink ──────────────────────>
              └ PUBLIC ┘

>──┬─────────────────────────────────────────────┬──┬──────────────────────────┬──><
   └ CONNECT TO user IDENTIFIED BY password ─────┘  └ USING 'connect_string' ──┘
```

Keyword and Parameters

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 "Referring to Objects In Remote Databases," on page 2 – 11.

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 Oracle7 and can only be specified for public database links. For more information on specifying read–only mounts, see *Trusted Oracle7 Server Administrator's Guide*.

Usage Notes

You cannot create a database link in another user's schema and you cannot qualify *dblink* with the name of a schema. Since periods are permitted in names of database links, Oracle7 interprets the entire name, such as RALPH.LINKTOSALES, as the name of a database link in your schema rather than as a database link named LINKTOSALES in the schema RALPH.

Once you have created a database link, you can use it to refer to tables and views on the remote database. You can refer to a remote table or view in a SQL statement by appending *@dblink* to the table or view name. You can query a remote table or view with the SELECT command. If you are using Oracle7 with the distributed option, you can also access remote tables and views in any of the following commands:

- DELETE command on page 4 – 286
- INSERT command on page 4 – 361
- LOCK TABLE command on page 4 – 369
- UPDATE command on page 4 – 460

The number of different database links that can appear in a single statement is limited to the value of the initialization parameter OPEN_LINKS.

When you create a database link in Trusted Oracle7, it is labeled with your DBMS label.

Example The following statement defines a database link named SALES.HQ.ACME.COM that refers to user SCOTT with password TIGER on the database specified by the string D:BOSTON–MFG:

```
CREATE DATABASE LINK sales.hq.acme.com
    CONNECT TO scott IDENTIFIED BY tiger
    USING 'D:BOSTON–MFG'
```

Once this database link is created, you can query tables in the schema SCOTT on the remote database in this manner:

```
SELECT *
    FROM emp@sales.hq.acme.com
```

You can also use Data Manipulation Language commands to modify data on the remote database:

```
INSERT INTO accounts@sales.hq.acme.com(acc_no, acc_name, balance)
    VALUES (5001, 'BOWER', 2000)

UPDATE accounts@sales.hq.acme.com
    SET balance = balance + 500

DELETE FROM accounts@sales.hq.acme.com
    WHERE acc_name = 'BOWER'
```

You can also access tables owned by other users on the same database. This example assumes SCOTT has access to ADAM's DEPT table:

```
SELECT *
    FROM adams.dept@sales.hq.acme.com
```

The previous statement connects to the user SCOTT on the remote database and then queries ADAM's DEPT table.

A synonym may be created to hide the fact that SCOTT's EMP table is on a remote database. The following statement causes all future references to EMP to access a remote EMP table owned by SCOTT.

```
CREATE SYNONYM emp
    FOR scott.emp@sales.hq.acme.com
```

Related Topics

CREATE SYNONYM command on 4 – 241
DELETE command on page 4 – 286
INSERT command on page 4 – 361
LOCK TABLE command on page 4 – 369
SELECT command on 4 – 405
UPDATE command on page 4 – 460

CREATE FUNCTION

Purpose

To create a *user function*. A user function or *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.

User functions can be used as part of a SQL expression.

Prerequisites

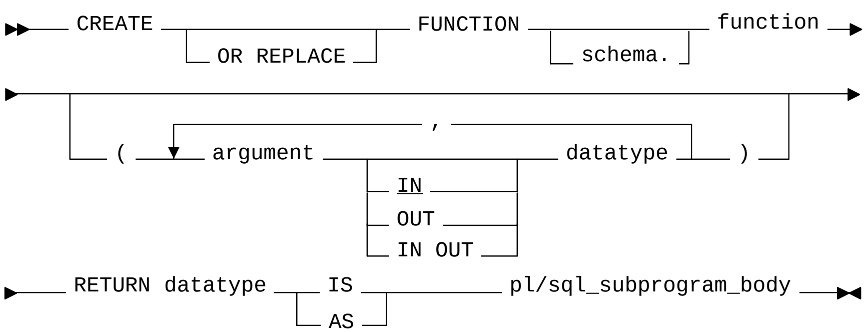
Before a stored function can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

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 Oracle7 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.

To create a stored function, you must be using Oracle7 with PL/SQL installed. For more information, see *PL/SQL User's Guide and Reference.*

Syntax



Keywords and Parameters

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, Oracle7 recompiles it. For information on recompiling functions, see the ALTER FUNCTION command on page 4 – 31.

  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*, Oracle7 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 the default.

OUT
: specifies the function will set the value of the argument.

IN OUT
: specifies that a value for the argument can be supplied by you and may be set by the function.

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. Oracle7 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. Oracle7 derives the length, precision, or scale of the return value from the environment from which the function is called. For information on PL/SQL datatypes, see the *PL/SQL User's Guide and Reference.*

pl/sql_subprogram_body

is the definition of the function. Function definitions are written in PL/SQL. For information on PL/SQL, including

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.

Usage Notes

A *stored function* is a set of PL/SQL statements that you can call by name. Functions are very similar to procedures, except that a function explicitly returns a value to its calling environment. For a general discussion of procedures and functions, see the CREATE PROCEDURE command on page 4 – 206.

The CREATE FUNCTION command creates a function as a stand–alone schema object. You can also create a function as part of a package. For information on creating packages, see the CREATE PACKAGE command 4 – 198.

When you create a stored function in Trusted Oracle7, it is labeled with your DBMS label.

Example The following statement creates the function GET_BAL:

```
CREATE FUNCTION get_bal(acc_no IN NUMBER)
    RETURN NUMBER
    IS
        acc_bal NUMBER(11,2);
    BEGIN
        SELECT balance
            INTO acc_bal
            FROM accounts
            WHERE account_id = acc_no;
        RETURN(acc_bal);
    END
```

The GET_BAL function returns the balance of a specified account.

When you call the function, you must specify the argument ACC_NO, the number of the account whose balance is sought. The datatype of ACC_NO is NUMBER.

The function returns the account balance. The RETURN clause of the CREATE FUNCTION statement specifies the datatype of the return value to be NUMBER.

The function uses a SELECT statement to select the BALANCE column from the row identified by the argument ACC_NO in the ACCOUNTS table. The function uses a RETURN statement to return this value to the environment in which the function is called.

The above function can be used in a SQL statement. For example:

```
SELECT get_bal(100) FROM DUAL;
```

Related Topics

ALTER FUNCTION command on 4 – 188
CREATE PACKAGE command on 4 – 198
CREATE PACKAGE BODY command on 4 – 202
CREATE PROCEDURE command on 4 – 206
DROP FUNCTION command on 4 – 304

CREATE INDEX

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.

Prerequisites

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

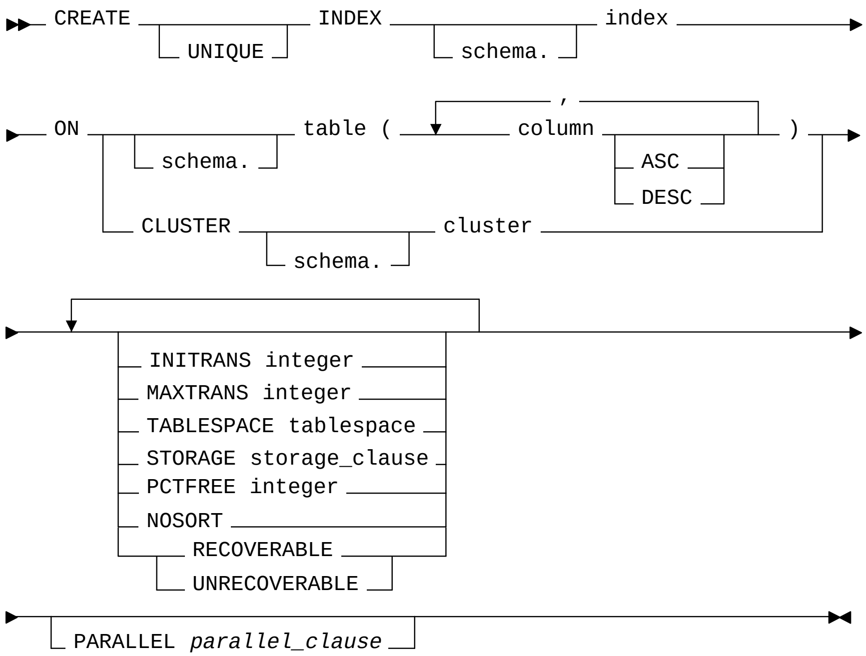
- The table or cluster to be indexed must be in your own schema.
- You must have INDEX privilege on the table to be indexed.
- You must have CREATE ANY INDEX system privilege.

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 Oracle7 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.

Syntax



Keywords and Parameters

UNIQUE — specifies that the value of the column (or combination of columns) in the table to be indexed must be unique.

schema — is the schema to contain the index. If you omit *schema*, Oracle7 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*, Oracle7 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*, Oracle7 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 on page 4 – 245.

TABLESPACE — is the name of the tablespace to hold the index. If you omit this option, Oracle7 creates the index in the default tablespace of the owner of the schema containing the index.

STORAGE — establishes the storage characteristics for the index. See the STORAGE clause on page 4 – 449.

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

NOSORT — indicates to Oracle7 that the rows are stored in the database in ascending order and therefore Oracle7 does not have to sort the rows when creating the index.

RECOVERABLE — specifies that the creation of the index will be logged in the redo log file. This is the default.

If the database is run in ARCHIVELOG mode, media recovery from a backup will recreate the index. You cannot specify RECOVERABLE when using NOARCHIVELOG mode.

UNRECOVERABLE — specifies that the creation of the index will not be logged in the redo log file. As a result, media recovery will not recreate the index.

Using this keyword makes index creation faster than using the RECOVERABLE option because redo log entries are not written.

PARALLEL — specifies the degree of parallelism for creating the index. See the *parallel_clause* on page 4 – 378.

Usage Notes

An index is an ordered list of all the values that reside in a group of one or more columns at a given time. Such a list makes queries that test the values in those columns vastly more efficient. Indexes also take up data storage space, however, and must be changed whenever the data is, so a cost–benefit analysis must be made in each case to determine whether and how indexes should be used. Oracle7 can use indexes to improve performance when:

- searching for rows with specified index column values
- accessing tables in index column order

When you initially insert rows into a new table, it is generally faster to create the table, insert the rows, and then create the index. If you create the index before inserting the rows, Oracle7 must update the index for every row inserted.

Oracle recommends that you do not explicitly define UNIQUE indexes on tables; uniqueness is strictly a logical concept and should be associated with the definition of a table. Alternatively, define UNIQUE integrity constraints on the desired columns. Oracle enforces UNIQUE integrity constraints by automatically defining a unique index on the unique key. Exceptions to this recommendation are usually performance related. For example, using a CREATE TABLE ... AS SELECT with a UNIQUE constraint is very much slower than creating the table without the constraint and then manually creating the UNIQUE index.

If indexes contain NULLs, the NULLS generally are considered distinct values. There is, however, one exception: if all the non–NULL values in two or more rows of an index are identical, the rows are considered identical; therefore, UNIQUE indexes prevent this from occurring. This does not apply if there are no non–NULL values—in other words, if the rows are entirely NULL..

When you create an index in Trusted Oracle7, it is labeled with your DBMS label.

Index Columns

An index can contain a maximum of 16 columns. The index entry becomes the concatenation of all data values from each column. You can specify the columns in any order. The order you choose is important to how Oracle7 uses the index.

When appropriate, Oracle7 uses the entire index or a leading portion of the index. Assume an index named IDX1 is created on columns A, B, and C of table TAB1 (in the order A, B, C). Oracle7 uses the index for references to columns A, B, C (the entire index); A, B; or just column A.

References to columns B and C do not use the IDX1 index. Of course, you can also create another index just for columns B and C.

Multiple Indexes Per Table

Unlimited indexes can be created for a table provided that the combination of columns differ for each index. You can create more than one index using the same columns provided that you specify distinctly different combinations of the columns. For example, the following statements specify valid combinations:

```
CREATE INDEX emp_idx1 ON emp (ename, job);
CREATE INDEX emp_idx2 ON emp (job, ename);
```

You cannot create an index that references only one column in a table if another such index already exists.

Note that each index increases the processing time needed to maintain the table during updates to indexed data.

Note that there is overhead in maintaining indexes when a table is updated. Thus, updating a table with a single index will take less time than if the table had five indexes.

The NOSORT Option

The NOSORT option can substantially reduce the time required to create an index. Normal index creation first sorts the rows of the table based on the index columns and then builds the index. The sort operation is often a substantial portion of the total work involved. If the rows are physically stored in ascending order (based on the indexed column values), then the NOSORT option causes Oracle7 to bypass the sort phase of the process.

You cannot use the NOSORT option to create a cluster index.

The NOSORT option also reduces the amount of space required to build the index. Oracle7 uses temporary segments during the sort. Since a sort is not performed, the index is created with much less temporary space.

To use the NOSORT option, you must guarantee that the rows are physically sorted in ascending order. Because of the physical data independence inherent in relational database management systems, especially Oracle7, there is no way to force a physical internal order on a table. The CREATE INDEX command with the NOSORT option should be used immediately after the initial load of rows into a table.

You run no risk by trying the NOSORT option. If your rows are not in the ascending order, Oracle7 returns an error. You can issue another CREATE INDEX without the NOSORT option.

UNRECOVERABLE

The UNRECOVERABLE option may substantially reduce the time required to create a large index. This feature is particularly useful after creating a large table or cluster in parallel. For backup and recovery considerations, see *Oracle7 Server Administrator's Guide.*

Example I

To quickly create an index in parallel on a table that was created using a fast parallel load (so all rows are already sorted), you might issue the following statement:

```
CREATE INDEX i_loc
    ON big_table (akey)
    NOSORT
    UNRECOVERABLE
    PARALLEL (DEGREE 5)
```

Nulls

Nulls are not indexed.

Example II

Consider the following statement:

```
SELECT ename
    FROM emp
    WHERE comm IS NULL
```

The above query does not use an index created on the COMM column.

Creating Cluster Indexes

Oracle7 does not automatically create an index for a cluster when the cluster is initially created. Data Manipulation Language statements cannot be issued against clustered tables until a cluster index has been created.

Example III

To create an index for the EMPLOYEE cluster, issue the following statement:

```
CREATE INDEX ic_emp
    ON CLUSTER employee
```

Note that no index columns are specified since the index is automatically built on all the columns of the cluster key.

Related Topics

ALTER INDEX command on 4 – 33
DROP INDEX command on 4 – 306
CONSTRAINT clause on 4 – 149
STORAGE clause on 4 – 449

CREATE PACKAGE

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.

Prerequisites

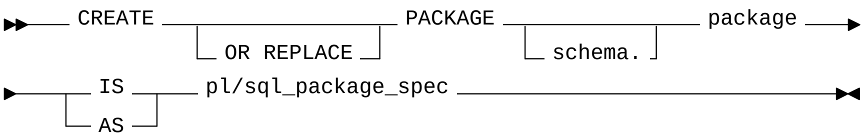
Before a package can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

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 Oracle7 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 Oracle7 with PL / SQL installed. For more information, see *PL/SQL User's Guide and Reference.*

Syntax



Keywords and Parameters

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, Oracle7 recompiles it. For information on recompiling package specifications, see the ALTER PROCEDURE command on page 4 – 42.

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

schema
is the schema to contain the package. If you omit *schema*, Oracle7 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. For information on PL / SQL, including writing package specifications, see *PL/SQL User's Guide and Reference***.**

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.

Packages

A *package* is an encapsulated collection of related program objects stored together in the database. Program objects are:

- procedures
- functions
- variables
- constants
- cursors
- exceptions

Using packages is an alternative to creating procedures and functions as stand–alone schema objects. Packages have many advantages over stand–alone procedures and functions:

- Packages allow you to organize your application development more efficiently.
- Packages allow you to grant privileges more efficiently.
- Packages allow you to modify package objects without recompiling dependent schema objects.
- Packages allow Oracle7 to read multiple package objects into memory at once.
- Packages can contain global variables and cursors that are available to all procedures and functions in the package.
- Packages allow you to overload procedures or functions. *Overloading* a procedure means creating multiple procedures with the same name in the same package, each taking arguments of different number or datatype.

For more information on these and other benefits of packages, see the "Using Procedures and Packages" chapter of the *Oracle7 Server Application Developer's Guide.*

When you create a package in Trusted Oracle7, it is labeled with your DBMS label.

How to Create Packages

To create a package, you must perform two distinct steps:

1. Create the package specification with the CREATE PACKAGE command. You can declare program objects in the package specification. Such objects are called *public* objects. Public objects can be referenced outside the package as well as by other objects in the package.

2. Create the package body with the CREATE PACKAGE BODY command. You can declare and define program objects in the package body:

- You must define public objects declared in the package specification.
- You can also declare and define additional package objects. Such objects are called *private* objects. Since private objects are declared in the package body rather than in the package specification, they can only be referenced by other objects in the package. They cannot be referenced outside the package.

See the CREATE PACKAGE BODY command 4 – 202.

The Separation of Specification and Body

Oracle7 stores the specification and body of a package separately in the database. Other schema objects that call or reference public program objects depend only on the package specification, not on the package body. This distinction allows you to change the definition of a program object in the package body without causing Oracle7 to invalidate other schema objects that call or reference the program object. Oracle7 only invalidates dependent schema objects if you change the declaration of the program object in the package specification.

Example This SQL statement creates the specification of the EMP_MGMT package:

```
CREATE PACKAGE emp_mgmt AS
   FUNCTION hire(ename VARCHAR2, job VARCHAR2, mgr NUMBER,
            sal NUMBER, comm NUMBER, deptno NUMBER)
      RETURN NUMBER;
   FUNCTION create_dept(dname VARCHAR2, loc VARCHAR2)
      RETURN NUMBER;
   PROCEDURE remove_emp(empno NUMBER);
   PROCEDURE remove_dept(deptno NUMBER);
   PROCEDURE increase_sal(empno NUMBER, sal_incr NUMBER);
   PROCEDURE increase_comm(empno NUMBER, comm_incr NUMBER);
   no_comm EXCEPTION;
   no_sal EXCEPTION;
   END emp_mgmt
```

The specification for the EMP_MGMT package declares the following public program objects:

- the functions HIRE and CREATE_DEPT
- the procedures REMOVE_EMP, REMOVE_DEPT, INCREASE_SAL, and INCREASE_COMM
- the exceptions NO_COMM and NO_SAL

All of these objects are available to users who have access to the package. After creating the package, you can develop applications that call any of the package's public procedures or functions or raise any of the package's public exceptions.

Before you can call this package's procedures and functions, you must define these procedures and functions in the package body. For an example of a CREATE PACKAGE BODY statement that creates the body of the EMP_MGMT package, see the CREATE PACKAGE BODY command on page 4 – 202.

Related Topics

ALTER PACKAGE command on 4 – 39
CREATE FUNCTION command on 4 – 188
CREATE PROCEDURE command on 4 – 206
CREATE PACKAGE BODY command on 4 – 202
DROP PACKAGE command 4 – 307

CREATE PACKAGE BODY

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.

Prerequisites

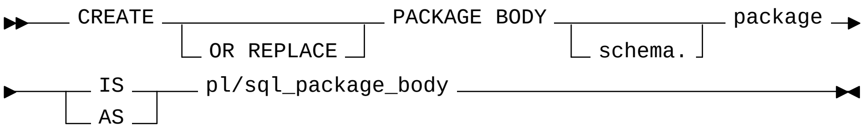
Before a package can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

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 Oracle7 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 Oracle7 with PL/SQL installed. For more information, see *PL/SQL User's Guide and Reference.*

Syntax



Keywords and Parameters

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, Oracle7 recompiles it. For information on recompiling package bodies, see the ALTER PACKAGE BODY command on page 4 – 39.

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*, Oracle7 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. For information on PL/SQL, including writing package bodies, see *PL/SQL User's Guide and Reference.*

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.

Packages

A *package* is an encapsulated collection of related procedures, functions, and other program objects stored together in the database. Packages are an alternative to creating procedures and functions as stand–alone schema objects. For a discussion of packages, including how to create packages, see the CREATE PACKAGE command on page 4 – 198.

Example

This SQL statement creates the body of the EMP_MGMT package:

```
CREATE PACKAGE BODY emp_mgmt AS
    tot_emps  NUMBER;
    tot_depts NUMBER;

    FUNCTION hire(ename VARCHAR2, job VARCHAR2, mgr NUMBER,
                sal NUMBER, comm NUMBER, deptno NUMBER)
        RETURN NUMBER IS
          new_empno NUMBER(4);
        BEGIN
          SELECT empseq.NEXTVAL
            INTO new_empno
            FROM DUAL;
          INSERT INTO emp
            VALUES (new_empno, ename, job, mgr, sal, comm, deptno,
            tot_emps := tot_emps + 1;
            RETURN(new_empno);
        END;

    FUNCTION create_dept(dname VARCHAR2, loc VARCHAR2)
        RETURN NUMBER IS
            new_deptno NUMBER(4);
        BEGIN
          SELECT deptseq.NEXTVAL
            INTO new_deptno
            FROM dual;
          INSERT INTO dept
            VALUES (new_deptno, dname, loc);
          tot_depts := tot_depts + 1;
          RETURN(new_deptno);
        END;
```

```
PROCEDURE remove_emp(empno NUMBER) IS
    BEGIN
      DELETE FROM emp
        WHERE emp.empno = remove_emp.empno;
      tot_emps := tot_emps – 1;
    END;

PROCEDURE remove_dept(deptno NUMBER) IS
    BEGIN
      DELETE FROM dept
        WHERE dept.deptno = remove_dept.deptno;
      tot_depts := tot_depts – 1;
      SELECT COUNT(*)
        INTO tot_emps
        FROM emp;
    /* In case Oracle7 deleted employees from the EMP table
      to enforce referential integrity constraints, reset
      the value of the variable TOT_EMPS to the total
      number of employees in the EMP table. */
    END;

PROCEDURE increase_sal(empno NUMBER, sal_incr NUMBER) IS
        curr_sal NUMBER(7,2);
    BEGIN
      SELECT sal
        INTO curr_sal
        FROM emp
        WHERE emp.empno = increase_sal.empno;
      IF curr_sal IS NULL
      THEN RAISE no_sal;
      ELSE
        UPDATE emp
          SET sal = sal + sal_incr
            WHERE empno = empno;
      END IF;
    END;

PROCEDURE increase_comm(empno NUMBER, comm_incr NUMBER) IS
  curr_comm NUMBER(7,2);
    BEGIN
      SELECT comm
        INTO curr_comm
        FROM emp
        WHERE emp.empno = increase_comm.empno
      IF curr_comm IS NULL
      THEN RAISE no_comm;
      ELSE
        UPDATE emp
          SET comm = comm + comm_incr;
```

```
        END IF;
      END;

END emp_mgmt
```

This package body corresponds to the package specification in the example of the CREATE PACKAGE statement earlier in this chapter. The package body defines the public program objects declared in the package specification:

- the functions HIRE and CREATE_DEPT
- the procedures REMOVE_EMP, REMOVE_DEPT, INCREASE_SAL, and INCREASE_COMM

Since these objects are declared in the package specification, they can be called by application programs, procedures, and functions outside the package. For example, if you have access to the package, you can create a procedure INCREASE_ALL_COMMS separate from the EMP_MGMT package that calls the INCREASE_COMM procedure.

Since these objects are defined in the package body, you can change their definitions without causing Oracle7 to invalidate dependent schema objects. For example, if you subsequently change the definition of HIRE, Oracle7 need not recompile INCREASE_ALL_COMMS before executing it.

The package body in this example also declares private program objects, the variables TOT_EMPS and TOT_DEPTS. Since these objects are declared in the package body rather than the package specification, they are accessible to other objects in the package, but they are not accessible outside the package. For example, you cannot develop an application that explicitly changes the value of the variable TOT_DEPTS. However, since the function CREATE_DEPT is part of the package, CREATE_DEPT can change the value of TOT_DEPTS.

Related Topics

ALTER PACKAGE command on 4 – 39
CREATE FUNCTION command on 4 – 188
CREATE PROCEDURE command on 4 – 206
CREATE PACKAGE command on 4 – 198
DROP PACKAGE command 4 – 307

CREATE PROCEDURE

Purpose

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

Prerequisites

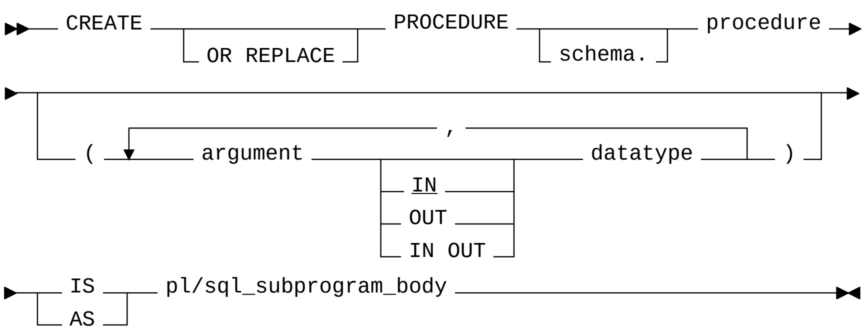
Before a procedure can be created, the user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary depending on your operating system.

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. To replace a procedure in another schema, you must have REPLACE ANY PROCEDURE system privilege.

If you are using Trusted Oracle7 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 Oracle7 with PL/SQL installed. For more information, see *PL/SQL User's Guide and Reference.*

Syntax



Keywords and Parameters

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, Oracle7 recompiles it. For information on recompiling procedures, see the ALTER PROCEDURE command on page 4 – 42.

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*, Oracle7 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*. As long as no length specifier is used, an argument can have any datatype supported by PL/SQL. For information on PL/SQL datatypes, see *PL/SQL User's Guide and Reference.*

Datatypes are specified without a length, precision, or scale. For example, VARCHAR2(10) is not valid, but VARCHAR2 is valid. Oracle7 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. For information on PL/SQL, including how to write a PL/SQL subprogram body, see *PL/SQL User's Guide and Reference.*

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.

Usage Notes

A *procedure* is a group of PL/SQLstatements that you can call by name. Stored procedures and stored functions are similar in many ways. This discussion applies to functions as well as to procedures. For information specific to functions, see the CREATE FUNCTION command on page 4 – 188.

With PL/SQL, you can group multiple SQL statements together with procedural PL/SQL statements similar to those in programming languages such as Ada and C. With the CREATE PROCEDURE command, you can create a procedure and store it in the database. You can call a stored procedure from any environment from which you can issue a SQL statement.

Stored procedures offer you advantages in the following areas:

- development
- integrity
- security
- performance
- memory allocation

For more information on stored procedures, including how to call stored procedures, see the "Using Procedures and Packages" chapter of *Oracle7 Server Application Developer's Guide.*

When you create a procedure in Trusted Oracle7, it is labeled with your DBMS label.

The CREATE PROCEDURE command creates a procedure as a stand–alone schema object. You can also create a procedure as part of a package. For information on creating packages, see the CREATE PACKAGE command on page 4 – 198.

Example

The following statement creates the procedure CREDIT in the schema SAM:

```
CREATE PROCEDURE sam.credit (acc_no IN NUMBER, amount IN NUMBER)
   AS BEGIN
         UPDATE accounts
            SET balance = balance + amount
            WHERE account_id = acc_no;
   END;
```

The CREDIT procedure credits a specified bank account with a specified amount. When you call the procedure, you must specify the following arguments:

ACC_NO — This argument is the number of the bank account to be credited. The argument's datatype is NUMBER.

AMOUNT — This argument is the amount of the credit. The argument's datatype is NUMBER.

The procedure uses an UPDATE statement to increase the value in the BALANCE column of the ACCOUNTS table by the value of the argument AMOUNT for the account identified by the argument ACC_NO.

Related Topics

ALTER PPROCEDURE command on 4 – 42
CREATE FUNCTION command on 4 – 188
CREATE PACKAGE command on 4 – 198
CREATE PROCEDURE BODY command on 4 – 202
DROP PPROCEDURE command 4 – 309

CREATE PROFILE

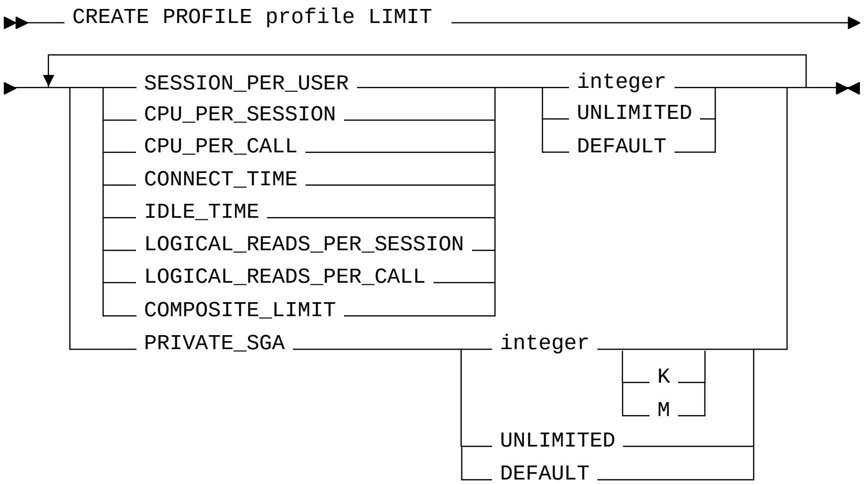
Purpose

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.

Prerequisites

You must have CREATE PROFILE system privilege.

Syntax



Keywords and Parameters

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.

Oracle7 calculates the total resource cost as a weighted sum of the following 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 on page 4 – 46.

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.

Usage Notes

In Trusted Oracle7, the new profile is automatically labeled with your DBMS label.

Using Profiles

A *profile* is a set of limits on database resources. You can use profiles to limit the database resources available to a user for a single call or a single session. Oracle7 enforces resource limits in the following ways:

- If a user exceeds the CONNECT_TIME or IDLE_TIME session resource limit, Oracle7 rolls back the current transaction and ends the session. When the user process next issues a call to Oracle7, an error message is returned.
- If a user attempts to perform an operation that exceeds the limit for other session resources, Oracle7 aborts the operation, rolls back the current statement, and immediately returns an error. The user can then commit or roll back the current transaction. The user must then end the session.
- If a user attempts to perform an operation that exceeds the limit for a single call, Oracle7 aborts the operation, rolls back the current statement, and returns an error message, leaving the current transaction intact.

How to Limit Resources

To specify resource limits for a user, you must perform both of the following operations:

Enable resource limits: You can enable resource limits through one of the following ways:

- You can enable resources limits with the initialization parameter RESOURCE_LIMIT.
- You can enable resource limits dynamically with the ALTER SYSTEM command. See the ALTER SYSTEM command 4 – 76.

Specify resource limits: To specify a resource limit for a user, you must perform following steps:

1. Create a profile that defines the limits using the CREATE PROFILE command.
2. Assign the profile to the user using the CREATE USER or ALTER USER command.

Note that you can specify resource limits for users regardless of whether resource limits are enabled. However, Oracle7 does not enforce these limits until you enable them.

The DEFAULT Profile

Oracle7 automatically creates a default profile named DEFAULT. This profile initially defines unlimited resources. You can change the limits defined in this profile with the ALTER PROFILE command.

Any user who is not explicitly assigned a profile is subject to the limits defined in the DEFAULT profile. Also, if the profile that is explicitly assigned to a user omits limits for some resources or specifies DEFAULT for some limits, the user is subject to the limits on those resources defined by the DEFAULT profile.

Example

The following statement creates the profile SYSTEM_MANAGER:

```
CREATE PROFILE system_manager
    LIMIT   SESSIONS_PER_USER          UNLIMITED
            CPU_PER_SESSION            UNLIMITED
            CPU_PER_CALL               3000
            CONNECT_TIME               45
            LOGICAL_READS_PER_SESSION  DEFAULT
            LOGICAL_READS_PER_CALL     1000
            PRIVATE SGA                15K
            COMPOSITE_LIMIT            5000000
```

If you then assign the SYSTEM_MANAGER profile to a user, the user is subject to the following limits in subsequent sessions:

- The user can have any number of concurrent sessions.
- In a single session, the user can consume an unlimited amount of CPU time.
- A single call made by the user cannot consume more than 30 seconds of CPU time.
- A single session cannot last for more than 45 minutes.
- In a single session, the number of data blocks from memory and disk is subject to the limit specified in the DEFAULT profile.
- A single call made by the user cannot read more than 1000 total data blocks from memory and disk.
- A single session cannot allocate more than 15 kilobytes of memory in the SGA.
- In a single session, the total resource cost cannot exceed 5 million service units. The formula for calculating the total resource cost is specified by the ALTER RESOURCE COST command.
- Since the SYSTEM_MANAGER profile omits a limit for IDLE_TIME, the user is subject to the limit on this resource specified in the DEFAULT profile.

Related Topics

ALTER PROFILE command on 4 – 44
ALTER RESOURCE COST command on 4 – 46
ALTER SYSTEM command on 4 – 76
ALTER USER command on 4 – 108
DROP PROFILE command on 4 – 311

CREATE ROLE

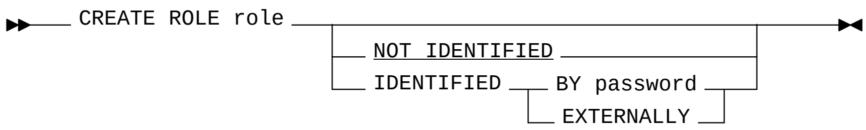
Purpose

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

Prerequisites

You must have CREATE ROLE system privilege.

Syntax



Keywords and Parameters

role
is the name of the role to be created. It is recommended 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 Oracle7 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.

Usage Notes

In Trusted Oracle7, the new role is automatically labeled with your DBMS label.

Using Roles

A *role* is a set of privileges that can be granted to users or to other roles. You can use roles to administer database privileges. You can add privileges to a role's privilege domain and then grant the role to a user. The user can then enable the role and exercise the privileges in the role's privilege domain. For information on enabling roles, see the ALTER USER command on page 4 – 108.

A role's privilege domain contains all privileges granted to the role and all privileges in the privilege domains of the other roles granted to it. A new role's privilege domain is initially empty. You can add privileges to a role's privilege domain with the GRANT command.

When you create a role, Oracle7 grants you the role with ADMIN OPTION. The ADMIN OPTION allows you to perform the following operations:

- grant the role to another user or role
- revoke the role from another user or role
- alter the role to change the authorization needed to access it
- drop the role

Roles Defined by Oracle7

Some roles are defined by SQL scripts provided on your distribution media. The following roles are predefined:

- CONNECT
- RESOURCE
- DBA
- EXP_FULL_DATABASE
- IMP_FULL_DATABASE

The CONNECT, RESOURCE, and DBA roles are provided for compatibility with previous versions of Oracle7. You should not rely on these roles, rather, it is recommended that you to design your own roles for database security. These roles may not be created automatically by future versions of Oracle7.

The EXP_FULL_DATABASE and IMP_FULL_DATABASE roles are provided for convenience in using the Import and Export utilities.

For more information on these roles, see Table 4 – 12 on page 4 – 352.

Oracle7 also creates other roles that authorize you to administer the database. On many operating systems, these roles are called OSOPER and OSDBA. Their names may be different on your operating system.

Example The following statement creates the role TELLER:

```
CREATE ROLE teller
    IDENTIFIED BY cashflow
```

Users who are subsequently granted the TELLER role must specify the passwords CASHFLOW to enable the role.

Related Topics

ALTER ROLE command on 4 – 49
DROP ROLE command on 4 – 312
GRANT (System Privileges and Roles) command on 4 – 346
REVOKE (System Privileges and Roles) command on 4 – 388
SET ROLE command on 4 – 442

CREATE ROLLBACK SEGMENT

Purpose

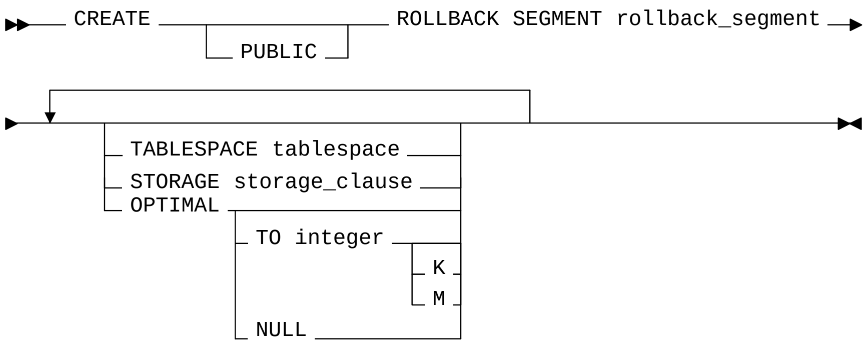
To create a rollback segment. A *rollback segment* is an object that Oracle7 uses to store data necessary to reverse, or undo, changes made by transactions.

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 Oracle7 in DBMS MAC mode, your DBMS label must dominate the tablespace's label.

Syntax



Keyword and Parameters

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, Oracle7 creates the rollback segment in the SYSTEM tablespace.

STORAGE
: specifies the characteristics for the rollback segment. See the STORAGE clause on page 4 – 449.

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. Oracle7 tries to maintain this size for the rollback segment by dynamically deallocating extents when their data is no longer needed for active transactions. Oracle7 deallocates as many extents as possible without reducing the total size of the rollback segment below the OPTIMAL value.

NULL

specifies no optimal size for the rollback segment, meaning that Oracle7 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. Oracle7 rounds values to the next multiple of the data block size.

Usage Notes

The tablespace must be online for you to add a rollback segment to it.

When you create a rollback segment, it is initially offline. To make it available for transactions by your Oracle7 instance, you must bring it online using one of the following:

- ALTER ROLLBACK SEGMENT command
- ROLLBACK_SEGMENTS initialization parameter

For more information on creating rollback segments and making them available, see the "Managing Rollback Segments" chapter of the *Oracle7 Server Administrator's Guide.*

A tablespace can have multiple rollback segments. Generally, multiple rollback segments improve performance. When you create a rollback segment in Trusted Oracle7, it is labeled with your DBMS label.

Example The following statement creates a rollback segment with default storage values in the system tablespace:

```
CREATE ROLLBACK SEGMENT rbs_2
    TABLESPACE system;
```

The above statement is the equivalent of the following:

```
CREATE ROLLBACK SEGEMENT rbs_2
    TABLESPACE system
    STORAGE
    (   INITIAL 2
        MINEXTENTS 121
        MAXEXTENTS 10240
        NEXT 10240
        PCT_INCREASE 0 )
```

Related Topics

CREATE TABLESPACE command on 4 – 254
CREATE DATABASE command on 4 – 178
ALTER ROLLBACK SEGMENT command on 4 – 50
DROP ROLLBACK SEGMENT command on 4 – 313
STORAGE clause on 4 – 449

CREATE SCHEMA

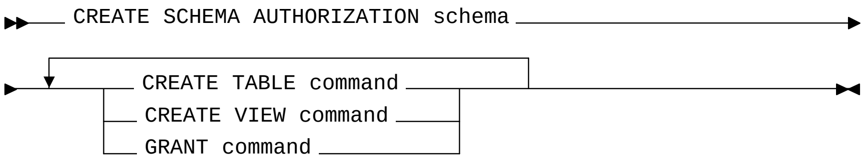
Purpose

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

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.

Syntax



Keyword and Parameters

schema is the name of the schema. The schema name must be the same as your Oracle7 username.

CREATE TABLE *command*
is a CREATE TABLE statement to be issued as part of this CREATE SCHEMA statement. See the CREATE TABLE command on page 4 – 245.

CREATE VIEW *command*
is a CREATE VIEW statement to be issued as part of this CREATE SCHEMA statement. See the CREATE VIEW command on page 4 – 271.

GRANT *command* is a GRANT statement (Objects Privileges) to be issued as part of this CREATE SCHEMA statement. See the GRANT command on page 4 – 355.

The CREATE SCHEMA statement only supports the syntax of these commands as defined by standard SQL, rather than the complete syntax supported by Oracle7. For information on which parts of the syntax for these commands are standard SQL and which are Oracle7 extensions, see Appendix B of this manual.

Usage Notes

With the CREATE SCHEMA command, you can issue multiple Data Definition Language statements in a single transaction. To execute a CREATE SCHEMA statement, Oracle7 executes each included statement. If all statements execute successfully, Oracle7 commits the transaction. If any statement results in an error, Oracle7 rolls back all the statements.

Terminate a CREATE SCHEMA statement just as you would any other SQL statement using the terminator character specific to your tool. For example, if you issue a CREATE SCHEMA statement in SQL*Plus or Server Manager, terminate the statement with a semicolon (;). Do not separate the individual statements within a CREATE SCHEMA statement with the terminator character.

The order in which you list the CREATE TABLE, CREATE VIEW, and GRANT statements is unimportant:

- A CREATE VIEW statement can create a view that is based on a table that is created by a later CREATE TABLE statement.
- A CREATE TABLE statement can create a table with a foreign key that depends on the primary key of a table that is created by a later CREATE TABLE statement.
- A GRANT statement can grant privileges on a table or view that is created by a later CREATE TABLE or CREATE VIEW statement.

The statements within a CREATE SCHEMA statement can also reference existing objects:

- A CREATE VIEW statement can create a view on a table that existed before the CREATE SCHEMA statement.
- A GRANT statement can grant privileges on a previously existing object.

PARALLEL Clause Syntax

The syntax of the PARALLEL clause is allowed for a CREATE TABLE, INDEX, or CLUSTER, when used in CREATE SCHEMA, but parallelism is **not** used when creating the objects.

Example The following statement creates a schema named BLAIR for the user BLAIR:

```
CREATE SCHEMA AUTHORIZATION blair
   CREATE TABLE sox
      (color VARCHAR2(10)  PRIMARY KEY, quantity NUMBER)
   CREATE VIEW red_sox
      AS SELECT color, quantity FROM sox WHERE color = 'RED'
   GRANT select ON red_sox TO waites
```

The following statement creates the table SOX, creates the view RED_SOX, and grants SELECT privilege on the RED_SOX view to the user WAITES.

Related Topics

CREATE TABLE command on 4 – 245
CREATE VIEW command on 4 – 271
GRANT command on 4 – 346

CREATE SEQUENCE

Purpose

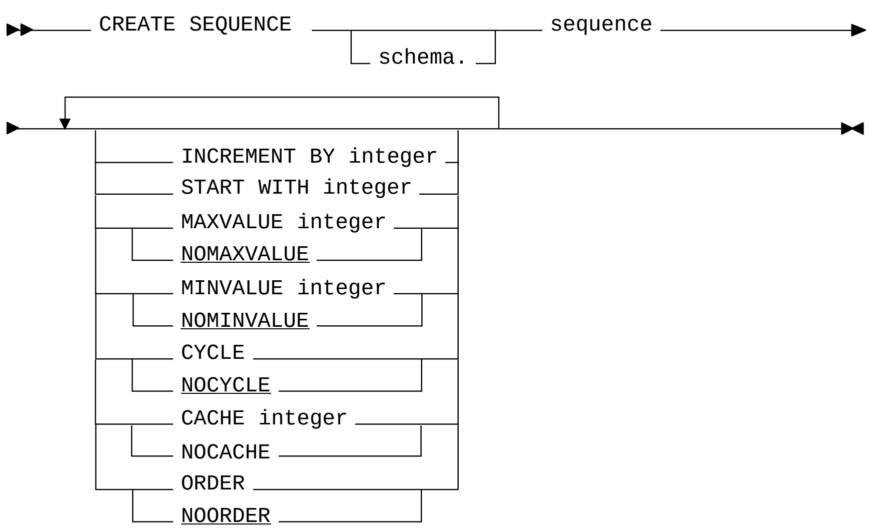
To create a sequence. A *sequence* is a database object from which multiple users may generate unique integers. You can use sequences to automatically generate primary key values.

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 Oracle7 in DBMS MAC mode, your DBMS label must dominate the creation label of the owner of the schema to contain the sequence.

Syntax



Keywords and Parameters

schema is the schema to contain the sequence. If you omit *schema*, Oracle7 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 integer value can be any positive or negative integer, but it cannot be 0. This value can have 28 or less digits. The absolute of this value must be less than the difference of MAXVALUE and MINVALUE. 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. This integer value can have 28 or less digits. MINVALUE must be less than or equal to START WITH and must be less than MAXVALUE.
NOMINVALUE	specifies a minimum value of 1 for an ascending sequence or $-(10^{26})$ for a descending sequence. The default is NOMINVALUE.
MAXVALUE	specifies the maximum value the sequence can generate. This integer value can have 28 or less digits. MAXVALUE must be equal to or less than START WITH and must be greater than MINVALUE.
NOMAXVALUE	specifies a maximum value of 10^{27} for an ascending sequence or –1 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. This integer value can have 28 or less digits.
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 Oracle7 pre–allocates and keeps in memory for faster access. This integer value can have 28 or less digits. 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. You cannot cache more values than will fit in a given cycle of sequence numbers; thus, the maximum value allowed for CACHE must be less than the value determined by the following formula: `(CEIL (MAXVALUE–MINVALUE)) / ABS(INCREMENT)`
NOCACHE	specifies that values of the sequence are not pre–allocated. If you omit both the CACHE parameter and the NOCACHE option, Oracle7 caches 20 sequence numbers by default. However, if you are using Oracle7 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, Oracle7 chooses NOORDER by default. Note that the ORDER option is only necessary to guarantee ordered generation if you are using Oracle7 with the Parallel Server option in parallel mode. If you are using exclusive mode, sequence numbers are always generated in order.

Usage Notes

If you are using Trusted Oracle7, the new sequence is automatically labeled with your DBMS label.

Using Sequences

You can use sequence numbers to automatically generate unique primary key values for your data, and you can also coordinate the keys across multiple rows or tables.

Values for a given sequence are automatically generated by special Oracle7 routines and, consequently, sequences avoid the performance bottleneck which results from implementation of sequences at the application level. For example, one common application–level implementation is to force each transaction to lock a sequence number table, increment the sequence, and then release the table. Under this implementation, only one sequence number may be generated at a time. In contrast, Oracle7 sequences permit the simultaneous generation of multiple sequence numbers while guaranteeing that every sequence number is unique.

When a sequence number is generated, the sequence is incremented, independent of the transaction committing or rolling back. If two users concurrently increment the same sequence, the sequence numbers each user acquires may have gaps because sequence numbers are being generated by the other user. One user can never acquire the sequence number generated by another user. Once a sequence value is generated by one user, that user can continue to access that value regardless of whether the sequence is incremented by another user.

Because sequence numbers are generated independently of tables, the same sequence can be used for one or for multiple tables. It is possible that individual sequence numbers will appear to be skipped, because they were generated and used in a transaction that ultimately rolled back. Additionally, a single user may not realize that other users are drawing from the same sequence.

Sequence Defaults

The sequence defaults are designed so that if you specify none of the clauses, you create an ascending sequence that starts with 1 and increases by 1 with no upper limit. Specifying only INCREMENT BY –1 creates a descending sequence that starts with –1 and decreases with no lower limit.

Incrementing Sequence Values

You can create a sequence so that its values increment in one of following ways:

- The sequence values increment without bound.
- The sequence values increment to a predefined limit and then stop.
- The sequence values increment to a predefined limit and then restart.

To create a sequence that increments without bound, omit the MAXVALUE parameter or specify the NOMAXVALUE option for ascending sequences or omit the MINVALUE parameter or specify the NOMINVALUE for descending sequences.

To create a sequence that stops at a predefined limit, specify a value for the MAXVALUE parameter for an ascending sequence or a value for the MINVALUE parameter for a descending sequence. Also specify the NOCYCLE option. Any attempt to generate a sequence number once the sequence has reached its limit results in an error.

To create a sequence that restarts after reaching a predefined limit, specify values for both the MAXVALUE and MINVALUE parameters. Also specify the CYCLE option. If you do not specify MINVALUE, then it defaults to NOMINVALUE; that is, the value 1.

The value of the START WITH parameter establishes the initial value generated after the sequence is created. Note that this value is not necessarily the value to which an ascending cycling sequence cycles after reaching its maximum or minimum value.

Caching Sequence Numbers

The number of values cached in memory for a sequence is specified by the value of the sequence's CACHE parameter. Cached sequences allow faster generation of sequence numbers. A cache for a given sequence is populated at the first request for a number from that sequence. The cache is repopulated every CACHE requests. If there is a system failure, all cached sequence values that have not been used in committed Data Manipulation Language statements are lost. The potential number of lost values is equal to the value of the CACHE parameter.

A CACHE of 20 future sequence numbers is the default.

Accessing and Incrementing Sequence Values

Once a sequence is created, you can access its values in SQL statements with the following pseudocolumns:

CURRVAL	returns the current value of the sequence.
NEXTVAL	increments the sequence and returns the new value.

For more information on using the above pseudocolumns, see the section "Pseudocolumns" beginning on page 2 – 38.

Example

The following statement creates the sequence ESEQ:

```
CREATE SEQUENCE eseq
   INCREMENT BY 10
```

The first reference to ESEQ.NEXTVAL returns 1. The second returns 11. Each subsequent reference will return a value 10 greater than the one previous.

Related Topics

ALTER SEQUENCE command on 4 – 53
DROP SEQUENCE command on 4 – 314

CREATE SNAPSHOT

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.

Prerequisites

The following prerequisites apply to creating snapshots:

- The distributed option must be installed.
- To create a snapshot in your own schema, you must have the CREATE SNAPSHOT, CREATE TABLE, and CREATE VIEW system privileges, and SELECT privilege on the master tables.
- To create a snapshot in another user's schema, you must have the CREATE ANY SNAPSHOT system privilege, as well as SELECT privilege on the master table. Additionally, the owner of the snapshot must be able to create the snapshot.
- To use updatable snapshots, the replication option must be installed and you must have the CREATE TRIGGER system privilege.

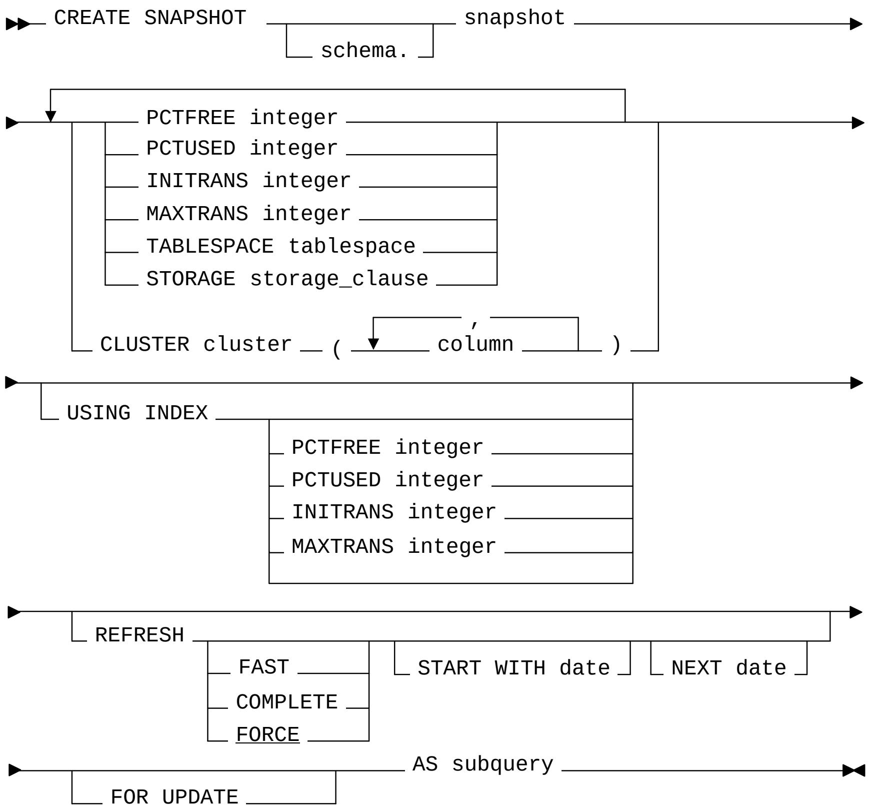
Before a snapshot can be created, the user SYS must run the SQL scripts DBMSSNAP.SQL and PRVTSNAP.PLB on both the database to contain the snapshot and the database(s) containing the tables and views of the snapshot's query. If you have the procedural option, this is done automatically. This script creates the package DBMS_SNAPSHOT which contains the 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, Oracle7 creates a table, two views, and an index in the schema of the snapshot. Oracle7 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 command on 4 – 245, the CREATE VIEW command on 4 – 271, and the CREATE INDEX command on 4 – 192.

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. For information on these privileges, see the SELECT command on page 4 – 405.

To create or refresh a snapshot, Oracle7 must be installed with PL/SQL. To create a snapshot on a remote table or view, Oracle7 must be installed with the distributed option.

Syntax



Keywords and Parameters

schema — is the schema to contain the snapshot. If you omit *schema*, Oracle7 creates the snapshot in your schema.

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

Oracle7 chooses names for the table, views, and index used to maintain the snapshot by adding a prefix and suffix to the snapshot name. To limit these names to 30 bytes and allow them to contain the entire snapshot name, It is recommended that you limit your snapshot names to 19 bytes.

PCTFREE
PCTUSED
INITRANS
MAXTRANS — establishes values for the specified parameters for the internal table Oracle7 uses to maintain the snapshot's data. For information on the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters, see the CREATE TABLE command on

4 – 245. For information on the STORAGE clause, see page 4 – 449.

TABLESPACE — specifies the tablespace in which the snapshot is to be created. If you omit this option, Oracle7 creates the snapshot in the default tablespace of the owner of the snapshot's schema.

STORAGE — establishes storage characteristics for the table Oracle7 uses to maintain the snapshot's data.

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, MAXTRANS, TABLESPACE, or STORAGE parameters with the CLUSTER option.

USING INDEX — specifies parameters for the index Oracle7 creates to maintain the snapshot. You can choose the values of the INITRANS, MAXTRANS, TABLESPACE, STORAGE, and PCTFREE parameters for the index. For information on the PCTFREE, PCTUSED, INITRANS, and MAXTRANS parameters, see the CREATE TABLE command on 4 – 245. For information on the STORAGE clause, see page 4 – 449.

REFRESH — specifies how and when Oracle7 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. Oracle7 decides whether a fast refresh is possible at refresh time.

If you omit the FAST, COMPLETE, and FORCE options, Oracle7 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, Oracle7 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, Oracle7 refreshes the snapshot only once. If you omit both the START WITH and NEXT values or if you omit the REFRESH clause entirely, Oracle7 does not automatically refresh the snapshot.

FOR UPDATE — Allows a simple snapshot to be updated. When used in conjunction with the Replication Option, these updates will be propagated to the master. For more information, see *Oracle7 Server Distributed Systems, Volume II*.

AS *subquery* — specifies the snapshot query. When you create the snapshot, Oracle7 executes this query and places the results in the snapshot. The select list can contain up to 253 expressions. For the syntax of a snapshot query, see the syntax description of *subquery* on page 4 – 436. The syntax of a snapshot query is subject to the same restrictions as a view query. For a list of these restrictions, see the CREATE VIEW command on 4 – 271.

Usage Notes

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. The tables or views in the query are called *master tables*. The databases containing the master tables are called the *master databases.* Note that a snapshot query cannot select from tables or views owned by the user SYS.

Snapshots are useful in distributed databases. Snapshots allow you to maintain read–only copies of remote data on your local node. You can select data from a snapshot as if it were a table or view.

It is recommended that you qualify each table and view in the FROM clause of the snapshot query with the schema containing it.

Snapshots cannot contain long columns.

For more information on snapshots, see *Oracle7 Server Distributed Systems, Volume II*.

Types of Snapshots

You can create the following types of snapshots:

simple — A *simple snapshot* is one in which the snapshot query selects rows from only one master table. This master table must be a table, not a view. Each row of a simple snapshot must be based on a single row of this table. The query for a simple snapshot cannot contain any of the following SQL constructs:

- GROUP BY clause
- CONNECT BY clause
- subqueries
- joins
- set operations

complex — A *complex snapshot* is one in which the snapshot query contains one or more of the constructs not allowed in the query of a simple snapshot. A complex snapshot can be based on multiple master tables on multiple master databases.

Refreshing Snapshots

Because a snapshot's master tables can be modified, the data in a snapshot must occasionally be updated to ensure that the snapshot accurately reflects the data currently in its master tables. The process of updating a snapshot for this purpose is called *refreshing* the snapshot. With the REFRESH clause of the CREATE SNAPSHOT command, you can schedule the times and specify the mode for Oracle7 to automatically refresh the snapshot.

After you create a snapshot, you can subsequently change its automatic refresh mode and time with the REFRESH clause of the ALTER SNAPSHOT command. You can also refresh a snapshot immediately with the DBMS_SNAPSHOT.REFRESH() procedure.

Specifying Refresh Modes

You can use the FAST or COMPLETE options of the REFRESH clause to specify the refresh mode.

Fast To perform a *fast refresh*, Oracle7 updates the snapshot with the changes to the master table recorded in its snapshot log. For more information on snapshot logs, see the CREATE SNAPSHOT LOG command on 4 – 238.

Oracle7 can only perform a fast refresh if all of the following conditions are true:

- The snapshot is a simple snapshot.
- The snapshot's master table has a snapshot log.
- The snapshot log was created before the snapshot was last refreshed or created.

If you specify a fast refresh and all of above conditions are true, then Oracle7 performs a fast refresh. If any of the conditions are not true, Oracle7 returns an error at refresh time and does not refresh the snapshot.

Complete To perform a *complete refresh*, Oracle7 executes the snapshot query and places the results in the snapshot. If you specify a complete refresh, Oracle7 performs a complete refresh regardless of whether a fast refresh is possible.

A fast refresh is often faster than a complete refresh because it sends less data from the master database across the network to the snapshot's database. A fast refresh sends only changes to master table data, while a complete refresh sends the complete result of the snapshot query.

You can also use the FORCE option of the REFRESH clause to allow Oracle7 to decide how to refresh the snapshot at the scheduled refresh time. If a fast refresh is possible based on the fast refresh conditions, then Oracle7 performs a fast refresh. If a fast refresh is not possible, then Oracle7 performs a complete refresh.

Specifying Automatic Refresh Times

To cause Oracle7 to automatically refresh a snapshot, you must perform the following tasks:

1. Specify the START WITH and NEXT parameters in the REFRESH clause of the CREATE SNAPSHOT statement. These parameters establish the time of the first automatic refresh time and the interval between automatic refreshes.
2. Enable one or more snapshot refresh processes using the initialization parameters SNAPSHOT_REFRESH_PROCESSES, SNAPSHOT_REFRESH_INTERVAL, SNAPSHOT_REFRESH_KEEP_CONNECTIONS. The snapshot refresh processes then examine the automatic refresh time of each snapshot in the database. For each snapshot that is scheduled to be refreshed at or before the current time, one of the snapshot refresh processes performs the following operations:
   - re–evaluates the snapshot's NEXT value to determine the next automatic refresh time
   - refreshes the snapshot
   - stores the next automatic refresh time in the data dictionary

For information, see the "Initialization Parameters" chapter of *Oracle7 Server Reference.*

Example I

The following statement creates the simple snapshot EMP_SF that contains the data from a SCOTT's employee table in New York:

```
CREATE SNAPSHOT emp_sf
    PCTFREE 5 PCTUSED 60
    TABLESPACE users
    STORAGE INITIAL 50K NEXT 50K
    REFRESH FAST NEXT sysdate + 7
    AS
        SELECT * FROM scott.emp@ny
```

Since the statement does not include a START WITH parameter, Oracle7 determines the first automatic refresh time by evaluating the NEXT value using the current SYSDATE. Provided a snapshot log currently exists for the employee table in New York, Oracle7 performs a fast refresh of the snapshot every 7 days, beginning 7 days after the snapshot is created.

The above statement also establishes storage characteristics for the table that Oracle7 uses to maintain the snapshot.

Example II The following statement creates the complex snapshot ALL_EMPS that queries the employee tables in Dallas and Baltimore:

```
CREATE SNAPSHOT all_emps
    PCTFREE 5 PCTUSED 60
    TABLESPACE users
    STORAGE INITIAL 50K NEXT 50K
    USING INDEX STORAGE (INITIAL 25K NEXT 25K)
    REFRESH START WITH ROUND(SYSDATE + 1) + 11/24
            NEXT NEXT_DAY(TRUNC(SYSDATE, 'MONDAY') + 15/24
    AS
        SELECT * FROM fran.emp@dallas
            UNION
        SELECT * FROM marco.emp@balt
```

Oracle7 automatically refreshes this snapshot tomorrow at 11:00am. and subsequently every Monday at 3:00pm. Since this command does not specify either fast or complete refreshes, Oracle7 must decide how to refresh the snapshot. Since ALL_EMPS is a complex snapshot, Oracle7 must perform a complete refresh.

The above statement also establishes storage characteristics for both the table and the index that Oracle7 uses to maintain the snapshot:

- The first STORAGE clause establishes the sizes of the first and second extents of the table as 50 kilobytes each.
- The second STORAGE clause (appearing with the USING INDEX option) establishes the sizes of the first and second extents of the index as 25 kilobytes each.

Related Topics

ALTER SNAPSHOT command on 4 – 71
CREATE SNAPSHOT LOG command on 4 – 238
DROP SNAPSHOT command on 4 – 315

CREATE SNAPSHOT LOG

Purpose

To create a snapshot log. A *snapshot log* is a table associated with the master table of a snapshot. Oracle7 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.

Prerequisites

The privileges required to create a snapshot log directly relate to the privileges necessary to create the underlying objects associated with a snapshot log. For example, 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 on 4 – 245.

If you own the master table, you can create an associated snapshot log if you have the CREATE TABLE and CREATE TRIGGER system privileges. If you are creating a snapshot log for a table in another user's schema, you must have the CREATE ANY TABLE and CREATE ANY TRIGGER system privileges. In either case, the owner of the snapshot log must have sufficient quota in the tablespace intended to hold the snapshot log.

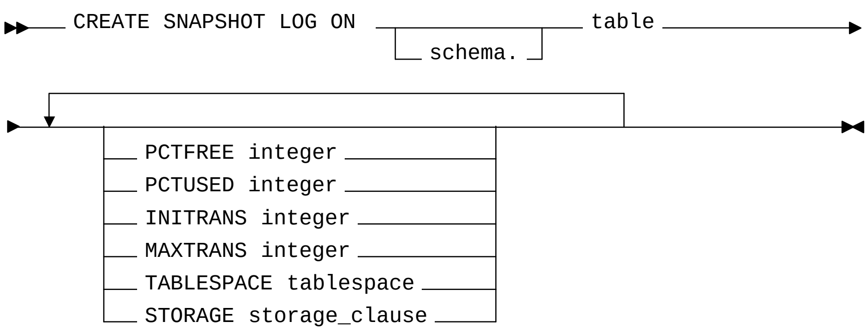
Before a snapshot log can be created, the user SYS must run the SQL scripts DBMSSNAP.SQL and PRVTSNAP.PLB on the database containing the master table.. If you have the procedural option, this is done automatically This script creates the package DBMS_SNAPSHOT, which contains the stored procedures used for refreshing the snapshot and for purging 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 on page 4 – 257.

To create a snapshot log, you must be using Oracle7 with PL/SQL installed.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must dominate the label of the tablespace in which the snapshot log is to be stored.

Syntax



Keywords and Parameters

schema is the schema containing the snapshot log's master table. If you omit *schema,* Oracle7 assumes the master table is contained in your own schema. Oracle7 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.

Oracle7 chooses names for the table and trigger used to maintain the snapshot log by prefixing and suffixing the master table name. To limit these names to 30 bytes and allow them to contain the entire master table name, It is recommended that you limit master table names to 20 bytes.

PCTFREE
PCTUSED
INITRANS
MAXTRANS establishes values for the specified parameters for the snapshot log. See the descriptions of these parameters in the CREATE TABLE command on page 4 – 245.

TABLESPACE specifies the tablespace in which the snapshot log is to be created. If you omit this option, Oracle7 creates the snapshot log in the default tablespace the owner of the snapshot log's schema.

STORAGE establishes storage characteristics for the snapshot log. See the STORAGE clause on page 4 – 449.

Usage Notes

If you are using Trusted Oracle7, the new snapshot log is automatically labeled with your DBMS label.

Using Snapshot Logs

A *snapshot log* is a table that is associated with the master table of a snapshot. When changes are made to the master table's data, Oracle7 adds rows describing these changes to the snapshot log. Later Oracle7 can use these rows to refresh snapshots based on the master table. This process is called a *fast refresh*. Without a snapshot log, Oracle7 must execute the snapshot query to refresh the snapshot. This process is called a *complete refresh*. Usually, a fast refresh takes less time than a complete refresh.

A snapshot log is located in the master database in the same schema as the master table. You can create only a single snapshot log for a master table. Oracle7 can use this snapshot log to perform fast refreshes for all simple snapshots based on the master table. Oracle7 records changes in the snapshot log only if there is a simple snapshot based on the master table. For more information on snapshots, including how Oracle7 refreshes snapshots, see the CREATE SNAPSHOT command on page 4 – 230 and *Oracle7 Server Distributed Systems, Volume II*.

Example The following statement creates a snapshot log on the employee table:

```
CREATE SNAPSHOT LOG ON emp
    PCTFREE 5
    TABLESPACE users
    STORAGE (INITIAL 10K NEXT 10K PCTINCREASE 50)
```

Oracle7 can use this snapshot log to perform a fast refresh on any simple snapshot subsequently created on the EMP table.

Related Topics

ALTER SNAPSHOT LOG command on 4 – 75
CREATE SNAPSHOT command on 4 – 230
DROP SNAPSHOT LOG command on 4 – 316

CREATE SYNONYM

Purpose

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

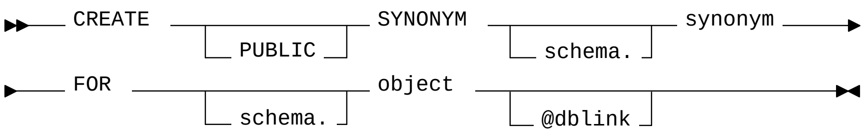
Prerequisites

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 Oracle7 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.

Syntax



Keywords and Parameters

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*, Oracle7 creates the synonym in your own schema. You cannot specify *schema* if you have specified PUBLIC.

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*, Oracle7 assumes that the object is in your own schema. The *object* can be of the following 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. For more information on referring to database links, see the section, "Referring to Objects in Remote Databases," on page 2 – 11. If you specify *dblink* and omit *schema*, the synonym refers to an object in the schema specified by the database link. It is recommended that you specify the schema containing the object in the remote database.

If you omit *dblink*, Oracle7 assumes the object is located on the local database.

Usage Notes

In Trusted Oracle7, the new synonym is automatically labeled with your DBMS label.

A synonym can be used to stand for its base object in any of the following Data Manipulation Language statements:

- SELECT
- INSERT
- UPDATE
- DELETE
- EXPLAIN PLAN
- LOCK TABLE

Synonyms can also be used in the following Data Definition Language statements:

- AUDIT
- NOAUDIT
- GRANT
- REVOKE
- COMMENT

Synonyms are used for security and convenience. Creating a synonym for an object allows you to:

- reference the object without specifying its owner
- reference the object without specifying the database on which it is located
- provide another name for the object

Synonyms provide both data independence and location transparency; synonyms permit applications to function without modification regardless of which user owns the table or view and regardless of which database holds the table or view.

Scope of Synonyms

A private synonym name must be distinct from all other objects in its schema. Oracle7 attempts to resolve references to objects at the schema level before resolving them at the PUBLIC synonym level. Oracle7 only uses a public synonym when resolving references to an object if both of the following cases are true:

- the object is not prefaced by a schema
- the object is not followed by a database link

For example, assume the schemas SCOTT and BLAKE each contain tables named DEPT and the user SYSTEM creates a PUBLIC synonym named DEPT for BLAKE.DEPT. If the user SCOTT then issues the following statement, Oracle7 returns rows from SCOTT.DEPT:

```
SELECT *
    FROM dept
```

To retrieve rows from BLAKE.DEPT, the user SCOTT must preface DEPT with the schema name:

```
SELECT *
    FROM blake.dept
```

If the user ADAM's schema does not contain an object named DEPT, then ADAM can access the DEPT table in BLAKE's schema by using the public synonym DEPT:

```
SELECT *
    FROM dept
```

Example I

To define the synonym MARKET for the table MARKET_RESEARCH in the schema SCOTT, issue the following statement:

```
CREATE SYNONYM market
    FOR scott.market_research
```

Example II

To create a PUBLIC synonym for the EMP table in the schema SCOTT on the remote SALES database, you could issue the following statement:

```
CREATE PUBLIC SYNONYM emp
    FOR scott.emp@sales
```

Note that a synonym may have the same name as the base table provided the base table is contained in another schema.

Related Topics

CREATE DATABASE LINK command on 4 – 185
CREATE TABLE command on 4 – 245
CREATE VIEW command 4 – 271

CREATE TABLE

Purpose

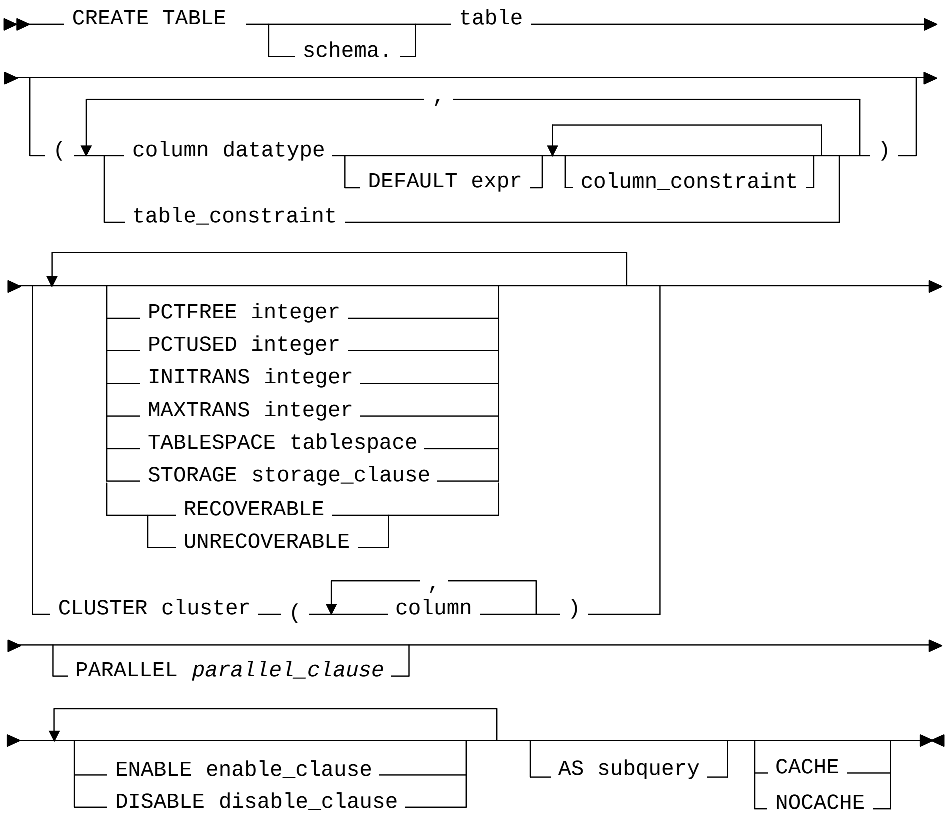
To create a *table*, the basic structure to hold user data, specifying the following information:

- column definitions
- integrity constraints
- the table's tablespace
- storage characteristics
- an optional cluster
- data from an arbitrary query
- degree of parallelism used to create the table and the default degree of parallelism for queries on the table

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.

Syntax



Keywords and Parameters

schema is the schema to contain the table. If you omit *schema*, Oracle7 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. A table can have up to 254 columns. You may only omit column definitions when using the AS *subquery* clause.

datatype is the datatype of a column. Datatypes are defined on page 2 – 18.

You can omit the datatype only if the statement also designates the column as part of a foreign key in a referential integrity constraint. Oracle7 automatically assigns the column the datatype of the corresponding column of the referenced key of the referential integrity constraint.

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. The column must also be long enough to hold this expression. For the syntax of *expr*, see page 3 – 73. 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. See the syntax description of *column_constraint* on page 4 – 152.

table_constraint
: defines an integrity constraint as part of the table definition. See the syntax description of *table_constraint* on page 4 – 152.

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 value from 0 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 Oracle7 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. For information on the performance effects of different values PCTUSED and PCTFREE, see *Oracle7 Server Tuning*.

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, Oracle7 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 Oracle7 creates the table. If you omit this option, then Oracle7 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. See the STORAGE clause on page 4 – 449.

RECOVERABLE specifies that the creation of the table (and any indices required because of constraints) will be logged in the redo log file. This is the default.

If the database is run in ARCHIVELOG mode, media recovery from a backup will recreate the table (and any indices required because of constraints). You cannot specify RECOVERABLE when using NOARCHIVELOG mode.

UNRECOVERABLE

specifies that the creation of the table (and any indices required because of constraints) will not be logged in the redo log file. As a result, media recovery will not recreate the table (and any indices required because of constraints).

This keyword can only be specified with the AS subquery clause. Using this keyword makes table creation faster than using the RECOVERABLE option because redo log entries are not written.

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 with the CLUSTER option.

PARALLEL — specifies the degree of parallelism for creating the table and the default degree of parallelism for queries on the table once created. For more information, see the *parallel_clause* on page 4 – 378.

ENABLE — enables an integrity constraint. See the ENABLE clause on page 4 – 326.

DISABLE — disables an integrity constraint. See the DISABLE clause on page 4 – 295.

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, Oracle7 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. See the syntax description of *subquery* on page 4 – 431.

The number of columns in the table must equal the number of expressions in the subquery. The column definitions can only specify column names, default values, and integrity constraints, not datatypes. Oracle7 derives datatypes and lengths from the subquery. Oracle7 also follows the following rules for integrity constraints:

- Oracle7 also automatically defines any NOT NULL constraints on columns in the new table that existed on the corresponding columns of the selected table if the subquery selects the column rather than an expression containing the column.
- A CREATE TABLE statement cannot contain both the AS clause and a referential integrity constraint definition.

- If a CREATE TABLE statement contains both the AS clause and a CONSTRAINT clause or an ENABLE clause with the EXCEPTIONS option, Oracle7 ignores the EXCEPTIONS option. If any rows violate the constraint, Oracle7 does not create the table and returns an error message.

If all expressions in the subquery are columns, rather than expressions, 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.

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

NOCACHE
specifies that the blocks retrieved for this table are placed at the least recently used end of the LRU list in the buffer cache when a full table scan is performed. This is the default behavior.

Usage Notes

Tables are created with no data unless a query is specified. You can add rows to a table with the INSERT command.

After creating a table, you can define additional columns and integrity constraints with the ADD clause of the ALTER TABLE command. You can change the definition of an existing column with the MODIFY clause of the ALTER TABLE command. To modify an integrity constraint, you must drop the constraint and redefine it.

UNRECOVERABLE

Use of this option may significantly reduce the time taken to create large tables. Note that the keyword UNRECOVERABLE must be explicitly specified. For backup and recovery considerations, see *Oracle7 Server Administrator's Guide.*

Example I To define the EMP table owned by SCOTT, you could issue the following statement:

```
CREATE TABLE scott.emp
    (empno     NUMBER        CONSTRAINT pk_emp PRIMARY KEY,
     ename     VARCHAR2(10)  CONSTRAINT nn_ename NOT NULL
                             CONSTRAINT upper_ename
                               CHECK (ename = UPPER(ename)),
     job       VARCHAR2(9),
     mgr       NUMBER        CONSTRAINT fk_mgr
                               REFERENCES scott.emp(empno),
     hiredate  DATE          DEFAULT SYSDATE,
     sal       NUMBER(10,2)  CONSTRAINT ck_sal
                               CHECK (sal > 500),
     comm      NUMBER(9,0)   DEFAULT NULL,
     deptno    NUMBER(2)     CONSTRAINT nn_deptno NOT NULL
                             CONSTRAINT fk_deptno
REFERENCES scott.dept(deptno) )
    PCTFREE 5 PCTUSED 75 ;
```

This table contains 8 columns. For example, the EMPNO column is of datatype NUMBER and has an associated integrity constraint named PK_EMP. The HIRDEDATE column is of datatype DATE and has a default value of SYSDATE.

This table definition specifies a PCTFREE of 5 and a PCTUSED of 75, which is appropriate for a relatively static table. The definition also defines integrity constraints on the columns of the EMP table.

Example II To define the sample table SALGRADE in the HUMAN_RESOURCE tablespace with a small storage and limited allocation potential, issue the following statement:

```
CREATE TABLE salgrade
    ( grade  NUMBER  CONSTRAINT pk_salgrade
                       PRIMARY KEY
                       USING INDEX TABLESPACE users_a,
      losal  NUMBER,
      hisal  NUMBER )
    TABLESPACE human_resource
    STORAGE (INITIAL     6144
             NEXT        6144
             MINEXTENTS     1
             MAXEXTENTS     5
             PCTINCREASE    5);
```

The above statement also defines a PRIMARY KEY constraint on the GRADE column and specifies that the index Oracle7 creates to enforce this constraint is created in the USERS_A tablespace.

For more examples of defining integrity constraints, see the CONSTRAINT clause on page 4 – 152. For examples of enabling and disabling integrity constraints, see the ENABLE and DISABLE clauses on pages 4 – 326 and 4 – 295, respectively.

Example III Assuming you have the parallel query option, then the fastest method to create a table that has the same columns as the EMP table, but only for those employees in department 10, is to issue a command similar to the following:

```
CREATE TABLE emp_tmp
    UNRECOVERABLE
    PARALLEL (DEGREE 3)
    AS SELECT * FROM emp WHERE deptno = 10;
```

The UNRECOVERABLE keyword speeds up table creation because there is no overhead in generating and logging redo information.

Using parallelism speeds up the creation of the table because three processes are used to create the table. After the table is created, querying the table is also faster because the same degree of parallelism is used to access the table.

Related Topics

ALTER TABLE command on 4 – 89
CREATE CLUSTER command on 4 – 164
CREATE INDEX command on 4 – 192
CREATE TABLESPACE command on 4 – 254
DROP TABLE command on 4 – 318
CONSTRAINT clause on 4 – 149
DISABLE clause on 4 – 295
ENABLE clause on 4 – 326
PARALLEL clause on 4 – 378
STORAGE clause on 4 – 449

CREATE TABLESPACE

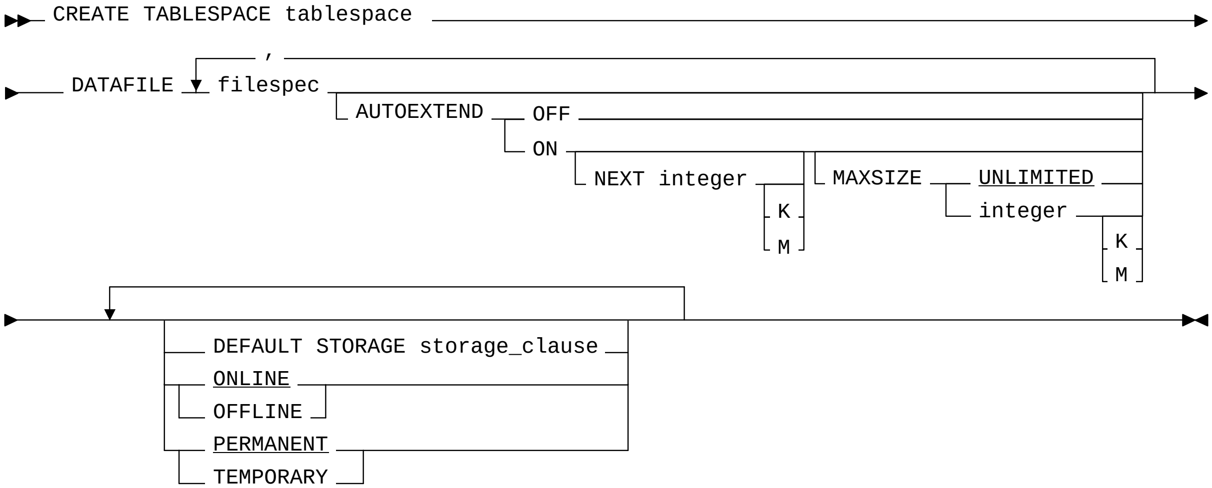
Purpose

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

Prerequisites

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

Syntax



Keywords and Parameters

tablespace — is the name of the tablespace to be created.

DATAFILE — specifies the data file or files to comprise the tablespace. See the syntax description of *filespec* on page 4 – 343.

AUTOEXTEND — enables or disables the automatic extension of datafile.

- OFF — disable autoextend if it is turned on. NEXT and MAXSIZE are set to zero. Values for NEXT and MAXSIZE must be respecified in further ALTER TABLESPACE AUTOEXTEND commands.
- ON — enable autoextend.

NEXT	disk space to allocate to the datafile when more extents are required.
MAXSIZE	maximum disk space allowed for allocation to the datafile.
UNLIMITED	set no limit on allocating disk space to the datafile.

DEFAULT STORAGE

specifies the default storage parameters for all objects created in the tablespace. For information on storage parameters, see the STORAGE clause.

ONLINE

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

OFFLINE

makes the tablespace unavailable immediately after creation.

If you omit both the ONLINE and OFFLINE options, Oracle7 creates the tablespace online by default. The data dictionary view DBA_TABLESPACES indicates whether each tablespace is online or offline.

PERMANENT

specifies that the tablespace will be used to hold permanent objects. This is the default.

TEMPORARY

specifies that the tablespace will only be used to hold temporary objects. For example, segments used by implicit sorts to handle ORDER BY clauses.

Usage Notes

A *tablespace* is an allocation of space in the database that can contain any of the following segments:

- data segments
- index segments
- rollback segments
- temporary segments

All databases have at least one tablespace, SYSTEM, which Oracle7 creates automatically when you create the database.

When you create a tablespace, it is initially a read–write tablespace. After creating the tablespace, you can subsequently use the ALTER TABLESPACE command to take it offline or online, add data files to it, or make it a read–only tablespace.

Many schema objects have associated segments that occupy space in the database. These objects are located in tablespaces. The user creating such an object can optionally specify the tablespace to contain the object. The owner of the schema containing the object must have space quota on the object's tablespace. You can assign space quota on a tablespace to a user with the QUOTA clause of the CREATE USER or ALTER USER commands.

Warning: For operating systems that support raw devices, be aware that the STORAGE clause REUSE keyword has no meaning when specifying a raw device as a datafile in a CREATE TABLESPACE command; such a command will always succeed even if REUSE is **not** specified.

Example I This command creates a tablespace named TABSPACE_2 with one datafile:

```
CREATE TABLESPACE tabspace_2
    DATAFILE 'diska:tabspace_file2.dat' SIZE 20M
    DEFAULT STORAGE (INITIAL 10K NEXT 50K
                     MINEXTENTS 1 MAXEXTENTS 999
                     PCTINCREASE 10)
    ONLINE
```

Example II This command creates a tablespace named TABSPACE_3 with one datafile; when more space is required, 50 kilobyte extents will be added up to a maximum size of 10 megabytes:

```
CREATE TABLESPACE tabspace_3
    DATAFILE 'diskb:tabspace_file3.dat' SIZE 500K REUSE
    AUTOEXTEND ON NEXT 500K MAXSIZX 10M
```

Related Topics

ALTER TABLESPACE command on 4 – 98
DROP TABLESPACE command on 4 – 320

CREATE TRIGGER

Purpose

To create and enable a database trigger. A *database trigger* is a stored PL/SQL block that is associated with a table. Oracle7 automatically executes a trigger when a specified SQL statement is issued against the table.

Prerequisites

OSDoc

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 the following system privileges:

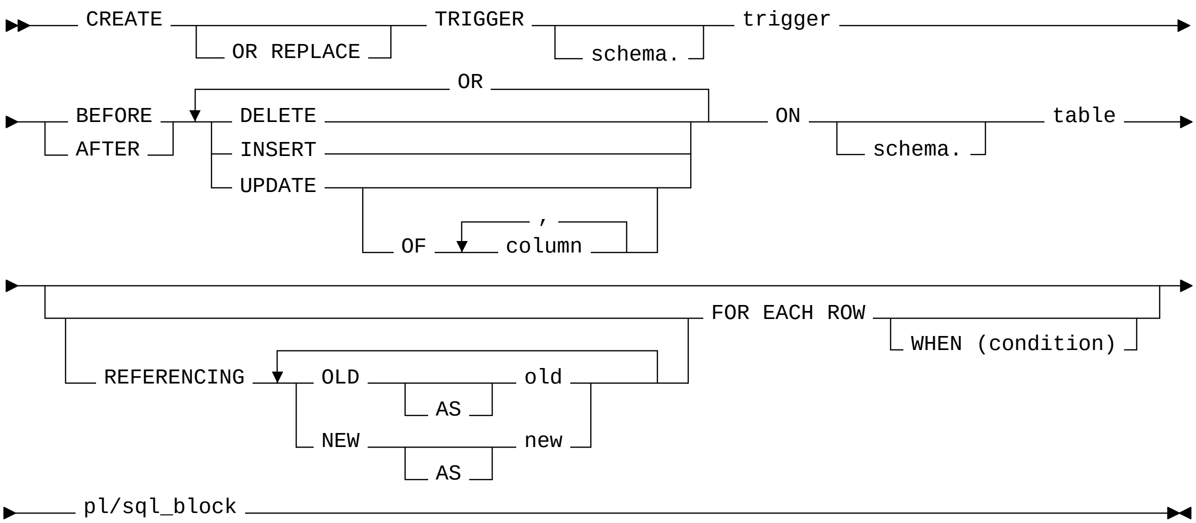
CREATE TRIGGER
: This system privilege allows you to create a trigger in your own schema on a table in your own schema.

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 Oracle7 with PL/SQL installed.

Syntax



Keywords and Parameters

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*, Oracle7 creates the trigger in your own schema.

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

BEFORE
: indicates that Oracle7 fires the trigger before executing the triggering statement. For row triggers, this is a separate firing before each affected row is changed.

AFTER
: indicates that Oracle7 fires the trigger after executing the triggering statement. For row triggers, this is a separate firing after each affected row is changed.

DELETE
: indicates that Oracle7 fires the trigger whenever a DELETE statement removes a row from the table.

INSERT
: indicates that Oracle7 fires the trigger whenever an INSERT statement adds a row to table.

UPDATE OF	indicates that Oracle7 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, Oracle7 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*, Oracle7 assumes the table is in your 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. Oracle7 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. Oracle7 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 Oracle7 to fire the trigger. See the syntax description of *condition* on page 3 – 78. This condition must contain correlation names and cannot contain a query. You can only specify a trigger restriction for a row trigger. Oracle7 evaluates this condition for each row affected by the triggering statement.

pl/sql_block is the PL / SQL block that Oracle7 executes to fire the trigger. For information on PL / SQL, including how to write PL / SQL blocks, see *PL/SQL User's Guide and Reference.*

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

Usage Notes

Before Release 7.3, triggers were parsed and compiled whenever a trigger was fired. From Release 7.3 onwards, the compiled version of a trigger is stored in the data dictionary and is called when a trigger is fired. This feature provides a significant performance improvement to applications that use many triggers.

If a trigger produces compilation errors, it still will be created, but it will fail on execution. This means it effectively blocks all triggering DML statements until it is disabled, replaced by a version without compilation errors, or dropped.

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.

Triggers

A database *trigger* is a stored procedure that is associated with a table. Oracle7 automatically *fires,* or executes, a trigger when a triggering statement is issued.

You can use triggers for the following purposes:

- to provide sophisticated auditing and transparent event logging
- to automatically generate derived column values
- to enforce complex security authorizations and business constraints
- to maintain replicate asynchronous tables

For more information on how to design triggers for the above purposes, see the "Using Database Triggers" chapter of *Oracle7 Server Application Developer's Guide.*

Parts of a Trigger

The syntax of the CREATE TRIGGER statement includes the following parts of the trigger:

Triggering statement The definition of the triggering statement specifies what SQL statements cause Oracle7 to fire the trigger.

DELETE INSERT UPDATE	You must specify at least one of these commands that causes Oracle7 to fire the trigger. You can specify as many as three.
ON	You must also specify the table with which the trigger is associated. The triggering statement is one that modifies this table.

Trigger restriction The trigger restriction specifies an additional condition that must be satisfied for a row trigger to be fired. You can specify this condition with the WHEN clause. This condition must be a SQL condition, rather than a PL/SQL condition.

Trigger action The trigger action specifies the PL/SQL block Oracle7 executes to fire the trigger.

Oracle7 evaluates the condition of the trigger restriction whenever a triggering statement is issued. If this condition is satisfied, then Oracle7 fires the trigger using the trigger action.

Types of Triggers

You can create different types of triggers. The type of a trigger determines the following things:

- when Oracle7 fires the trigger in relation to executing the triggering statement
- how many times Oracle7 fires the trigger

The type of a trigger is based on the use of the following options of the CREATE TRIGGER command:

- BEFORE
- AFTER
- FOR EACH ROW

Using all combinations of the options for the above parts, you can create four basic types of triggers. Table 4 – 10 describes each type of trigger, its properties, and the options used to create it.

		FOR EACH ROW option
BEFORE Option	**BEFORE statement trigger:** Oracle7 fires the trigger once before executing the triggering statement.	**BEFORE row trigger:** Oracle7 fires the trigger before modifying each row affected by the triggering statement.
AFTER Option	**AFTER statement trigger:** Oracle7 fires the trigger once after executing the triggering statement.	**AFTER row trigger:** Oracle7 fires the trigger after modifying each row affected by the triggering statement.

Table 4 – 10 Types of Triggers

For a single table, you can create each type of trigger for each of the following commands:

- DELETE
- INSERT
- UPDATE

You can also create triggers that fire for more than one command.

If you create multiple triggers of the same type that fire for the same command on the same table, the order in which Oracle7 fires these triggers is indeterminate. If your application requires that one trigger be fired before another of the same type for the same command, combine these triggers into a single trigger whose trigger action performs the trigger actions of the original triggers in the appropriate order.

Enabling and Disabling Triggers

An existing trigger must be in one of the following states:

enabled	If a trigger is *enabled*, Oracle7 fires the trigger whenever a triggering statement is issued and the condition of the trigger restriction is met.
disabled	If a trigger is *disabled*, Oracle7 does not fire the trigger when a triggering statement is issued and the condition of the trigger restriction is met.

When you create a trigger, Oracle7 enables it automatically.

You can subsequently disable and enable a trigger with one of the following commands:

- the ALTER TRIGGER command with the DISABLE and ENABLE options
- the ALTER TABLE command with the DISABLE and ENABLE clauses

For information on how to enable and disable triggers, see the ALTER TRIGGER command on page 4 – 105, the ALTER TABLE command on page 4 – 89, the ENABLE clause on page 4 – 326, and the DISABLE clause on page 4 – 295.

Snapshot Log Triggers

When you create a snapshot log for a table, Oracle7 implicitly creates an AFTER ROW trigger on the table. This trigger inserts a row into the snapshot log whenever an INSERT, UPDATE, or DELETE statement modifies the table's data. Since you cannot control the order in which multiple row triggers fire, you shouldn't write triggers intended to affect the content of the snapshot. For more information on snapshot logs, see the CREATE SNAPSHOT LOG command earlier in this chapter.

Example I This example creates a BEFORE statement trigger named EMP_PERMIT_CHANGES in the schema SCOTT. This trigger ensures that changes to employee records are only made during business hours on working days:

```
CREATE TRIGGER scott.emp_permit_changes
    BEFORE
    DELETE OR INSERT OR UPDATE
    ON scott.emp
    DECLARE
        dummy   INTEGER;
    BEGIN
        /* If today is a Saturday or Sunday,

           then return an error.*/
        IF (TO_CHAR(SYSDATE, 'DY') = 'SAT' OR
            TO_CHAR(SYSDATE, 'DY') = 'SUN')
            THEN raise_application_error( –20501,
        'May not change employee table during the weekend');
        END IF;
        /*  Compare today's date with the dates of all
            company holidays. If today is a company holiday,
            then return an error.                           */
        SELECT COUNT(*)
            INTO dummy
            FROM company_holidays
            WHERE day = TRUNC(SYSDATE);
        IF dummy > 0
            THEN raise_application_error( –20501,
            'May not change employee table during a holiday');
        END IF;
        /*  If the current time is before 8:00AM or after
            6:00PM, then return an error.

        */
        IF (TO_CHAR(SYSDATE, 'HH24') < 8 OR
            TO_CHAR(SYSDATE, 'HH24') >= 18)
            THEN raise_application_error( –20502,
        'May only change employee table during working hours');
        END IF;
    END;
```

Oracle7 fires this trigger whenever a DELETE, INSERT, or UPDATE statement affects the EMP table in the schema SCOTT.

Since EMP_PERMIT_CHANGES is a BEFORE statement trigger, Oracle7 fires it once before executing the triggering statement.

The trigger performs the following operations:

1. If the current day is a Saturday or Sunday, the trigger raises an application error with a message that the employee table cannot be changed during weekends.
2. The trigger compares the current date with the dates listed in the table of company holidays.
3. If the current date is a company holiday, the trigger raises an application error with a message that the employee table cannot be changed during holidays.
4. If the current time is not between 8:00AM and 6:00PM, the trigger raises an application error with a message that the employee table can only be changed during business hours.

Example II

This example creates a BEFORE row trigger named SALARY_CHECK in the schema SCOTT. Whenever a new employee is added to the employee table or an existing employee's salary or job is changed, this trigger guarantees that the employee's salary falls within the established salary range for the employee's job:

```
CREATE TRIGGER scott.salary_check
    BEFORE
    INSERT OR UPDATE OF sal, job ON scott.emp
    FOR EACH ROW
    WHEN (new.job <> 'PRESIDENT')
    DECLARE
        minsal          NUMBER;
        maxsal          NUMBER;
    BEGIN
        /* Get the minimum and maximum salaries for the
          employee's job from the SAL_GUIDE table.   */
        SELECT minsal, maxsal
            INTO minsal, maxsal
            FROM sal_guide
            WHERE job = :new.job;
        /* If the employee's salary is below the minimum or  */
        /* above the maximum for the job, then generate an   */
        /* error.                          */
        IF (:new.sal < minsal OR :new.sal > maxsal)
        THEN raise_application_error( -20601,
            'Salary ' || :new.sal || ' out of range for job '
            || :new.job || ' for employee ' || :new.ename );
        END IF;
    END;
```

Oracle7 fires this trigger whenever one of the following statements is issued:

- an INSERT statement that adds rows to the EMP table
- an UPDATE statement that changes values of the SAL or JOB columns of the EMP table

Since SALARY_CHECK is a BEFORE row trigger, Oracle7 fires it before changing each row that is updated by the UPDATE statement or before adding each row that is inserted by the INSERT statement.

SALARY_CHECK has a trigger restriction that prevents it from checking the salary of the company president. For each new or modified employee row that meets this condition, the trigger performs the following steps:

1. The trigger queries the salary guide table for the minimum and maximum salaries for the employee's job.
2. The trigger compares the employee's salary with these minimum and maximum values.
3. If the employee's salary does not fall within the acceptable range, the trigger raises an application error with a message that the employee's salary is not within the established range for the employee's job.

Related Topics

ALTER TRIGGER command on 4 – 105
DROP TRIGGER command on 4 – 322
ENABLE clause on 4 – 326
DISABLE clause on 4 – 295

CREATE USER

Purpose

To create a database *user*, or an account through which you can log in to the database, and establish the means by which Oracle7 permits access by the user. You can optionally assign the following properties to the user:

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

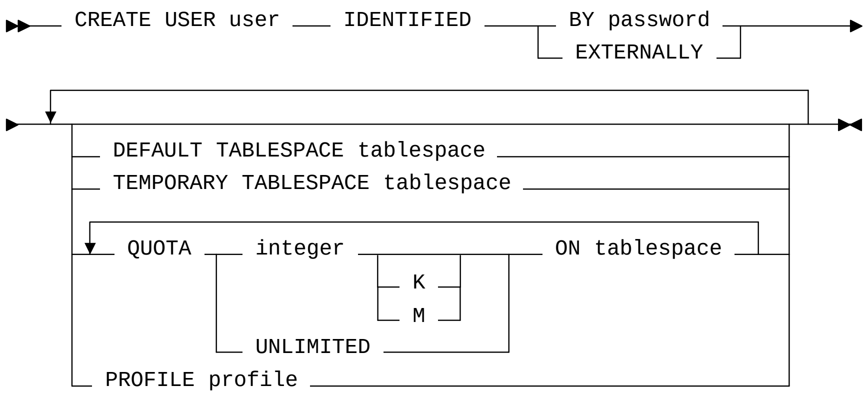
Prerequisites

You must have CREATE USER system privilege.

If you are using Trusted Oracle7 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.

Syntax



Keywords and Parameters

user is the name of the user to be created. This name can only contain characters from your database character set and must follow the rules described in the section "Object Naming on Rule" on page 2 – 3. It is recommended 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 Oracle7 permits user access:

BY *password* The user must specify this password to logon. Password must follow the rules described in the section "Object Naming Rules" on page 2 – 3 and can only contain single–byte characters from your database character set regardless of whether this character set also contains multi–byte characters.

EXTERNALLY Oracle7 verifies that the operating system username matches the database username specified in a database connection.

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.

Note that a CREATE USER command can have multiple QUOTA clauses for multiple tablespaces.

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

PROFILE reassigns the profile named profile to the user. The profile limits the amount of database resources the user can use. If you omit this clause, Oracle7 assigns the DEFAULT profile to the user.

Usage Notes

If you create a new user in Trusted Oracle7, the user's creation label is your DBMS label.

Verifying Users Through Your Operating System

Using CREATE USER ... INDENTIFIED EXTERNALLY allows a database administrator to create a database user that can only be accessed from a specific operating system account. During a database connection, Oracle7 verifies that the operating system username matches the specified database username (prefixed by the value of the initialization parameter OS_AUTHENT_PREFIX). Effectively, you are relying on the login authentication of the operating system to ensure that a specific operating system user has access to a specific database user. Thus, the effective security of such database accounts is dependent entirely on the strength of the operating security mechanisms. For more information, see the *Oracle7 Server Administrator's Guide*.

Oracle Corporation strongly recommends that you do not use IDENTIFIED EXTERNALLY with operating systems that have inherently weak login security.

Establishing Tablespace Quotas for Users

To create an object or a temporary segment, the user must allocate space in some tablespace. To allow the user to allocate space, use the QUOTA clause. A CREATE USER statement can have multiple QUOTA clauses, each for a different tablespace. Other clauses can appear only once.

Note that you need not have a quota on a tablespace to establish a quota for another user on that tablespace.

Granting Privileges to a User

For a user to perform any database operation, the user's privilege domain must contain a privilege that authorizes that operation. A user's privilege domain contains all privileges granted to the user and all privileges in the privilege domains of the user's enabled roles. When you create a user with the CREATE USER command, the user's privilege domain is empty.

> **Note:** To logon to Oracle7, a user must have CREATE SESSION system privilege. After creating a user, you should grant the user this privilege.

Example I You can create the user SIDNEY by issuing the following statement:

```
CREATE USER sidney
    IDENTIFIED BY carton
    DEFAULT TABLESPACE cases_ts
    QUOTA 10M ON cases_ts
    QUOTA 5M ON temp_ts
    QUOTA 5M ON system
    PROFILE engineer
```

The user SIDNEY has the following characteristics:

- the password CARTON
- default tablespace CASES_TS, with a quota of 10 megabytes
- temporary tablespace TEMP_TS, with a quota of 5 megabytes
- access to the tablespace SYSTEM, with a quota of 5 megabytes
- limits on database resources defined by the profile ENGINEER

Example II To create a user accessible only by the operating system account GEORGE, prefix GEORGE by the value of the initialization parameter OS_AUTHENT_PREFIX. For example, if this value is "OPS$", you can create the user OPS$GEORGE with the following statement:

```
CREATE USER ops$george
    IDENTIFIED EXTERNALLY
    DEFAULT TABLESPACE accs_ts
    TEMPORARY TABLESPACE temp_ts
    QUOTA UNLIMITED ON accs_ts
    QUOTA UNLIMITED ON temp_ts
```

The user OPS$GEORGE has the following additional characteristics:

- default tablespace ACCS_TS
- default temporary tablespace TEMP_TS
- unlimited space on the tablespaces ACCS_TS and TEMP_TS
- limits on database resources defined by the DEFAULT profile

Related Topics

ALTER USER command on 4 – 108
CREATE PROFILE command on 4 – 210
CREATE TABLESPACE command 4 – 254
GRANT command on 4 – 346

CREATE VIEW

Purpose

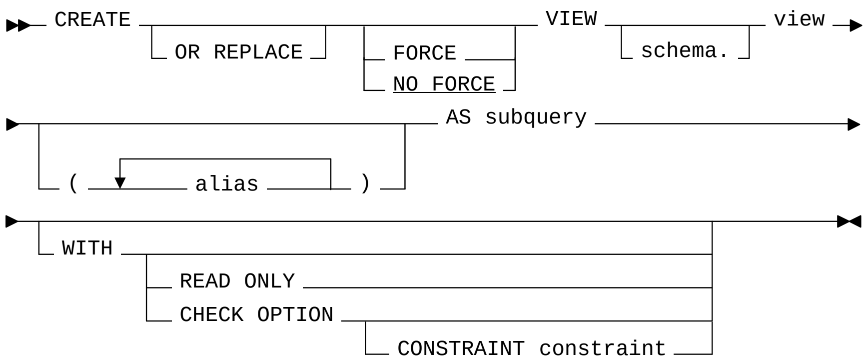
To define a *view*, a logical table based on one or more tables or views.

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 command on page 4 – 405, the INSERT command on page 4 – 361, the UPDATE command on page 4 – 460, and the DELETE command on page 4 – 286. The owner must be granted these privileges directly, rather than through a role.

Syntax



Keywords and Parameters

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*, Oracle7 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 in the section, "Naming Objects and Parts," on page 2 – 3. Aliases must be unique within the view.

If you omit the aliases, Oracle7 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. See the syntax description of *subquery* on page 4 – 436.

WITH READ ONLY

specifies that no deletes, inserts, or updates can be performed through the view.

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, Oracle7 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.

Usage Notes

A *view* is a logical table that allows you to access data from other tables and views. A view contains no data itself. The tables upon which a view is based are called *base tables*.

Views are used for the following purposes:

- To provide an additional level of table security, by restricting access to a predetermined set of rows and / or columns of a base table.
- To hide data complexity. For example, a view may be used to act as one table when actually several tables are used to construct the result.
- To present data from another perspective. For example, views provide a means of renaming columns without actually changing the base table's definition.
- To cause Oracle7 to perform some operations, such as joins, on the database containing the view, rather than another database referenced in the same SQL statement.

You can use a view anywhere you can use a table in any of the following SQL statements:

- COMMENT
- DELETE
- INSERT
- LOCK TABLE
- UPDATE
- SELECT

The View Query

For the syntax of the view's query, see the syntax description of *subquery* on page 4 – 436. Note the following caveats:

- A view's query cannot select the CURRVAL or NEXTVAL pseudocolumns.
- If a view's query selects the ROWID, ROWNUM, or LEVEL pseudocolumns, they must have aliases in the view's query.

- You can define a view with a query that uses an asterisk (*) to select all the columns of a table:

```
CREATE VIEW emp_vu
     AS SELECT * FROM emp
```

  Oracle7 translates the asterisk into a list of all the columns in the table at the time the CREATE VIEW statement is issued. If you subsequently add new columns to the table, the view will not contain these columns unless you recreate the view by issuing another CREATE VIEW statement with the OR REPLACE option. It is recommended that you explicitly specify all columns in the select list of a view query, rather than use the asterisk.

- You can create views that refer to remote tables and views by using database links in the view query. It is recommended that any remote table or view referenced in the view query be qualified with the name of the schema containing it. It is recommended that any database links used in the view query be defined using the CONNECT TO clause of the CREATE DATABASE LINK command.

The above caveats also apply to the query for a snapshot.

If the view query contains any of the following constructs, you cannot perform inserts, updates, or deletes on the view:

- set operators
- group functions
- GROUP BY, CONNECT BY, or START WITH clauses
- the DISTINCT operator

Note that if a view contains pseudocolumns or expressions, you can only update the view with an UPDATE statement that does not refer to any of the pseudocolumns or expressions.

Join Views

A join view is a view with a subquery containing a join. The restrictions described above also apply to join views.

If at least one column in the subquery join has a unique index, then it may be possible to modify one base table in a join view. You can query USER_UPDATABLE_COLUMNS to see whether the columns in a join view are updatable. For example:

```
CREATE VIEW ed AS
    SELECT e.empno, e.ename, d.deptno, d.loc
      FROM emp e, dept d
     WHERE e.deptno = d.deptno

View created.

SELECT column_name, updatable
  FROM user_updatable_columns
 WHERE table_name = 'ED';

COLUMN_NAME     UPD
--------------- ---
ENAME           YES
DEPTNO          NO
EMPNO           YES
LOC             NO
```

In the above example, note that there is a unique index on the DEPTNO column of the DEPT table.

In the above example, you may insert, update or delete a row from the EMP base table because all the columns in the view mapping to the emp table are marked as updatable and because the primary key of emp is included in the view. For more information on updating join views, see "Modifying a Join View" in the *Oracle7 Server Application Developer's Guide*. If there were not null columns in the base EMP table that were not specified in the view subquery, then you could not insert into the table using the view.

Partition Views

A partition view is a view that for performance reasons brings together several tables to behave as one. The effect is as though a single table were divided into multiple tables (partitions) that could be independently accessed. Each partition contains some subset of the values in the view, typically a range of values in some column. Among the advantages of partition views are the following:

- each table in the view is separately indexed, and all indexes can be scanned in parallel.
- if Oracle can tell by the definition of a partition that it can produce no rows to satisfy a query, Oracle will save time by not examining that partition.
- the partitions can be as sophisticated as can be expressed in CHECK constraints.
- if you have the parallel query option, the partitions can be scanned in parallel.

* partitions can overlap.

Among the disadvantages of partition views are the following:

* they cannot be updated.
* they have no master index; rather each component table is separately indexed. For this reason, they are recommended for DSS (Decision Support Systems or "data warehousing") applications, but not for OLTP.

To create a partition view, do the following:

* CREATE the tables that will comprise the view or ALTER existing tables suitably.
* give each table a constraint that limits the values it can hold to the range or other restriction criteria desired.
* create a local index on the constrained column(s) of each table.
* create the partition view as a series of SELECT statements whose outputs are combined using UNION ALL. The view should select all rows and columns from the underlying tables. For more information on SELECT or UNION ALL, see "SELECT" on page 4 – 405.
* if you have the parallel query option enabled, specify that the view is parallel, so that the tables within it are accessed simultaneously when the view is queried. There are two ways to do this:
  – specify "parallel" for each underlying table. For more information on this, see page 4 – 378.
  – place a comment in the SELECT statement that the view contains to give a hint of "parallel" to the Oracle optimizer. For more information on how to do this, see *Oracle7 Server Tuning*.

There is no special syntax required for partition views. Oracle interprets a UNION ALL view of several tables, each of which have local indexes on the same columns, as a partition view. To confirm that Oracle has correctly identified a partition view, examine the output of the EXPLAIN PLAN command. For more information on EXPLAIN PLAN, or on partition views, see *Oracle7 Server Tuning*.

Example I The following statement creates a view of the EMP table named DEPT20. The view shows the employees in department 20 and their annual salary:

```
CREATE VIEW dept20
    AS SELECT ename, sal*12 annual_salary
        FROM emp
        WHERE deptno = 20
```

Note that the view declaration need not define a name for the column based on the expression SAL*12 because the subquery uses a column alias (ANNUAL_SALARY) for this expression.

Example II The following statement creates an updatable view named CLERKS of all clerks in the employee table; only the employees' IDs, names, and department numbers are visible in this view and only these columns can be updated in rows identified as clerks:

```
CREATE VIEW clerk (id_number, person, department, position)
    AS SELECT empno, ename, deptno, job
        FROM emp
        WHERE job = 'CLERK'
    WITH CHECK OPTION CONSTRAINT wco
```

Example III The following statement creates a read only view named CLERKS of all clerks in the employee table; only the employee's IDs, names, and department numbers are visible in this view:

```
CREATE VIEW clerk (id_number, person, department, position)
    AS SELECT empno, ename, deptno, job
        FROM emp
        WHERE job = 'CLERK'
    WITH READ ONLY
```

Because of the CHECK OPTION, you cannot subsequently insert a new row into CLERK if the new employee is not a clerk.

Related Topics

CREATE TABLE command on 4 – 245
CREATE SYNONYM command on 4 – 241
DROP VIEW command on 4 – 325
RENAME command on 4 – 386
SELECT command on 4 – 405

DEALLOCATE clause

Purpose

To specify the amount of unused space to deallocate from extents.

Prerequisites

This clause can only be used in the following commands:

- ALTER CLUSTER
- ALTER TABLE
- ALTER INDEX

Syntax

```
>>— DEALLOCATE UNUSED ————————————————————————————————————————><
                        └ KEEP integer ————┘
                                       ├ K ┤
                                       └ M ┘
```

Keywords and Parameters

KEEP specifies the amount of unused space to keep.

integer the number of bytes to keep. You can also use K or M to specify the size in kilobytes or megabytes.

Usage Notes

For more information on the administration of schema objects, see *Oracle7 Server Administrator's Guide.*

You use the DEALLOCATE clause to reclaim unused space in extents in a cluster, table or index for reuse by other objects in the tablespace. The user quota for the tablespace in which the deallocation occurs is credited by the amount of the released space.

Unused space is deallocated from the end of the object toward the high water mark at the beginning of the object. If an extent is completely contained in the deallocation, then the whole extent is freed for reuse. If an extent is partially contained in the deallocation, then the used part up to the high water mark becomes the extent and the remaining unused space is freed for reuse.

INITIAL, MINEXTENTS and NEXT are described in the STORAGE clause on page 4 – 449.

If you omit the KEEP option and the high water mark is above the size of INITIAL and MINEXTENTS, then all unused space above the high water mark is freed. When the high water mark is less than the size of INITIAL or MINEXTENTS, then all unused space above MINEXTENTS is freed.

If you use the KEEP option, then the specified amount of space is kept and the remaining space is freed. When the remaining number of extents is less than MINEXTENTS, then MINEXTENTS is adjusted to the new number of extents. If the initial extent becomes smaller than INITIAL, then INITIAL is adjusted to the new size.

NEXT is set to the size of the last extent that was deallocated.

Example I The following command frees all unused space for reuse in table EMP, where the high water mark is above MINEXTENTS:

```
ALTER TABLE emp
   DEALLOCATE UNUSED
```

Related Topics

ALTER CLUSTER command on page 4 – 16
ALTER INDEX command on page 4 – 33
ALTER TABLE command on 4 – 89
Chapter "Managing Schema Objects," of *Oracle7 Server Administrator's Guide.*

DECLARE CURSOR (Embedded SQL)

Purpose

To declare a cursor, giving it a name and associating it with a SQL statement or a PL / SQL block.

Prerequisites

If you associate the cursor with an identifier for a SQL statement or PL / SQL block, you must have declared this identifier in a previous DECLARE STATEMENT statement.

Syntax

```
>>— EXEC SQL ———————————————————————————————————————————————————>
            |__ AT ______________ db_name ____________|
                        |__ :host_variable __|
>— DECLARE cursor CURSOR FOR ________ SELECT command ___________><
                               |___ statement_name __|
                               |___ blockname _______|
```

Keywords and Parameters

AT — identifies the database on which the cursor is declared. The database can be identified by either:

db_name — is a database identifier declared in a previous DECLARE DATABASE statement.

:*host_variable* — is a host variable whose value is a previously declared *db_name*.

If you omit this clause, Oracle7 declares the cursor on your default database.

cursor — is the name of the cursor to be declared.

SELECT *command* — is a SELECT statement to be associated with the cursor. The following statement cannot contain an INTO clause.

statement_name
block_name — identifies a SQL statement or PL / SQL block to be associated with the cursor. The *statement_name* or *block_name* must be previously declared in a DECLARE STATEMENT statement.

Usage Notes

You must declare a cursor before referencing it in other embedded SQL statements. The scope of a cursor declaration is global within its precompilation unit and the name of each cursor must be unique in its scope. You cannot declare two cursors with the same name in a single precompilation unit.

You can reference the cursor in the WHERE clause of an UPDATE or DELETE statement using the CURRENT OF syntax, provided that the cursor has been opened with an OPEN statement and positioned on a row with a FETCH statement. For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example

This example illustrates the use of a DECLARE CURSOR:

```
EXEC SQL DECLARE emp_cursor CURSOR
    FOR SELECT ename, empno, job, sal
            FROM emp
            WHERE deptno = :deptno
            FOR UPDATE OF sal
```

Related Topics

CLOSE command on 4 – 139
DECLARE DATABASE command on 4 – 282
DECLARE STATEMENT command on 4 – 283
DELETE command on 4 – 286
FETCH command on 4 – 341
OPEN command on 4 – 376
PREPARE command on 4 – 381
SELECT command on 4 – 405
UPDATE command on 4 – 460

DECLARE DATABASE (Embedded SQL)

Purpose

To declare an identifier for a non–default database to be accessed in subsequent embedded SQL statements.

Prerequisites

You must have access to a username on the non–default database.

Syntax

```
►►─ EXEC SQL DECLARE db_name DATABASE ───────────────────────────►◄
```

Keywords and Parameters

db_name is the identifier established for the non–default database.

Usage Notes

You declare a *db_name* for a non–default database so that other embedded SQL statements can refer to that database using the AT clause. Before issuing a CONNECT statement with an AT clause, you must declare a *db_name* for the non–default database with a DECLARE DATABASE statement.

For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example

This example illustrates the use of a DECLARE DATABASE statement:

```
EXEC SQL DECLARE oracle3 DATABASE
```

Related Topics

COMMIT command on 4 – 141
CONNECT command on 4 – 147
DECLARE CURSOR command on 4 – 280
DECLARE STATEMENT command on 4 – 283
DELETE command on 4 – 286
EXECUTE command on 4 – 332
EXECUTE IMMEDIATE command on 4 – 336
INSERT command on 4 – 361
SELECT command on 4 – 405
UPDATE command on 4 – 460

DECLARE STATEMENT (Embedded SQL)

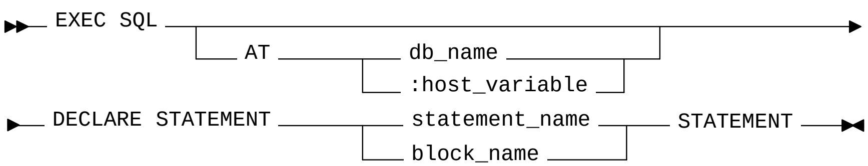
Purpose

To declare an identifier for a SQL statement or PL/SQL block to be used in other embedded SQL statements.

Prerequisites

None.

Syntax



Keywords and Parameters

AT identifies the database on which the SQL statement or PL/SQL block is declared. The database can be identified by either:

db_name is a database identifier declared in a previous DECLARE DATABASE statement.

:*host_variable* is a host variable whose value is a previously declared *db_name.*

If you omit this clause, Oracle7 declares the SQL statement or PL/SQL block on your default database.

statement_name
block_name is the declared identifier for the statement.

Usage Notes

You must declare an identifier for a SQL statement or PL/SQL block with a DECLARE STATEMENT statement only if a DECLARE CURSOR statement referencing the identifier appears physically (not logically) in the embedded SQL program before the PREPARE statement that parses the statement or block and associates it with its identifier.

The scope of a statement declaration is global within its precompilation unit, like a cursor declaration. For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example I This example illustrates the use of the DECLARE STATEMENT statement:

```
EXEC SQL AT remote_db
    DECLARE my_statement STATEMENT
EXEC SQL PREPARE my_statement FROM :my_string
EXEC SQL EXECUTE my_statement
```

Example II In this example from a Pro*C embedded SQL program, the DECLARE STATEMENT statement is required because the DECLARE CURSOR statement precedes the PREPARE statement:

```
EXEC SQL DECLARE my_statement STATEMENT;
call prepare_my_statement;
EXEC SQL DECLARE emp_cursor CURSOR FOR my_statement;
...
PROCEDURE prepare_my_statement
BEGIN
    EXEC SQL PREPARE my_statement FROM :my_string;
END;
```

Related Topics

CLOSE command on 4 – 139
DECLARE DATABASE command on 4 – 282
FETCH command on 4 – 341
PREPARE command on 4 – 381
OPEN command on 4 – 376

DECLARE TABLE (Embedded SQL)

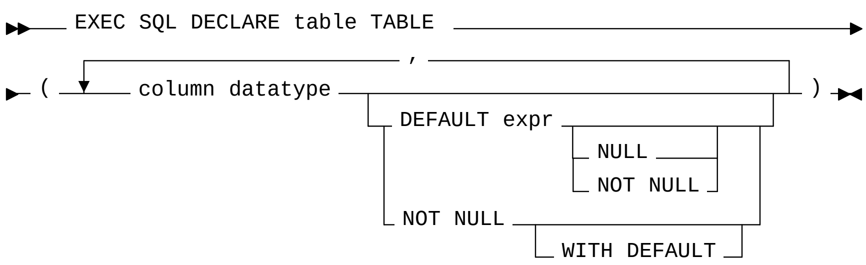
Purpose

To define the structure of a table or view, including each column's datatype, default value, and NULL or NOT NULL specification for semantic checking by the Oracle Precompilers.

Prerequisites

None.

Syntax



Keywords and Parameters

table	is the name of the declared table.
column	is a column of the *table*.
datatype	is the datatype of a *column*. For information on Oracle7 datatypes, see the section "Datatypes" on page 2 – 18.
DEFAULT	specifies the default value of a *column*.
NULL	specifies that a *column* can contain nulls.
NOT NULL	specifies that a *column* cannot contain nulls.
WITH DEFAULT	is supported for compatibility with IBM's DB2 database.

Usage Notes

For information on using this command, see *Programmer's Guide to the Oracle Precompilers.*

Example

The following statement declares the PARTS table with the PARTNO, BIN, and QTY columns:

```
EXEC SQL DECLARE parts TABLE
    (partno  NUMBER  NOT NULL,
     bin     NUMBER,
     qty     NUMBER)
```

Related Topics

None.

DELETE

Purpose

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

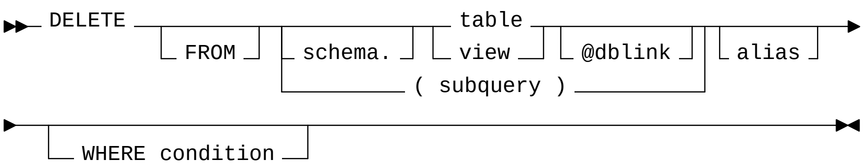
If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must dominate the creation label of the table or view or you must meet one of the following criteria:

- 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 not comparable 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 the following 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema — is the schema containing the table or view. If you omit *schema*, Oracle7 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*, Oracle7 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. For information on referring to database links, see the section "Referring to Objects in Remote Databases" on page 2 – 11. You can only delete rows from a remote table or view if you are using Oracle7 with the distributed option.

If you omit *dblink*, Oracle7 assumes that the table or view is located on the local database.

subquery — is a subquery from which data is selected for deletion. For the syntax of *subquery*, see page 4 – 431. Oracle executes the subquery and then uses the resulting rows as a table in the FROM clause. The subquery cannot query a table that appears in the same FROM clause as the subquery.

alias — is an alias assigned to the table, view or subquery. 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. See the syntax description of *condition* on page 2 – 11. You can only delete rows from a remote table or view if you are using Oracle7 with the distributed option.

If you omit *dblink*, Oracle7 assumes that the table or view is located on the local database.

Usage Notes

All table and index space released by the deleted rows is retained by the table and index. You cannot delete from a view if the view's defining query contains one of the following constructs:

- join
- set operator
- GROUP BY clause
- group function
- DISTINCT operator

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

Example I

The following statement deletes all rows from a table named TEMP_ASSIGN.

```
DELETE FROM temp_assign
```

Example II

The following statement deletes from the employee table all sales staff who made less than $100 commission last month:

```
DELETE FROM emp
    WHERE JOB = 'SALESMAN'
      AND COMM < 100
```

Example III

The following statement has the same effect as in Example II:

```
DELETE FROM (select * from emp)
    WHERE JOB = 'SALESMAN'
      AND COMM < 100
```

Example IV

The following statement deletes all rows from the bank account table owned by the user BLAKE on a database accessible by the database link DALLAS:

```
DELETE FROM blake.accounts@dallas
```

Related Topics

UPDATE command on 4 – 460

DELETE (Embedded SQL)

Purpose

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

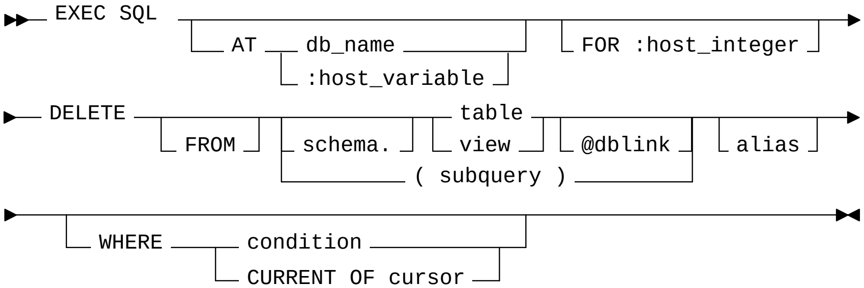
If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must dominate the creation label of the table or view or you must meet one of the following criteria:

- 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 not comparable 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 the following 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

AT — identifies the database to which the DELETE statement is issued. The database can be identified by either:

- *db_name* — is a database identifier declared in a previous DECLARE DATABASE statement.
- is a host variable whose value is a previously declared *db_name*.

If you omit this clause, the DELETE statement is issued to your default database.

FOR :*host_integer* — limits the number of times the statement is executed if the WHERE clause contains array host variables. If you omit this clause, Oracle7 executes the statement once for each component of the smallest array.

schema — is the schema containing the table or view. If you omit *schema*, Oracle7 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*, Oracle7 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. For information on referring to database links, see the section "Referring to Objects in Remote Databases" on page 2 – 11. You can only delete rows from a remote table or view if you are using Oracle7 with the distributed option.

If you omit *dblink*, Oracle7 assumes that the table or view is located on the local database.

subquery — is a subquery from which data is selected for deletion. For the syntax of subquery, see page 4 – 431. Oracle executes the subquery and then uses the resulting rows as a table in the FROM clause. The subquery cannot query a table that appears in the same FROM clause as the subquery.

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

WHERE — specifies which rows are deleted:

condition — deletes only rows that satisfy the condition. This condition can contain host variables and optional indicator variables. See the syntax description of *condition* on page 3 – 78.

CURRENT OF — deletes only the row most recently fetched by the *cursor*. The *cursor* cannot be associated with a SELECT statement that performs a join, unless its FOR UPDATE clause specifically locks only one table.

If you omit this clause entirely, Oracle7 deletes all rows from the table or view.

Usage Notes

The host variables in the WHERE clause must be either all scalars or all arrays. If they are scalars, Oracle7 executes the DELETE statement only once. If they are arrays, Oracle7 executes the statement once for each set of array components. Each execution may delete zero, one, or multiple rows.

Array host variables in the WHERE clause can have different sizes. In this case, the number of times Oracle7 executes the statement is determined by the smaller of the following values:

- the size of the smallest array
- the value of the :*host_integer* in the optional FOR clause

If no rows satisfy the condition, no rows are deleted and the SQLCODE returns a NOT_FOUND condition.

The cumulative number of rows deleted is returned through the SQLCA. If the WHERE clause contains array host variables, this value reflects the total number of rows deleted for all components of the array processed by the DELETE statement.

If no rows satisfy the condition, Oracle7 returns an error through the SQLCODE of the SQLCA. If you omit the WHERE clause, Oracle7 raises a warning flag in the 5th component of SQLWARN in the SQLCA. For more information on this command and the SQLCA, see *Programmer's Guide to the Oracle Precompilers.*

You can use comments in a DELETE statement to pass instructions, or *hints*, to the Oracle7 optimizer. The optimizer uses hints to choose an execution plan for the statement. For more information on hints, see *Oracle7 Server Tuning*.

Example This example illustrates the use of the DELETE statement within a Pro*C embedded SQL program:

```
EXEC SQL DELETE FROM emp
   WHERE deptno = :deptno
   AND job = :job; ...
EXEC SQL DECLARE emp_cursor CURSOR
   FOR SELECT empno, comm
      FROM emp;
EXEC SQL OPEN emp_cursor;
EXEC SQL FETCH c1
   INTO :emp_number, :commission;
EXEC SQL DELETE FROM emp
   WHERE CURRENT OF emp_cursor;
```

Related Topics

DECLARE DATABASE command on 4 – 282
DECLARE STATEMENT command on 4 – 283
TRUNCATE command on 4 – 455

DESCRIBE (Embedded SQL)

Purpose

To initialize a descriptor to hold descriptions of host variables for a dynamic SQL statement or PL/SQL block.

Prerequisites

You must have prepared the SQL statement or PL/SQL block in a previous embedded SQL PREPARE statement.

Syntax

```
>>-- EXEC SQL DESCRIBE --+--------------------------+-------------->
                        +-- BIND VARIABLES FOR ----+
                        +-- SELECT LIST FOR -------+

>----+-- statement_name --+-- INTO descriptor ----------------><
     +-- block_name ------+
```

Keywords and Parameters

BIND VARIABLES
: initializes the descriptor to hold information about the input variables for the SQL statement or PL/SQL block.

SELECT LIST
: initializes the descriptor to hold information about the select list of a SELECT statement.

: The default is SELECT LIST FOR.

statement_name
block_name
: identifies a SQL statement or PL/SQL block previously prepared with a PREPARE statement.

descriptor
: is the name of the descriptor to be initialized.

Usage Notes

You must issue a DESCRIBE statement before manipulating the bind or select descriptor within an embedded SQL program.

You cannot describe both input variables and output variables into the same descriptor.

The number of variables found by a DESCRIBE statement is the total number of placeholders in the prepare SQL statement or PL/SQL block, rather than the total number of uniquely named placeholders. For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example This example illustrates the use of the DESCRIBE statement in a Pro*C embedded SQL program:

```
EXEC SQL PREPARE my_statement     FROM :my_string;
EXEC SQL DECLARE emp_cursor
   FOR SELECT empno, ename, sal, comm
         FROM emp
         WHERE deptno = :dept_number
EXEC SQL DESCRIBE BIND VARIABLES FOR my_statement
   INTO bind_descriptor;
EXEC SQL OPEN emp_cursor
   USING bind_descriptor;
EXEC SQL DESCRIBE SELECT LIST FOR my_statement
   INTO select_descriptor;
EXEC SQL FETCH emp_cursor
   INTO select_descriptor;
```

Related Topics

PREPARE command on 4 – 381

DISABLE clause

Purpose

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

- If you disable an integrity constraint, Oracle7 does not enforce it. However, disabled integrity constraints appear in the data dictionary along with enabled integrity constraints.
- If you disable a trigger, Oracle7 does not fire it if its triggering condition is satisfied.

Prerequisites

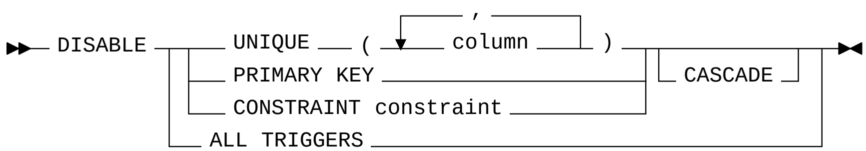
A DISABLE clause that disables an integrity constraint can appear in either a CREATE TABLE or ALTER TABLE command. To disable an integrity constraint, you must have the privileges necessary to issue one of these commands. For information on these privileges, see the CREATE TABLE command on page 4 – 245 and the ALTER TABLE command on page 4 – 89.

For an integrity constraint to appear in a DISABLE clause, one of the following 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 on page 4 – 89. Also, the triggers must be in your own schema or you must have ALTER ANY TRIGGER system privilege.

Syntax



Keywords and Parameters

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.

Usage Notes

You can use the DISABLE clause to disable:

- a single integrity constraint
- all triggers associated with a table

To disable a single trigger, use the DISABLE option of the ALTER TRIGGER command.

How to Disable Integrity Constraints

You can disable an integrity constraint by naming it in a DISABLE clause of either a CREATE TABLE or ALTER TABLE statement. You can define an integrity constraint with a CONSTRAINT clause and disable it with a DISABLE clause together in the same statement. You can also define an integrity constraint in one statement and subsequently disable it in another.

You can also disable an integrity constraint with the DISABLE keyword in the CONSTRAINT clause that defines the integrity constraint. For information on this keyword, see the CONSTRAINT clause on page 4 – 152.

How Oracle7 Disables Integrity Constraints If you disable an integrity constraint, Oracle7 does not enforce it. If you define an integrity constraint and disable it, Oracle7 does not apply it to existing rows of the table, although Oracle7 does store it in the data dictionary along with enabled integrity constraints. Also, Oracle7 can execute Data Manipulation Language statements that change table data and violate a disabled integrity constraint.

If you disable a UNIQUE or PRIMARY KEY constraint that was previously enabled, Oracle7 drops the index that enforces the constraint.

You can enable a disabled integrity constraint with the ENABLE clause.

Disabling Referenced Keys in Referential Integrity Constraints To disable a UNIQUE or PRIMARY KEY constraint that identifies the referenced key of a referential integrity constraint, you must also disable the foreign key. To disable a constraint and all its dependent constraints, use the CASCADE option of the DISABLE clause.

You cannot enable a foreign key that references a unique or primary key that is disabled.

Example I The following statement creates the DEPT table and defines a disabled PRIMARY KEY constraint:

```
CREATE TABLE dept
    (deptno  NUMBER(2)  PRIMARY KEY,
     dname   VARCHAR2(10),
     loc     VARCHAR2(9) )
    DISABLE PRIMARY KEY
```

Since the primary key is disabled, you can add rows to the table that violate the primary key. You can add departments with null department numbers or multiple departments with the same department number.

Example II The following statement defines and disables a CHECK constraint on the EMP table:

```
ALTER TABLE emp
    ADD (CONSTRAINT check_comp  CHECK (sal + comm <= 5000) )
    DISABLE CONSTRAINT check_comp
```

The constraint CHECK_COMP ensures that no employee's total compensation exceeds $5000. Since the constraint is disabled, you can increase an employee's compensation above this limit.

Example III Consider a referential integrity constraint involving a foreign key on the combination of the AREACO and PHONENO columns of the PHONE_CALLS table. The foreign key references a unique key on the combination of the AREACO and PHONENO columns of the CUSTOMERS table. The following statement disables the unique key on the combination of the AREACO and PHONENO columns of the CUSTOMERS table:

```
ALTER TABLE customers
    DISABLE UNIQUE (areaco, phoneno) CASCADE
```

Since the unique key in the CUSTOMERS table is referenced by the foreign key in the PHONE_CALLS table, you must use the CASCADE option to disable the unique key. This option disables the foreign key as well.

How to Disable Triggers

You can disable all triggers associated with the table by using the ALL TRIGGERS option in a DISABLE clause of an ALTER TABLE statement. After you disable a trigger, Oracle7 does not fire the trigger when a triggering statement meets the condition of the trigger restriction.

Example IV

The following statement disables all triggers associated with the EMP table:

```
ALTER TABLE emp
    DISABLE ALL TRIGGERS
```

Related Topics

ALTER TABLE command on 4 – 89
ALTER TRIGGER command on 4 – 105
CONSTRAINT clause on 4 – 149
CREATE TABLE command on 4 – 245
CREATE TRIGGER command on 4 – 257
ENABLE clause on 4 – 326

DROP clause

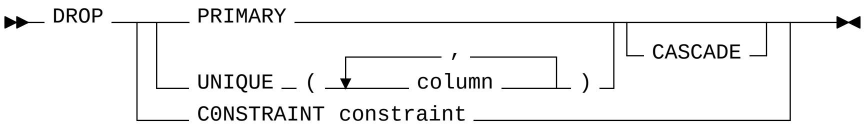
Purpose

To remove an integrity constraint from the database.

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 on page 4 – 89.

Syntax



Keywords and Parameters

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.

Usage Notes

You can drop an integrity constraint by naming it in a DROP clause of an ALTER TABLE statement. When you drop an integrity constraint, Oracle7 stops enforcing the integrity constraint and removes it from the data dictionary.

You cannot drop a unique or primary key that is part of a referential integrity constraint without also dropping the foreign key. You can drop the referenced key and the foreign key together by specifying the referenced key with the CASCADE option in the DROP clause.

Example I

The following statement drops the primary key of the DEPT table:

```
ALTER TABLE dept
    DROP PRIMARY KEY CASCADE
```

If you know that the name of the PRIMARY KEY constraint is PK_DEPT, you could also drop it with the following statement:

```
ALTER TABLE dept
    DROP CONSTRAINT pk_dept CASCADE
```

The CASCADE option drops any foreign keys that reference the primary key.

Example II

The following statement drops the unique key on the DNAME column of the DEPT table:

```
ALTER TABLE dept
    DROP UNIQUE (dname)
```

Note that the DROP clause in this example omits the CASCADE option. Because of this omission, Oracle7 does not drop the unique key if any foreign key references it.

Related Topics

ALTER TABLE command on 4 – 89
CONSTRAINT clause on 4 – 149

DROP CLUSTER

Purpose

To remove a cluster from the database.

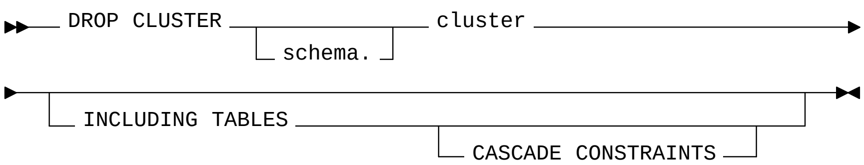
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the cluster's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the cluster. If you omit *schema*, Oracle7 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, Oracle7 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, Oracle7 returns an error message and does not drop the cluster.

Usage Notes

Dropping a cluster also drops the cluster index and returns all cluster space, including data blocks for the index, to the appropriate tablespace(s).

You cannot un–cluster an individual table. To create an un–clustered table identical to an existing clustered table, follow the following steps:

1. Create a new table with the same structure and contents as the old one but with no CLUSTER option.
2. Drop the old table.
3. Use the RENAME command to give the new table the name of the old one.

Grants on the old clustered table do not apply to the new un–clustered table and must be regranted.

Example This command drops a cluster named GEOGRAPHY, all its tables, and any referential integrity constraints that refer to primary or unique keys in those tables:

```
DROP CLUSTER geography
   INCLUDING TABLES
      CASCADE CONSTRAINTS
```

Related Topic

DROP TABLE command on 4 – 318

DROP DATABASE LINK

Purpose

To remove a database link from the database.

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 Oracle7 in DBMS MAC mode, your DBMS label must match the database link's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>— DROP ———————————————— DATABASE LINK dblink ——————————————><
           └— PUBLIC —┘
```

Keywords and Parameters

PUBLIC	must be specified to drop a PUBLIC database link.
dblink	specifies the database link to be dropped.

Usage Notes

You cannot drop a database link in another user's schema and you cannot qualify *dblink* with the name of a schema. Since periods are permitted in names of database links, Oracle7 interprets the entire name, such as RALPH.LINKTOSALES, as the name of a database link in your schema rather than as a database link named LINKTOSALES in the schema RALPH.

Example

The following statement drops a private database link named BOSTON:

```
DROP DATABASE LINK boston
```

Related Topics

CREATE DATABASE LINK command on 4 – 185

DROP FUNCTION

Purpose

To remove a stand–alone stored function from the database.

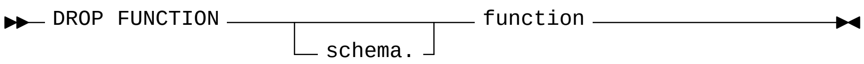
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the function's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the function. If you omit *schema*, Oracle7 assumes the function is in your own schema.

function is the name of the function to be dropped.

Usage Notes

When you drop a function, Oracle7 invalidates any local objects that depend on, or call, the dropped function. If you subsequently reference one of these objects, Oracle7 tries to recompile the object and returns an error message if you have not recreated the dropped function. For more information on how Oracle7 maintains dependencies among schema objects, including remote objects, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

You can only use this command to drop a stand–alone function. To remove a function that is part of a package, use one of the following methods:

- Drop the entire package using the DROP PACKAGE command.
- Redefine the package without the function using the CREATE PACKAGE command with the OR REPLACE option.

Example

The following statement drops the function NEW_ACCT in the schema RIDDLEY:

```
DROP FUNCTION riddley.new_acct
```

When you drop the NEW_ACCT function, Oracle7 invalidates all objects that depend upon NEW_ACCT.

Related Topics

CREATE FUNCTION command on 4 – 188

DROP INDEX

Purpose

To remove an index from the database.

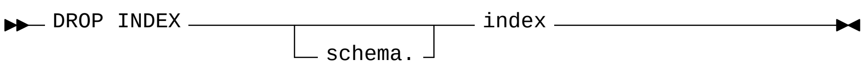
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the index's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema — is the schema containing the index. If you omit *schema*, Oracle7 assumes the index is in your own schema.

index — is the name of the index to be dropped.

Usage Notes

When the index is dropped all data blocks allocated to the index are returned to the index's tablespace.

Example

This command drops an index named MONOLITH:

```
DROP INDEX monolith
```

Related Topics

ALTER INDEX command on 4 – 33

CREATE INDEX command on 4 – 192

CREATE TABLE command on 4 – 245

DROP PACKAGE

Purpose

To remove a stored package from the database.

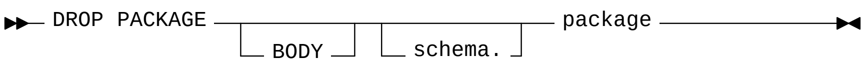
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the cluster's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

BODY — drops only the body of the package. If you omit this option, Oracle7 drops both the body and specification of the package.

schema — is the schema containing the package. If you omit *schema*, Oracle7 assumes the package is in your own schema.

package — is the name of the package to be dropped.

Usage Notes

When you drop the body and specification of a package, Oracle7 invalidates any local objects that depend on the package specification. If you subsequently reference one of these objects, Oracle7 tries to recompile the object and returns an error if you have not recreated the dropped package. For information on how Oracle7 maintains dependencies among schema objects, including remote objects, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

When you drop only the body of a package but not its specification, Oracle7 does not invalidate dependent objects. However, you cannot call one of the procedures or stored functions declared in the package specification until you recreate the package body.

The DROP PACKAGE command drops the package and all its objects together. To remove a single object from a package, you can recreate the package without the object using the CREATE PACKAGE and CREATE PACKAGE BODY commands with the OR REPLACE option.

Example

The following statement drops the specification and body of the BANKING package, invalidating all objects that depend on the specification:

```
DROP PACKAGE banking
```

Related Topics

CREATE PACKAGE command on 4 – 198

DROP PROCEDURE

Purpose

To remove a stand–alone stored procedure from the database.

Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the cluster's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>— DROP PROCEDURE ——┬——————————┬—— procedure ——————————><
                     └ schema. ┘
```

Keywords and Parameters

schema is the schema containing the procedure. If you omit *schema*, Oracle7 assumes the procedure is in your own schema.

procedure is the name of the procedure to be dropped.

Usage Notes

When you drop a procedure, Oracle7 invalidates any local objects that depend upon the dropped procedure. If you subsequently reference one of these objects, Oracle7 tries to recompile the object and returns an error message if you have not recreated the dropped procedure.

For information on how Oracle7 maintains dependencies among schema objects, including remote objects, see the "Dependencies Among Schema Objects" chapter of *Oracle7 Server Concepts*.

You can only use this command to drop a stand–alone procedure. To remove a procedure that is part of a package, use one of the following methods:

- Drop the entire package using the DROP PACKAGE command.
- Redefine the package without the procedure using the CREATE PACKAGE command with the OR REPLACE option.

Example

The following statement drops the procedure TRANSFER owned by the user KERNER:

```
DROP PROCEDURE kerner.transfer
```

When you drop the TRANSFER procedure, Oracle7 invalidates all objects that depend upon TRANSFER.

Related Topics

CREATE PROCEDURE command on 4 – 206

DROP PROFILE

Purpose

To remove a profile from the database.

Prerequisites

You must have DROP PROFILE system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the profile's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>— DROP PROFILE profile ——┬─────────┬——————————————>< 
                           └ CASCADE ┘
```

Keywords and Parameters

profile — is the name of the profile to be dropped.

CASCADE — de–assigns the profile from any users to whom it is assigned. Oracle7 automatically assigns the DEFAULT profile to such users. You must specify this option to drop a profile that is currently assigned to users.

Usage Notes

You cannot drop the DEFAULT profile.

Example

The following statement drops the profile ENGINEER:

```
DROP PROFILE engineer
    CASCADE
```

Oracle7 assigns the DEFAULT profile to any users currently assigned the ENGINEER profile.

Related Topics

CREATE PROFILE command on 4 – 210

DROP ROLE

Purpose

To remove a role from the database.

Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the role's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>— DROP ROLE role ————————————————————————————————><
```

Keywords and Parameters

role is the role to be dropped.

Usage Notes

When you drop a role, Oracle7 revokes it from all users and roles to whom it has been granted and removes it from the database.

Example

To drop the role FLORIST, issue the following statement:

```
DROP ROLE florist
```

Related Topics

CREATE ROLE command on 4 – 215
SET ROLE command on 4 – 442

DROP ROLLBACK SEGMENT

Purpose

To remove a rollback segment from the database.

Prerequisites

You must have DROP ROLLBACK SEGMENT system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the rollback segment's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
>>— DROP ROLLBACK SEGMENT rollback_segment ——————————————><
```

Keywords and Parameters

rollback_segment is the name the rollback segment to be dropped.

Usage Notes

When you drop a rollback segment, all space allocated to the rollback segment returns to the tablespace.

You can only drop a rollback segment that is offline. To determine whether a rollback segment is offline, query the data dictionary view DBA_ROLLBACK_SEGS. Offline rollback segments have the value 'AVAILABLE' in the STATUS column. You can take a rollback segment offline with the OFFLINE option of the ALTER ROLLBACK SEGMENT command.

You cannot drop the SYSTEM rollback segment.

Example

The following statement drops the rollback segment ACCOUNTING:

```
DROP ROLLBACK SEGMENT accounting
```

Related Topics

ALTER ROLLBACK SEGMENT command on 4 – 50
CREATE ROLLBACK SEGMENT command on 4 – 218
CREATE TABLESPACE command on 4 – 254

DROP SEQUENCE

Purpose

To remove a sequence from the database.

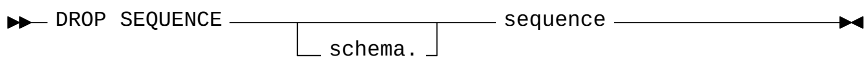
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the sequence's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the sequence. If you omit *schema*, Oracle7 assumes the sequence is in your own schema.

sequence is the name of the sequence to be dropped.

Usage Notes

One method for restarting a sequence is to drop and recreate it. For example, if you have a sequence with a current value of 150 and you would like to restart the sequence with a value of 27, you would:

1. Drop the sequence.
2. Create it with the same name and a START WITH value of 27.

Example

The following statement drops the sequence ESEQ owned by the user ELLY:

```
DROP SEQUENCE elly.eseq
```

To issue the above statement, you must either be connected as the user ELLY or have DROP ANY SEQUENCE system privilege.

Related Topics

ALTER SEQUENCE command on 4 – 53
CREATE SEQUENCE command on 4 – 224

DROP SNAPSHOT

Purpose

To remove a snapshot from the database.

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 Oracle7 uses to maintain the snapshot's data. For information on these privileges, see the DROP TABLE command on page 4 – 318 the DROP VIEW command on page 4 – 325, and the DROP INDEX command on page 4 – 306.

Syntax

```
>>— DROP SNAPSHOT ——————┬——————————┬—— snapshot ———————————————><
                        └─ schema. ┘
```

Keywords and Parameters

schema is the schema containing the snapshot. If you omit *schema*, Oracle7 assumes the snapshot is in your own schema.

snapshot is the name of the snapshot to be dropped.

Usage Notes

When you drop a simple snapshot, if it is the least recently refreshed snapshot of a master table, Oracle7 automatically purges the master table's snapshot log of the rows needed only to refresh the dropped snapshot.

When you drop a master table, Oracle7 does not automatically drop snapshots based on the table. However, Oracle7 returns an error message when it tries to refresh a snapshot based on a master table that has been dropped.

Example

The following statement drops the snapshot PARTS owned by the user HQ:

```
DROP SNAPSHOT hq.parts
```

Related Topics

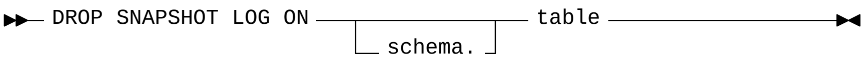
CREATE SNAPSHOT command on 4 – 230

DROP SNAPSHOT LOG

Purpose To remove a snapshot log from the database.

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 later in this chapter. 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 on page 4 – 322.

Syntax



Keywords and Parameters

schema is the schema containing the snapshot log and its master table. If you omit *schema*, Oracle7 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.

Usage Notes After you drop a snapshot log, snapshots based on the snapshot log's master table can no longer be refreshed fast. They must be refreshed completely. For more information on refreshing snapshots, see the CREATE SNAPSHOT command on page 4 – 230.

Example The following statement drops the snapshot log on the PARTS master table:

```
DROP SNAPSHOT LOG ON parts
```

Related Topics CREATE SNAPSHOT LOG command on 4 – 238

DROP SYNONYM

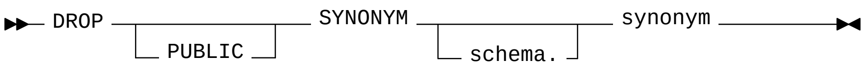
Purpose To remove a synonym from the database.

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 Oracle7 in DBMS MAC mode, your DBMS label must match the synonym's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

PUBLIC — must be specified to drop a public synonym. You cannot specify *schema* if you have specified PUBLIC.

schema — is the schema containing the synonym. If you omit *schema*, Oracle7 assumes the synonym is in your own schema.

synonym — is the name of the synonym to be dropped.

Usage Notes You can change the definition of a synonym by dropping and recreating it.

Example To drop a synonym named MARKET, issue the following statement:

```
DROP SYNONYM market
```

Related Topic CREATE SYNONYM command on 4 – 241

DROP TABLE

Purpose

To remove a table and all its data from the database.

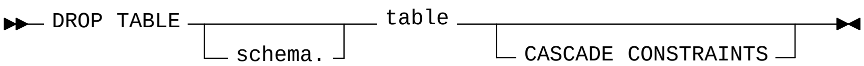
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the table's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the table. If you omit *schema*, Oracle7 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, Oracle7 returns an error message and does not drop the table.

Usage Notes

When you drop a table, Oracle7 also automatically performs the following operations:

- Oracle7 removes all rows from the table (as if the rows were deleted).
- Oracle7 drops all the table's indexes, regardless of who created them or whose schema contains them.
- If the table is not part of a cluster, Oracle7 returns all data blocks allocated to the table and its indexes to the tablespaces containing the table and indexes.
- If the table is a base table for views or if it is referenced in stored procedures, functions, or packages, Oracle7 invalidates these objects but does not drop them. You cannot use these objects unless you recreate the table or drop and recreate the objects so that they no longer depend on the table.

  If you choose to recreate the table, it must contain all the columns selected by the queries originally used to define the views and all the columns referenced in the stored procedures, functions, or packages. Note that any users previously granted object privileges on the views, synonyms, stored procedures, functions, or packages need not be regranted these privileges.
- If the table is a master table for snapshots, Oracle7 does not drop the snapshots. Such a snapshot can still be queried, but it cannot be refreshed unless the table is recreated so that it contains all the columns selected by the snapshot's query.

  If you choose to recreate the table, it must contain all the columns selected by the queries originally used to define the snapshots.
- If the table has a snapshot log, Oracle7 drops the snapshot log.

You can drop a cluster and all of its tables using the DROP CLUSTER command with the INCLUDING TABLES clause and avoid dropping each table individually.

Example The following statement drops the TEST_DATA table:

```
DROP TABLE test_data
```

Related Topics

DROP CLUSTER command on 4 – 301
ALTER TABLE command on 4 – 89
CREATE INDEX command on 4 – 192
CREATE TABLE command on 4 – 245

DROP TABLESPACE

Purpose

To remove a tablespace from the database.

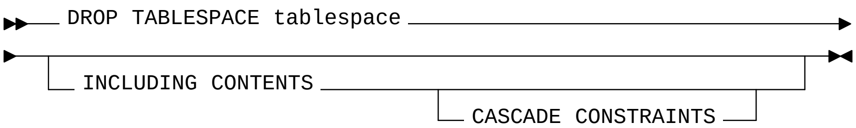
Prerequisites

You must have DROP TABLESPACE system privilege. No rollback segments in the tablespace can be assigned active transactions.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the tablespace's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

tablespace is the name of the tablespace to be dropped.

INCLUDING CONTENTS

drops all the contents 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, Oracle7 returns an error message 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, Oracle7 returns an error message and does not drop the tablespace.

Usage Notes

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

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

You cannot drop the SYSTEM tablespace.

Example

The following statement drops the MFRG tablespace and all its contents:

```
DROP TABLESPACE mfrg
    INCLUDING CONTENTS
        CASCADE CONSTRAINTS
```

Related Topics

ALTER TABLESPACE command on 4 – 98
CREATE DATABASE command on 4 – 178
CREATE TABLESPACE command on 4 – 254

DROP TRIGGER

Purpose

To remove a database trigger from the database.

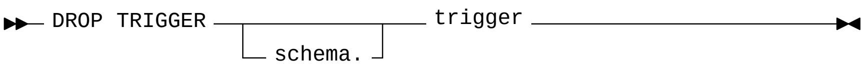
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the trigger's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the trigger. If you omit *schema*, Oracle7 assumes the trigger is in your own schema.

trigger is the name of the trigger to be dropped.

Usage Notes

When you drop a database trigger, Oracle7 removes it from the database and does not fire it again.

Example

The following statement drops the REORDER trigger in the schema RUTH:

```
DROP TRIGGER ruth.reorder
```

Related Topics

CREATE TRIGGER command on 4 – 257

DROP USER

Purpose

To remove a database user and optionally remove the user’s objects.

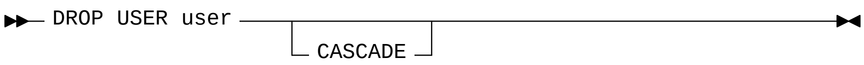
Prerequisites

You must have DROP USER system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the user’s creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

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.

Usage Notes

Oracle7 does not drop users whose schemas contain objects. To drop such a user, you must perform one of the following actions:

- explicitly drop the user's objects before dropping the user
- drop the user and objects together using the CASCADE option

If you specify the CASCADE option and drop tables in the user's schema, Oracle7 also automatically drops any referential integrity constraints on tables in other schemas that refer to primary and unique keys on these tables. The CASCADE option causes Oracle7 to invalidate, but not drop, the following objects in other schemas:

- views or synonyms for objects in the dropped user's schema
- stored procedures, functions, or packages that query objects in the dropped user's schema

Oracle7 does not drop snapshots on tables or views in the user's schema or roles created by the user.

Example I If BRADLEY's schema contains no objects, you can drop BRADLEY by issuing the statement:

```
DROP USER bradley
```

Example II If BRADLEY's schema contains objects, you must use the CASCADE option to drop BRADLEY and the objects:

```
DROP USER bradley CASCADE
```

Related Topics

CREATE USER command on 4 – 267
DROP TABLE command on 4 – 318
DROP TABLESPACE command on 4 – 320
DROP TRIGGER command on 4 – 322
DROP VIEW command on 4 – 325

DROP VIEW

Purpose

To remove a view from the database.

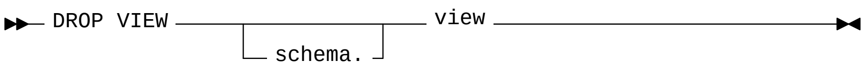
Prerequisites

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

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the view's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the view. If you omit *schema*, Oracle7 assumes the view is in your own schema.

view is the name of the view to be dropped.

Usage Notes

When you drop a view, views and synonyms that refer to the view are not dropped, but become invalid. Drop them or redefine them, or define other views in such a way that the invalid views and synonyms become valid again.

You can change the definition of a view by dropping and recreating it.

Example

The following statement drops the VIEW_DATA view:

```
DROP VIEW view_data
```

Related Topics

CREATE TABLE command on 4 – 245
CREATE VIEW command on 4 – 271
CREATE SYNONYM command on 4 – 241

ENABLE clause

Purpose

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

- If you enable a constraint, Oracle7 enforces it by applying it to all data in the table. All table data must satisfy an enabled constraint.
- If you enable a trigger, Oracle7 fires the trigger whenever its triggering condition is satisfied.

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 command on page 4 – 245 or the ALTER TABLE command on page 4 – 89.

If you enable a UNIQUE or PRIMARY KEY constraint, Oracle7 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 on page 4 – 192.

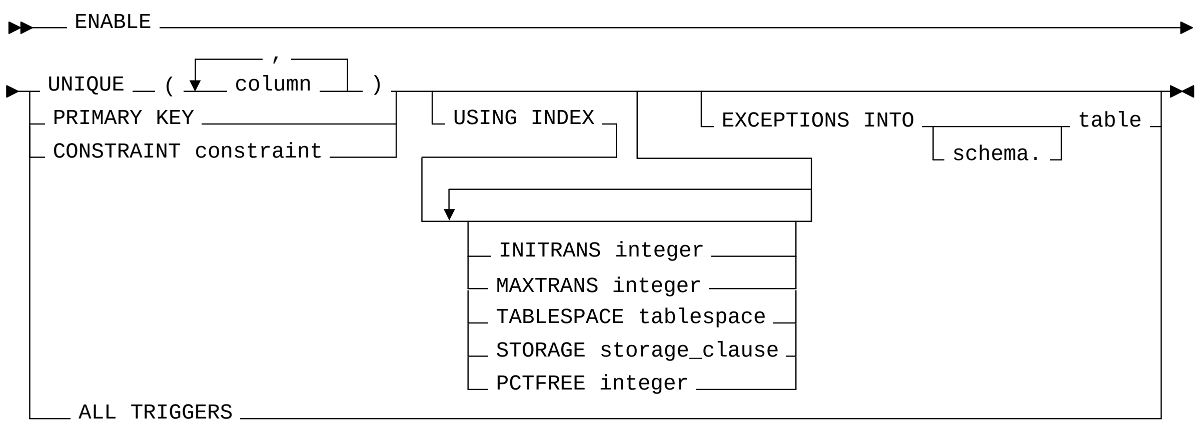
If you enable a referential integrity constraint, the referenced UNIQUE or PRIMARY KEY constraint must already be enabled.

For an integrity constraint to appear in an ENABLE clause, one of the following 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 on page 4 – 89. Also, the triggers must be in your own schema or you must have ALTER ANY TRIGGER system privilege.

Syntax



Keywords and Parameters

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 Oracle7 creates to enforce a UNIQUE or PRIMARY KEY constraint. Oracle7 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 on page 4 – 245.

Only use these parameters when enabling UNIQUE and PRIMARY KEY constraints.

EXCEPTIONS INTO
identifies a table into which Oracle7 places information about rows that violate the integrity constraint. The table must exist before you use this option. If you omit *schema*, Oracle7 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.

Usage Notes

You can use the ENABLE clause to enable either:

- a single integrity constraint
- all triggers associated with a table

To enable a single trigger, use the ENABLE option of the ALTER TRIGGER command.

How to Enable Integrity Constraints

You can enable an integrity constraint by including an ENABLE clause in either a CREATE TABLE or ALTER TABLE statement. You can define an integrity constraint with a CONSTRAINT clause and enable it with an ENABLE clause together in the same statement. You can also define an integrity constraint in one statement and subsequently enable it in another.

You can also enable an integrity constraint by including the ENABLE keyword in CONSTRAINT clause that defines the integrity constraint. For information on this keyword, see the CONSTRAINT clause on page 4 – 149.

If you define an integrity constraint and do not explicitly enable or disable it, Oracle7 enables it by default.

How Oracle7 Enforces Integrity Constraints When you attempt to enable an integrity constraint, Oracle7 applies the integrity constraint to any existing rows in the table:

- If all rows in the table satisfy the integrity constraint, then Oracle7 enables the integrity constraint.
- If any row in the table violates the integrity constraint, the integrity constraint remains disabled. Oracle7 returns an error message indicating the integrity constraint is still disabled.

Once an integrity constraint is enabled, Oracle7 applies the integrity constraint whenever an INSERT, UPDATE, or DELETE statement tries to change table data:

- If the new data satisfies the integrity constraint, then Oracle7 executes the statement.
- If the new data violates the integrity constraint, then Oracle7 does not execute the statement. Instead, Oracle7 generates an error message indicating the integrity constraint violation.

How to Identify Exceptions An *exception* is a row in a table that violates an integrity constraint. You can request that Oracle7 identify exceptions to an integrity constraint. If you specify an exception table in your ENABLE clause, Oracle7 inserts a row into the exception table for each exception. A row of the exception table contains the following information:

- the ROWID of the exception
- the name of the integrity constraint
- the schema and name of the table

A definition of a sample exception table named EXCEPTIONS appears in a SQL script available on your distribution media. Your exception table must have the same column datatypes and lengths as the sample. The common name of this script is UTLEXCPT.SQL, although its exact name and location may vary depending on your operating system. You can request that Oracle7 send exceptions from multiple enabled integrity constraints to the same exception table.

To specify an exception table in an ENABLE clause, you must have the privileges necessary to insert rows into the table. For information on these privileges, see the INSERT command on page 4 – 361. To examine the identified exceptions, you must have the privileges necessary to query the exceptions table. For information on these privileges, see the SELECT command on page 4 – 405.

If a CREATE TABLE statement contains both the AS clause and an ENABLE clause with the EXCEPTIONS option, Oracle7 ignores the EXCEPTIONS option. If there are any exceptions, Oracle7 does not create the table and returns an error message.

Example I The following statement creates the DEPT table and defines and enables a PRIMARY KEY constraint:

```
CREATE TABLE dept
    (deptno  NUMBER(2)      PRIMARY KEY,
     dname   VARCHAR2(10),
     loc     VARCHAR2(9) )
    TABLESPACE user_a
    ENABLE PRIMARY KEY USING INDEX INITRANS 3
                       STORAGE (INITIAL 10K   NEXT 10K
                                MINEXTENTS 2  MAXEXTENTS 10)
                       TABLESPACE user_b
                       PCTFREE 5
```

Oracle7 enforces the PRIMARY KEY constraint with an index. The ENABLE clause specifies INITRANS, STORAGE parameters, TABLESPACE, and PCTFREE values for the data blocks of the index.

Example II The following statement enables an integrity constraint named FK_DEPTNO in the EMP table:

```
ALTER TABLE emp
    ENABLE CONSTRAINT fk_deptno
        EXCEPTIONS INTO except_table
```

Each row of the EMP table must satisfy the constraint for Oracle7 to enable the constraint. If any row violates the constraint, the constraint remains disabled. Oracle7 lists any exceptions in the table EXCEPT_TABLE. You can query this table with the following statement:

```
SELECT *
    FROM except_table
```

The output of this query might look like this:

```
ROW_ID              OWNER  TABLE_NAME  CONSTRAINT
------------------  -----  ----------  ----------
0000346A.0001.0003  SCOTT  EMP         FK_DEPTNO
```

You can also identify the exceptions in the EMP table with the following statement:

```
SELECT emp.*
    FROM emp, except_table
    WHERE emp.row_id except_table.row_id
      AND except_table.table_name = 'EMP'
      AND except_table.constraint = 'FK_DEPTNO'
```

If there are exceptions to the FK_DEPTNO constraint, the output of this query might look like this:

```
EMPNO ENAME    JOB    MGR      HIREDATE   SAL      COMM    DEPTNO
---- -------- ------ -------- ---------- -------- -------- -------
8001 JACK     CLERK       7788 25-AUG-92     1100               70
```

Example III The following statement tries to enable two constraints on the EMP table:

```
ALTER TABLE emp
    ENABLE UNIQUE (ename)
    ENABLE CONSTRAINT nn_ename
```

The preceding statement has two ENABLE clauses:

- The first enables a UNIQUE constraint on the ENAME column.
- The second enables the constraint named NN_ENAME.

In this case, Oracle7 only enables the constraints if both are satisfied by each row in the table. If any row violates either constraint, Oracle7 returns an error message and both constraints remain disabled.

How to Enable Triggers You can enable all triggers associated with the table by including the ALL TRIGGERS option in an ENABLE clause of an ALTER TABLE statement. After you enable a trigger, Oracle7 fires the trigger whenever a triggering statement is issued that meets the condition of the trigger restriction. When you create a trigger, Oracle7 enables it automatically.

Example IV The following statement enables all triggers associated with the EMP table:

```
ALTER TABLE emp
   ENABLE ALL TRIGGERS
```

Related Topics

ALTER TABLE command on 4 – 89
ALTER TRIGGER command on 4 – 105
CONSTRAINT clause on 4 – 149
CREATE TABLE command on 4 – 245
CREATE TRIGGER command on 4 – 257
DISABLE clause on 4 – 295
STORAGE clause on 4 – 449

EXECUTE (Prepared SQL Statements and PL/SQL Blocks) (Embedded SQL)

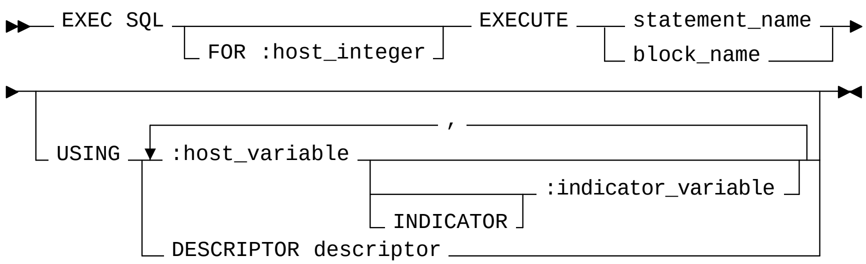
Purpose

To execute a DELETE, INSERT, or UPDATE statement or a PL/SQL block that has been previously prepared with an embedded SQL PREPARE statement.

Prerequisites

You must first prepare the SQL statement or PL/SQL block with an embedded SQL PREPARE statement.

Syntax



Keywords and Parameters

FOR :*host_integer*
limits the number of times the statement is executed when the USING clause contains array host variables If you omit this clause, Oracle7 executes the statement once for each component of the smallest array.

statement_name
block_name
identifies the SQL statement or PL/SQL block to be executed. The SQL statement can only be a DELETE, INSERT, or UPDATE statement. You must use the embedded SQL PREPARE command to associate this identifier with the statement.

USING
specifies a list of host variables with optional indicator variables that Oracle7 substitutes as input variables into the statement to be executed. The host and indicator variables must be either all scalars or all arrays.

Usage Notes

For more information on this command, see the *Programmer's Guide to the Oracle Precompilers.*

Example

This example illustrates the use of the EXECUTE statement in a Pro*C embedded SQL program:

```
EXEC SQL PREPARE my_statement
   FROM :my_string;
EXEC SQL EXECUTE my_statement
   USING :my_var;
```

Related Topics

DECLARE DATABASE command on 4 – 282
PREPARE command on 4 – 381

EXECUTE (Anonymous PL/SQL Blocks) (Embedded SQL)

Purpose

To embed an anonymous PL / SQL block into an Oracle Precompiler program.

Prerequisites

None.

Syntax

```
>>- EXEC SQL ----------------------------- EXECUTE pl/sql_block END-EXEC -><
              |                         |
              +- AT -+- db_name --------+
                     |                  |
                     +- :host_variable -+
```

Keywords and Parameters

AT — identifies the database on which the PL / SQL block is executed. The database can be identified by either:

db_name — is a database identifier declared in a previous DECLARE DATABASE statement.

:*host_variable* — is a host variable whose value is a previously declared *db_name*.

If you omit this clause, the PL / SQL block is executed on your default database.

pl/sql_block — For information on PL / SQL, including how to write PL / SQL blocks, see *PL/SQL User's Guide and Reference.*

END–EXEC — must appear after the embedded PL / SQL block, regardless of which programming language your Oracle Precompiler program uses. Of course, the keyword END–EXEC must be followed by the embedded SQL statement terminator for the specific language.

Usage Notes

Since the Oracle Precompilers treat an embedded PL/SQL block like a single embedded SQL statement, you can embed a PL/SQL block anywhere in an Oracle Precompiler program that you can embed a SQL statement. For more information on embedding PL/SQL blocks in Oracle Precompiler programs, see *Programmer's Guide to the Oracle Precompilers.*

Example

Placing this EXECUTE statement in an Oracle Precompiler program embeds a PL/SQL block in the program:

```
EXEC SQL EXECUTE
   BEGIN
      SELECT ename, job, sal
         INTO :emp_name:ind_name, :job_title, :salary
         FROM emp
         WHERE empno = :emp_number;
      IF :emp_name:ind_name IS NULL
         THEN RAISE name_missing;
      END IF;
   END;
END-EXEC
```

Related Topics

EXECUTE command on 4 – 332
EXECUTE IMMEDIATE embedded SQL command on 4 – 336

EXECUTE IMMEDIATE (Embedded SQL)

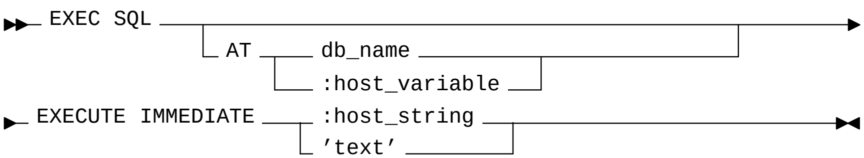
Purpose

To prepare and execute a DELETE, INSERT, or UPDATE statement or a PL/SQL block containing no host variables.

Prerequisites

None.

Syntax



Keywords and Parameters

AT — identifies the database on which the SQL statement or PL/SQL block is executed. The database can be identified by either:

db_name — is a database identifier declared in a previous DECLARE DATABASE statement.

:*host_variable* — is a host variable whose value is a previously declared *db_name*.

If you omit this clause, the statement or block is executed on your default database.

:*host_string* — is a host variable whose value is the SQL statement or PL/SQL block to be executed.

'*text*' — is a quoted text literal containing the SQL statement or PL/SQL block to be executed.

The SQL statement can only be a DELETE, INSERT, or UPDATE statement.

Usage Notes

When you issue an EXECUTE IMMEDIATE statement, Oracle7 parses the specified SQL statement or PL / SQL block, checking for errors, and executes it. If any errors are encountered, they are returned in the SQLCODE component of the SQLCA.

For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example This example illustrates the use of the EXECUTE IMMEDIATE statement:

```
EXEC SQL EXECUTE IMMEDIATE 'DELETE FROM emp WHERE empno = 9460'
```

Related Topics

PREPARE command on 4 – 381
EXECUTE command on 4 – 332

EXPLAIN PLAN

Purpose

To determine the execution plan Oracle7 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.

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 on page 4 – 361.

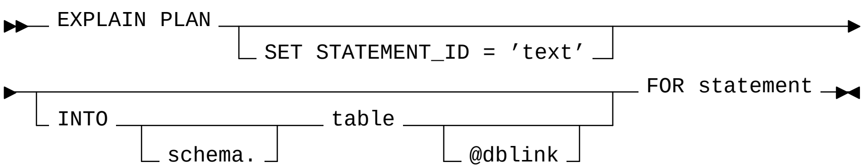
You must also have the privileges necessary to execute the SQL statement for which you are determining the execution plan. If the SQL statement accesses a view, you must have privileges to access any tables and views on which the view is based. If the view is based on another view that is based on a 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 on page 4 – 405.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must dominate the output table's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

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*, Oracle7 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. For information on referring to database links, see the section, "Referring to Objects in Remote Objects," on page 2 – 11. You can only specify a remote output table if you are using Oracle7 with the distributed option. If you omit *dblink*, Oracle7 assumes the table is on your local database.

  If you omit the INTO clause altogether, Oracle7 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.

Usage Notes

The definition of a sample output table PLAN_TABLE is available in SQL script on your distribution media. Your output table must have the same column names and datatypes as this table. The common name of this script is UTLXPLAN.SQL, although the exact name and location may vary depending on your operating system.

The value you specify in the SET clause appears in the STATEMENT_ID column in the rows of the execution plan. You can then use this value to identify these rows among others in the output table. Be sure to specify a STATEMENT_ID value if your output table contains rows from many execution plans.

Since the EXPLAIN PLAN command is a Data Manipulation Language command, rather than a Data Definition Language command, Oracle7 does not implicitly commit the changes made by an EXPLAIN PLAN statement. If you want to keep the rows generated by an EXPLAIN PLAN statement in the output table, you must commit the transaction containing the statement.

You should not use the EXPLAIN PLAN command to determine the execution plans of SQL statements that access data dictionary views or dynamic performance tables.

You can also issue the EXPLAIN PLAN command as part of the SQL trace facility. For information on how to use the SQL trace facility and how to interpret execution plans, see Appendix A "Performance Diagnostic Tools" of *Oracle7 Server Tuning*.

Example

This EXPLAIN PLAN statement determines the execution plan and cost for an UPDATE statement and inserts rows describing the execution plan into the specified OUTPUT table with the STATEMENT_ID value of 'Raise in Chicago':

```
EXPLAIN PLAN
    SET STATEMENT_ID = 'Raise in Chicago'
    INTO output
    FOR UPDATE emp
        SET sal = sal * 1.10
        WHERE deptno =  (SELECT deptno
                    FROM dept
                    WHERE loc = 'CHICAGO')
```

This SELECT statement queries the OUTPUT table and returns the execution plan and the cost:

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

The query returns this execution plan:

```
OPERATION             OPTIONS     OBJECT_NAME     POSITION
----------------------------------------------------------
UPDATE STATEMENT                                         1
  FILTER                                                 0
    TABLE ACCESS      FULL        EMP                    1
    TABLE ACCESS      FULL        DEPT                   2
```

The value in the POSITION column of the first row shows that the statement has a cost of 1.

Related Topics

Appendix A of *Oracle7 Server Tuning*.

FETCH (Embedded SQL)

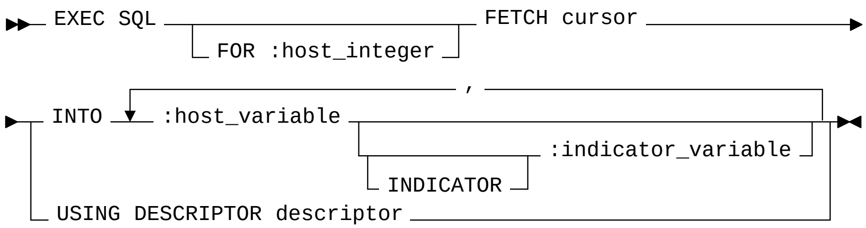
Purpose

To retrieve one or more rows returned by a query, assigning the select list values to host variables.

Prerequisites

You must first open the cursor with an the OPEN statement.

Syntax



Keywords and Parameters

FOR :*host_integer* — limits the number of rows fetched if you are using array host variables. If you omit this clause, Oracle7 fetches enough rows to fill the smallest array.

cursor — is a cursor that has been declared by a DECLARE CURSOR statement. The FETCH statement returns one of the rows selected by the query associated with the cursor.

INTO — specifies a list of host variables and optional indicator variables into which data is fetched. These host variables and indicator variables must be declared within the program.

USING — specifies the descriptor referenced in a previous DESCRIBE statement. Only use this clause with dynamic embedded SQL, method 4.

Usage Notes

The FETCH statement reads the rows of the active set and names the output variables which contain the results. Indicator values are set to –1 if their associated host variable is null. The first FETCH statement for a cursor also sorts the rows of the active set, if necessary.

The number of rows retrieved is specified by the size of the output host variables and the value specified in the FOR clause. The host variables to receive the data must be either all scalars or all arrays. If they are scalars, Oracle7 fetches only one row. If they are arrays, Oracle7 fetches enough rows to fill the arrays.

Array host variables can have different sizes. In this case, the number of rows Oracle7 fetches is determined by the smaller of the following values:

- the size of the smallest array
- the value of the :*host_integer* in the optional FOR clause

Of course, the number of rows fetched can be further limited by the number of rows that actually satisfy the query.

If a FETCH statement does not retrieve all rows returned by the query, the cursor is positioned on the next returned row. When the last row returned by the query has been retrieved, the next FETCH statement results in an error code returned in the SQLCODE element of the SQLCA.

Note that the FETCH command does not contain an AT clause. You must specify the database accessed by the cursor in the DECLARE CURSOR statement.

You can only move forward through the active set with FETCH statements. If you want to revisit any of the previously fetched rows, you must reopen the cursor and fetch each row in turn. If you want to change the active set, you must assign new values to the input host variables in the cursor's query and reopen the cursor.

Example

This example illustrates the FETCH command in a pseudo–code embedded SQL program:

```
EXEC SQL DECLARE emp_cursor CURSOR FOR
   SELECT job, sal FROM emp WHERE deptno = 30;

...

EXEC SQL WHENEVER NOT FOUND GOTO ...

LOOP

   EXEC SQL FETCH emp_cursor INTO :job_title1, :salary1;
   EXEC SQL FETCH emp_cursor INTO :job_title2, :salary2;

   ...

END LOOP;

...
```

Related Topics

PREPARE command on 4 – 381
DECLARE CURSOR command on 4 – 280
OPEN command on 4 – 376
CLOSE command on 4 – 139

Filespec

Purpose

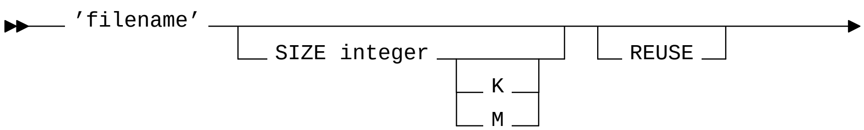
To either specify a file as a data file or specify a group of one or more files as a redo log file group.

Prerequisites

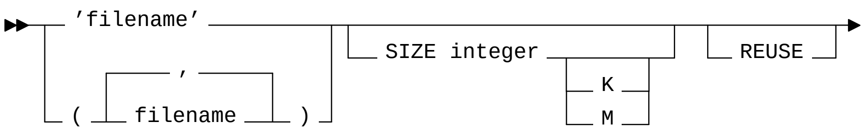
A *filespec* can appear in either CREATE DATABASE, ALTER DATABASE, CREATE TABLESPACE, or ALTER TABLESPACE commands. You must have the privileges necessary to issue one of these commands. For information on these privileges, see the CREATE DATABASE command on page 4 – 178, the ALTER DATABASE command on page 4 – 16, the CREATE TABLESPACE command on page 4 – 254, and the ALTER TABLESPACE command on page 4 – 98.

Syntax

filespec (Data Files) ::=



filespec (Redo Log File Groups) ::=



Keywords and Parameters

'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. Note that the tablespace size must be one block greater than the sum of the sizes of the objects contained in it.

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 Oracle7 to reuse an existing file. If the file already exists, Oracle7 verifies that its size matches the value of the SIZE parameter. If the file does not exist, Oracle7 creates it. If you omit this option, the file must not already exist and Oracle7 creates the file.

The REUSE option is only significant when used with the SIZE option. If you omit the SIZE option, Oracle7 expects the file to exist already. Note that whenever Oracle7 uses an existing file, the file's previous contents are lost.

Example I

The following statement creates a database named PAYABLE that has two redo log file groups, each with two members, and one data file:

```
CREATE DATABASE payable
    LOGFILE GROUP 1 ('diska:log1.log', 'diskb:log1.log') SIZE 50K,
            GROUP 2 ('diska:log2.log', 'diskb:log2.log') SIZE 50K
    DATAFILE 'diskc:dbone.dat' SIZE 30M
```

The first *filespec* in the LOGFILE clause specifies a redo log file group with the GROUP value 1. This group has members named 'DISKA:LOG1.LOG' and 'DISKB:LOG1.LOG' each with size 50 kilobytes.

The second *filespec* in the LOGFILE clause specifies a redo log file group with the GROUP value 2. This group has members named 'DISKA:LOG2.LOG' and 'DISKB:LOG2.LOG', also with sizes of 50 kilobytes.

The *filespec* in the DATAFILE clause specifies a data file named 'DISKC:DBONE.DAT' of size 30 megabytes.

Since all of these *filespecs* specify a value for the SIZE parameter and omit the REUSE option, these files must not already exist. Oracle7 must create them.

Example II The following statement adds another redo log file group with two members to the PAYABLE database:

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

The *filespec* in the ADD LOGFILE clause specifies a new redo log file group with the GROUP value 3. This new group has members named 'DISKA:LOG3.LOG' and 'DISKB:LOG3.LOG' with sizes of 50 kilobytes each. Since the *filespec* specifies the REUSE option, each member can already exist. If a member exists, it must have a size of 50 kilobytes. If it does not exist, Oracle7 creates it with that size.

Example III The following statement creates a tablespace named STOCKS that has three data files:

```
CREATE TABLESPACE stocks
   DATAFILE 'diskc:stock1.dat',
            'diskc:stock2.dat',
            'diskc:stock3.dat'
```

The *filespecs* for the data files specifies files named 'DISKC:STOCK1.DAT', 'DISKC:STOCK2.DAT', 'DISKC:STOCK3.DAT'. Since each *filespec* omits the SIZE parameter, each file must already exist.

Example IV The following statement alters the STOCKS tablespace and adds a new data file:

```
ALTER TABLESPACE stocks
   ADD DATAFILE 'diskc:stock4.dat' REUSE
```

The *filespec* specifies a data file named 'DISKC:STOCK4.DAT'. Since the *filespec* omits the SIZE parameter, the file must already exist and the REUSE option is not significant.

Related Topics

CREATE DATABASE command on 4 – 178
ALTER DATABASE command on 4 – 16
CREATE TABLESPACE command on 4 – 254
ALTER TABLESPACE command on 4 – 98

GRANT (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) described in the next section of this chapter.

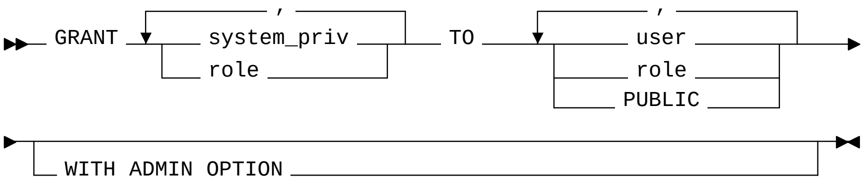
Prerequisites

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 Oracle7 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.

Syntax



Keywords and Parameters

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.

Usage Notes

You can use this form of the GRANT command to grant both system privileges and roles to users, roles, and PUBLIC:

If you grant a privilege to a user: Oracle7 adds the privilege to the user's privilege domain. The user can immediately exercise the privilege.

If you grant a privilege to a role: Oracle7 adds the privilege to the role's privilege domain. Users who have been granted and have enabled the role can immediately exercise the privilege. Other users who have been granted the role can enable the role and exercise the privilege.

If you grant a privilege to PUBLIC: Oracle7 adds the privilege to the privilege domains of each user. All users can immediately perform operations authorized by the privilege.

If you grant a role to a user: Oracle7 makes the role available to the user. The user can immediately enable the role and exercise the privileges in the role's privilege domain.

If you grant a role to another role: Oracle7 adds the granted role's privilege domain to the grantee role's privilege domain. Users who have been granted the grantee role can enable it and exercise the privileges in the granted role's privilege domain.

If you grant a role to PUBLIC: Oracle7 makes the role available to all users. All users can immediately enable the role and exercise the privileges in the roles privilege domain.

A privilege or role cannot appear more than once in the list of privileges and roles to be granted. A user, role, or PUBLIC cannot appear more than once in the TO clause.

You cannot grant roles circularly. For example, if you grant the role BANKER to the role TELLER, you cannot subsequently grant TELLER to BANKER. Also, you cannot grant a role to itself.

System Privileges

Table 4 – 11 lists system privileges and the operations that they authorize. You can grant any of these system privileges with the GRANT command.

System Privilege	Operations Authorized
ALTER ANY CLUSTER	Allows grantee to alter any cluster in any schema.
ALTER ANY INDEX	Allows grantee to alter any index in any schema
ALTER ANY PROCEDURE	Allows grantee to alter any stored procedure, function, or package in any schema.
ALTER ANY ROLE	Allows grantee to alter any role in the database.
ALTER ANY SEQUENCE	Allows grantee to alter any sequence in the database.
ALTER ANY SNAPSHOT	Allows grantee to alter any snapshot in the database.
ALTER ANY TABLE	Allows grantee to alter any table or view in the schema.
ALTER ANY TRIGGER	Allows grantee to enable, disable, or compile any database trigger in any schema.
ALTER DATABASE	Allows grantee to alter the database.
ALTER PROFILE	Allows grantee to alter profiles.
ALTER RESOURCE COST	Allows grantee to set costs for session resources.
ALTER ROLLBACK SEGMENT	Allows grantee to alter rollback segments.
ALTER SESSION	Allows grantee to issue ALTER SESSION statements.
ALTER SYSTEM	Allows grantee to issue ALTER SYSTEM statements.
ALTER TABLESPACE	Allows grantee to alter tablespaces.
ALTER USER	Allows grantee to alter any user. This privilege authorizes the grantee to change another user's password or authentication method, assign quotas on **any** tablespace, set default and temporary tablespaces, and assign a profile and default roles.
ANALYZE ANY	Allows grantee to analyze any table, cluster, or index in any schema.
AUDIT ANY	Allows grantee to audit any object in any schema using AUDIT (Schema Objects) statements.
AUDIT SYSTEM	Allows grantee to issue AUDIT (SQL Statements) statements.
BACKUP ANY TABLE	Allows grantee to use the Export utility to incrementally export objects from the schema of other users.
BECOME USER	Allows grantee to become another user. (Required by any user performing a full database import.)

Table 4 – 11 System Privileges

System Privilege	Operations Authorized
COMMENT ANY TABLE	Allows grantee to comment on any table, view, or column in any schema.
CREATE ANY CLUSTER	Allows grantee to create a cluster in any schema. Behaves similarly to CREATE ANY TABLE.
CREATE ANY INDEX	Allows grantee to create an index in any schema on any table in any schema.
CREATE ANY PROCEDURE	Allows grantee to create stored procedures, functions, and packages in any schema.
CREATE ANY SEQUENCE	Allows grantee to create a sequence in any schema.
CREATE ANY SNAPSHOT	Allows grantee to create snapshots in any schema.
CREATE ANY SYNONYM	Allows grantee to create private synonyms in any schema.
CREATE ANY TABLE	Allows grantee to create tables in any schema. The owner of the schema containing the table must have space quota on the tablespace to contain the table.
CREATE ANY TRIGGER	Allows grantee to create a database trigger in any schema associated with a table in any schema.
CREATE ANY VIEW	Allows grantee to create views in any schema.
CREATE CLUSTER	Allows grantee to create clusters in own schema.
CREATE DATABASE LINK	Allows grantee to create private database links in own schema.
CREATE PROCEDURE	Allows grantee to create stored procedures, functions, and packages in own schema.
CREATE PROFILE	Allows grantee to create profiles.
CREATE PUBLIC DATABASE LINK	Allows grantee to create public database links.
CREATE PUBLIC SYNONYM	Allows grantee to create public synonyms.
CREATE ROLE	Allows grantee to create roles.
CREATE ROLLBACK SEGMENT	Allows grantee to create rollback segments.
CREATE SEQUENCE	Allows grantee to create sequences in own schema.
CREATE SESSION	Allows grantee to connect to the database.
CREATE SNAPSHOT	Allows grantee to create snapshots in own schema.
CREATE SYNONYM	Allows grantee to create synonyms in own schema.
CREATE TABLE	Allows grantee to create tables in own schema. To create a table, the grantee must also have space quota on the tablespace to contain the table.

Table 4 – 11 (continued) System Privileges

System Privilege	Operations Authorized
CREATE TABLESPACE	Allows grantee to create tablespaces.
CREATE TRIGGER	Allows grantee to create a database trigger in own schema.
CREATE USER	Allows grantee to create users. This privilege also allows the creator to assign quotas on **any** tablespace, set default and temporary tablespaces, and assign a profile as part of a CREATE USER statement.
CREATE VIEW	Allows grantee to create views in own schema.
DELETE ANY TABLE	Allows grantee to delete rows from tables or views in any schema or truncate tables in any schema.
DROP ANY CLUSTER	Allows grantee to drop clusters in any schema.
DROP ANY INDEX	Allows grantee to drop indexes in any schema.
DROP ANY PROCEDURE	Allows grantee to drop stored procedures, functions, or packages in any schema.
DROP ANY ROLE	Allows grantee to drop roles.
DROP ANY SEQUENCE	Allows grantee to drop sequences in any schema.
DROP ANY SNAPSHOT	Allows grantee to drop snapshots in any schema.
DROP ANY SYNONYM	Allows grantee to drop private synonyms in any schema.
DROP ANY TABLE	Allows grantee to drop tables in any schema.
DROP ANY TRIGGER	Allows grantee to drop database triggers in any schema.
DROP ANY VIEW	Allows grantee to drop views in any schema
DROP PROFILE	Allows grantee to drop profiles.
DROP PUBLIC DATABASE LINK	Allows grantee to drop public database links.
DROP PUBLIC SYNONYM	Allows grantee to drop public synonyms.
DROP ROLLBACK SEGMENT	Allows grantee to drop rollback segments.
DROP TABLESPACE	Allows grantee to drop tablespaces.
DROP USER	Allows grantee to drop users.
EXECUTE ANY PROCEDURE	Allows grantee to execute procedures or functions (stand–alone or packaged) or reference public package variables in any schema.
FORCE ANY TRANSACTION	Allows grantee to for the commit or rollback of any in–doubt distributed transaction in the local database. Also allows the grantee to induce the failure of a distributed transaction.

Table 4 – 11 (continued) System Privileges

System Privilege	Operations Authorized
FORCE TRANSACTION	Allows grantee to force the commit or rollback of own in–doubt distributed transactions in the local database.
GRANT ANY PRIVILEGE	Allows grantee to grant any system privilege.
GRANT ANY ROLE	Allows grantee to grant any role in the database.
INSERT ANY TABLE	Allows grantee to insert rows into tables and views in any schema.
LOCK ANY TABLE	Allows grantee to lock tables and views in any schema.
MANAGE TABLESPACE	Allows grantee to take tablespaces offline and online and begin and end tablespace backups.
READUP	Allows grantee to query data having an access class higher than the grantee's session label. This privilege is only available in Trusted Oracle7.
RESTRICTED SESSION	Allows grantee to logon after the instance is started using the Server Manager STARTUP RESTRICT command.
SELECT ANY SEQUENCE	Allows grantee to reference sequences in any schema.
SELECT ANY TABLE	Allows grantee to query tables, views, or snapshots in any schema.
UNLIMITED TABLESPACE	Allows grantee to use an unlimited amount of any tablespace. This privilege overrides any specific quotas assigned. If you revoke this privilege from a user, the grantee's schema objects remain but further tablespace allocation is denied unless authorized by specific tablespace quotas. You cannot grant this system privilege to roles.
UPDATE ANY TABLE	Allows grantee to update rows in tables and views in any schema.
WRITEDOWN	Allows grantee to create, alter, and drop schema objects and to insert, update, and delete rows having access classes lower than the grantee's session label. This privilege is only available in Trusted Oracle7.
WRITEUP	Allows grantee to create, alter, and drop schema objects and to insert, update, and delete rows having access classes higher than the grantee's session label. This privilege is only available in Trusted Oracle7.

Table 4 – 11 (continued) System Privileges

Roles Defined by Oracle7

Some roles are created automatically by Oracle7. When you create a database, Oracle7 creates these roles and grants them certain system privileges. Table 4 – 12 lists each predefined role and its system privileges.

Role	System Privileges and Roles Granted
CONNECT	ALTER SESSION CREATE CLUSTER CREATE DATABASE LINK CREATE SEQUENCE CREATE SESSION CREATE SYNONYM CREATE TABLE CREATE VIEW
RESOURCE	CREATE CLUSTER CREATE PROCEDURE CREATE SEQUENCE CREATE TABLE CREATE TRIGGER
DBA	All systems privileges WITH ADMIN OPTION EXP_FULL_DATABASE role IMP_FULL_DATABASE role
EXP_FULL_DATABASE	SELECT ANY TABLE BACKUP ANY TABLE INSERT, UPDATE, DELETE ON sys.incexp sys.incvid sys.incfil
IMP_FULL_DATABASE	BECOME USER WRITEDOWN (in Trusted Oracle7)

Table 4 – 12 Roles defined by Oracle7

> **Note:** If you grant or revoke the RESOURCE or DBA role to or from a user, Oracle7 implicitly grants or revokes the UNLIMITED TABLESPACE system privilege to or from the user.

The CONNECT, RESOURCE, and DBA are provided for compatibility with previous versions of Oracle7. The SQL script SQL.BSQ creates these roles, grants privileges to them, and grants the DBA role with ADMIN OPTION to the users SYS and SYSTEM. This script is available on your distribution media, although its exact name and location may vary depending on your operating system. It is recommended that you to design your own roles for database security, rather than rely on these roles. These roles may not be automatically created by future versions of Oracle7.

The EXP_FULL_DATABASE and IMP_FULL_DATABASE roles are provided for convenience in using the Import and Export utilities. The SQL script CATEXP.SQL creates these roles, grants privileges to them, and grants them to the DBA role. This script is available on your distribution media, although its exact name and location may vary depending on your operating system.

DBA Role

Because the DBA role has all system privileges, a common misperception is that no other privileges are required to administer privileges on objects in the database. Although this is generally true, you may still need to grant object privileges to a user granted the DBA role. For example, for USER1 granted the DBA role to create a foreign key constraint against USER2's tables, USER2 must grant the REFERENCES object privilege on the tables to USER1.

ADMIN OPTION

A grant with the ADMIN OPTION supersedes a previous identical grant without the ADMIN OPTION. If you grant a system privilege or role to user without the ADMIN OPTION, and then subsequently grant the privilege or role to the user with the ADMIN OPTION, the user has the ADMIN OPTION on the privilege or role.

A grant without the ADMIN OPTION does not supersede a previous grant with the ADMIN OPTION. To revoke the ADMIN OPTION on a system privilege or role from a user, you must revoke the privilege or role from the user altogether and then grant the privilege or role to the user without the ADMIN OPTION.

Granting Roles Through Your Operating System

Some operating systems have facilities that grant operating system privileges to operating system users. You can use such facilities to grant roles to Oracle7 users with the initialization parameter OS_ROLES. If you choose to grant roles to users through operating system facilities, you cannot also grant roles to users with the GRANT command, although you can use the GRANT command to grant system privileges to users and system privileges and roles to other roles.

Example I

To grant the CREATE SESSION system privilege to RICHARD, allowing RICHARD to logon to Oracle7, issue the following statement:

```
GRANT CREATE SESSION
   TO richard
```

Example II

To grant the CREATE TABLE system privilege to the role TRAVEL_AGENT, issue the following statement:

```
GRANT CREATE TABLE
   TO travel_agent
```

TRAVEL_AGENT's privilege domain now contains the CREATE TABLE system privilege.

The following statement grants the TRAVEL_AGENT role to the EXECUTIVE role:

```
GRANT travel_agent
    TO executive
```

TRAVEL_AGENT is now granted to EXECUTIVE. EXECUTIVE's privilege domain contains the CREATE TABLE system privilege.

To grant the EXECUTIVE role with the ADMIN OPTION to THOMAS, issue the following statement:

```
GRANT executive
    TO thomas
    WITH ADMIN OPTION
```

THOMAS can now perform the following operations with the EXECUTIVE role:

- enable the role and exercise any privileges in the role's privilege domain, including the CREATE TABLE system privilege
- grant and revoke the role to and from other users
- drop the role

Related Topics

ALTER USER command on 4 – 108
CREATE USER command on 4 – 267
GRANT (Object Privileges) command on 4 – 355
REVOKE (System Privileges and Roles) command on 4 – 388

GRANT (Object Privileges)

Purpose

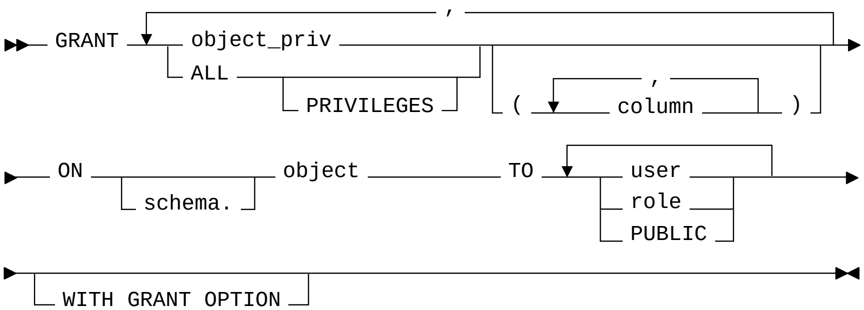
To grant privileges for a particular object to users and roles. To grant system privileges and roles, use the GRANT command (System Privileges and Roles) described in the previous section of this chapter.

Prerequisites

You must own the object or the owner of the object granted you the object privileges with the GRANT OPTION. This rule applies to users with the DBA role.

If you are using Trusted Oracle7 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.

Syntax



Keywords and Parameters

object_priv is an object privilege to be granted. You can substitute any of the following 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*, Oracle7 assumes the object is in your own schema. The object can be one of the following 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.

Usage Notes

You can use this form of the GRANT statement to grant object privileges to users, roles, and PUBLIC:

If you grant a privilege to a user: Oracle7 adds the privilege to the user's privilege domain. The user can immediately exercise the privilege.

If you grant a privilege to a role: Oracle7 adds the privilege to the role's privilege domain. Users who have been granted and have enabled the role can immediately exercise the privilege. Other users who have been granted the role can enable the role and exercise the privilege.

If you grant a privilege to PUBLIC: Oracle7 adds the privilege to the privilege domain of each user. All users can immediately exercise the privilege.

A privilege cannot appear more than once in the list of privileges to be granted. A user or role cannot appear more than once in the TO clause.

Object Privileges

Each object privilege that you grant authorizes the grantee to perform some operation on the object. Table 4 – 13 summarizes the object privileges that you can grant on each type of object.

Object Privilege	Tables	Views	Sequences	Procedure Functions Packages	Snapshots
ALTER	3		3		
DELETE	3	3			
EXECUTE				3	
INDEX	3				
INSERT	3	3			
REFERENCES	3				
SELECT	3	3	3		3
UPDATE	3	3			

Table 4 – 13 Object Privileges

Table Privileges

The following object privileges authorize operations on a table:

ALTER	allows the grantee to change the table definition with the ALTER TABLE command.
DELETE	allows the grantee to remove rows from the table with the DELETE command.
INDEX	allows the grantee to create an index on the table with the CREATE INDEX command.
INSERT	allows the grantee to add new rows to the table with the INSERT command.
REFERENCES	allows the grantee to create a constraint that refers to the table. You cannot grant this privilege to a role.
SELECT	allows the grantee to query the table with the SELECT command.
UPDATE	allows the grantee to change data in the table with the UPDATE command.

Any one of above object privileges allows the grantee to lock the table in any lock mode with the LOCK TABLE command.

View Privileges

The following object privileges authorize operations on a view:

DELETE	allows the grantee to remove rows from the view with the DELETE command.
INSERT	allows the grantee to add new rows to the view with the INSERT command.
SELECT	allows the grantee to query the view with the SELECT command.
UPDATE	allows the grantee to change data in the view with the UPDATE command.

Any one of the above object privileges allows the grantee to lock the view in any lock mode with the LOCK TABLE command.

To grant a privilege on a view, you must have that privilege with the GRANT OPTION on all of the view's base tables.

Sequence Privileges

The following object privileges authorize operations on a sequence:

ALTER	allows the grantee to change the sequence definition with the ALTER SEQUENCE command.
SELECT	allows the grantee to examine and increment values of the sequence with the CURRVAL and NEXTVAL pseudocolumns.

Procedure, Function, and Package Privileges

This object privilege authorizes operations on a procedure, function, or package:

EXECUTE	allows the grantee to execute the procedure or function or to access any program object declared in the specification of a package.

Snapshot Privileges

This object privilege authorizes operations on a snapshot:

SELECT	allows the grantee to query the snapshot with the SELECT command.

Synonym Privileges

The object privileges available for a synonym are the same as the privileges for the synonym's base object. Granting a privilege on a synonym is equivalent to granting the privilege on the base object. Similarly, granting a privilege on a base object is equivalent to granting the privilege on all synonyms for the object. If you grant a user a privilege on a synonym, the user can use either the synonym name or the base object name in the SQL statement that exercises the privilege.

Example I

To grant all privileges on the table BONUS to the user JONES with the GRANT OPTION, issue the following statement:

```
GRANT ALL
    ON bonus
    TO jones
    WITH GRANT OPTION
```

JONES can subsequently perform the following operations:

- exercise any privilege on the BONUS table
- grant any privilege on the BONUS table to another user or role

Example II To grant SELECT and UPDATE privileges on the view GOLF_HANDICAP to all users, issue the following statement:

```
GRANT SELECT, UPDATE
    ON golf_handicap
    TO PUBLIC
```

All users can subsequently query and update the view of golf handicaps.

Example III To grant SELECT privilege on the ESEQ sequence in the schema ELLY to the user BLAKE, issue the following statement:

```
GRANT SELECT
    ON elly.eseq
    TO blake
```

BLAKE can subsequently generate the next value of the sequence with the following statement:

```
SELECT elly.eseq.NEXTVAL
    FROM DUAL
```

Example IV To grant BLAKE the REFERENCES privilege on the EMPNO column and the UPDATE privilege on the EMPNO, SAL, and COMM columns of the EMP table in the schema SCOTT, issue the following statement:

```
GRANT REFERENCES (empno), UPDATE (empno, sal, comm)
    ON scott.emp
    TO blake
```

BLAKE can subsequently update values of the EMPNO, SAL, and COMM columns. BLAKE can also define referential integrity constraints that refer to the EMPNO column. However, since the GRANT statement lists only these columns, BLAKE cannot perform operations on any of the other columns of the EMP table.

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

```
CREATE TABLE dependent
 (dependno   NUMBER,
  dependname VARCHAR2(10),
  employee   NUMBER
             CONSTRAINT in_emp REFERENCES scott.emp(empno) )
```

The constraint IN_EMP ensures that all dependents in the DEPENDENT table correspond to an employee in the EMP table in the schema SCOTT.

Related Topics

GRANT (System Privileges and Roles) command on 4 – 346
REVOKE (Object Privileges) command on 4 – 391

INSERT

Purpose

To add rows to a table or to a view's base table.

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.

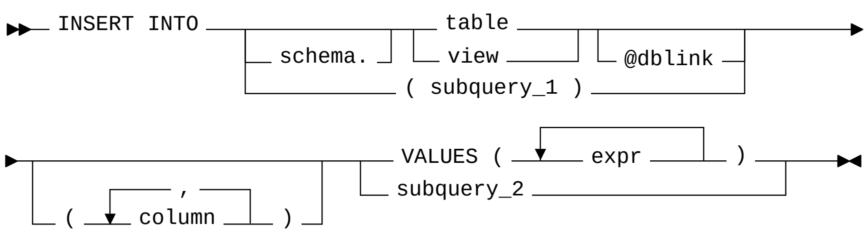
For you to insert rows into the base table of a view, the owner of the schema containing the view must have INSERT privilege on the base table. Also, if the view is in a schema other than your own, you must have INSERT privilege on the view.

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 Oracle7 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 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 WRITEUP and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema is the schema containing the table or view. If you omit *schema*, Oracle7 assumes the table or view is in your own schema.

table
view is the name of the table into which rows are to be inserted. If you specify *view*, Oracle7 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. For information on referring to database links, see the section "Referring to Objects" on page 2 – 11. You can only insert rows into a remote table or view if you are using Oracle7 with the distributed option.

If you omit *dblink*, Oracle7 assumes that the table or view is on the local database.

subquery_1 is a subquery that Oracle treats in the same manner as a view. For the syntax of *subquery*, see page 4 – 431.

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. See the syntax description of *expr* on page 3 – 73. You must specify a value in the VALUES clause for each column in the column list.

subquery_2 is a subquery 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. For the syntax description of *subquery*, see page 4 – 436.

Usage Notes

An INSERT statement with a VALUES clause adds a single row to the table. This row contains the values specified in the VALUES clause.

An INSERT statement with a subquery instead of a VALUES clause adds to the table all rows returned by the subquery. Oracle7 processes the subquery and inserts each returned row into the table. If the subquery selects no rows, Oracle7 inserts no rows into the table. The subquery can refer to any table, view, or snapshot, including the target table of the INSERT statement.

The number of columns in the column list of the INSERT statement must be the same as the number of values in the VALUES clause or the number of columns selected by the subquery. If you omit the column list, then the VALUES clause or the subquery must provide values for every column in the table. If you are using Trusted Oracle7 in DBMS MAC mode and you omit a value for the ROWLABEL column, the new row is automatically labeled with your DBMS label.

Oracle7 assigns values to fields in new rows based on the internal position of the columns in the table and the order of the values in the VALUES clause or in the select list of the query. You can determine the position of each column in the table by examining the data dictionary. See the "Data Dictionary" chapter in *Oracle7 Server Reference*.

If you omit any columns from the column list, Oracle7 assigns them their default values as specified when the table was created. For more information on the default column values, see the CREATE TABLE command on page 4 – 245. If any of these columns has a NOT NULL constraint, then Oracle7 returns an error indicating that the constraint has been violated and rolls back the INSERT statement.

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

Inserting Into Views

If a view was created using the WITH CHECK OPTION, then you can only insert rows into the view that satisfy the view's defining query.

You cannot insert rows into a view if the view's defining query contains one of the following constructs:

- join
- set operator
- GROUP BY clause
- group function
- DISTINCT operator

Example I The following statement inserts a row into the DEPT table:

```
INSERT INTO dept
   VALUES (50, 'PRODUCTION', 'SAN FRANCISCO')
```

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

```
INSERT INTO emp (empno, ename, job, sal, comm, deptno)
   VALUES (7890, 'JINKS', 'CLERK', 1.2E3, NULL, 40)
```

Example III The following statement has the same effect as Example II:

```
INSERT INTO (select empno, job, sal, comm, deptno from emp)
   VALUES (7890, 'JINKS', 'CLERK', 1.2E3, NULL, 40)
```

Example IV The following statement copies managers and presidents or employees whose commission exceeds 25% of their salary into the BONUS table:

```
INSERT INTO bonus
   SELECT ename, job, sal, comm
      FROM emp
      WHERE comm > 0.25 * sal
         OR job IN ('PRESIDENT', 'MANAGER')
```

Example V The following statement inserts a row into the ACCOUNTS table owned by the user SCOTT on the database accessible by the database link SALES:

```
INSERT INTO scott.accounts@sales (acc_no, acc_name)
   VALUES (5001, 'BOWER')
```

Assuming that the ACCOUNTS table has a BALANCE column, the newly inserted row is assigned the default value for this column because this INSERT statement does not specify a BALANCE value.

Example VI The following statement inserts a new row containing the next value of the employee sequence into the EMP table:

```
INSERT INTO emp
   VALUES (empseq.nextval, 'LEWIS', 'CLERK',
           7902, SYSDATE, 1200, NULL, 20)
```

Related Topics

DELETE command on 4 – 286
UPDATE command on 4 – 460

INSERT (Embedded SQL)

Purpose

To add rows to a table or to a view’s base table.

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.

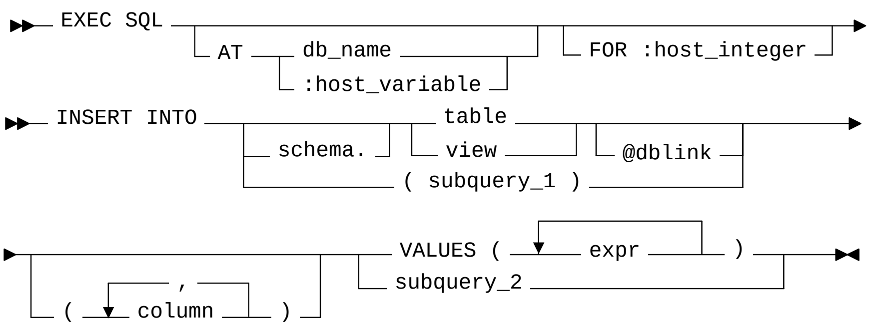
For you to insert rows into the base table of a view, the owner of the schema containing the view must have INSERT privilege on the base table. Also, if the view is in a schema other than your own, you must have INSERT privilege on the view.

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 Oracle7 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 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 WRITEUP and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

AT
: identifies the database on which the INSERT statement is executed. The database can be identified by either:

  db_name — is a database identifier declared in a previous DECLARE DATABASE statement.

  :*host_variable* — is a host variable whose value is a previously declared *db_name*

  If you omit this clause, the INSERT statement is executed on your default database.

FOR :*host_integer*
: limits the number of times the statement is executed if the VALUES clause contains array host variables. If you omit this clause, Oracle7 executes the statement once for each component in the smallest array.

schema
: is the schema containing the table or view. If you omit *schema*, Oracle7 assumes the table or view is in your own schema.

table
view
: is the name of the table into which rows are to be inserted. If you specify *view*, Oracle7 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. For information on referring to database links, see the section, "Referring to Objects in Remote Databases," on page 2 – 11. You can only insert rows into a remote table or view if you are using Oracle7 with the distributed option.

  If you omit *dblink*, Oracle7 assumes that the table or view is on the local database.

subquery_1
: is a subquery that Oracle treats in the same manner as a view. For the syntax of subquery, see page 4 – 431.

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 query.

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. See the syntax description of *expr* on page 3 – 73. Note that the expressions can be host variables with optional indicator variables. You must specify an expression in the VALUES clause for each column in the column list.

subquery_2 — is a subquery 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. For the syntax description of *subquery*, see page 4 – 436.

Usage Notes

Any host variables that appear in the WHERE clause must be either all scalars or all arrays. If they are scalars, Oracle7 executes the INSERT statement once. If they are arrays, Oracle7 executes the INSERT statement once for each set of array components, inserting one row each time.

Array host variables in the WHERE clause can have different sizes. In this case, the number of times Oracle7 executes the statement is determined by the smaller of the following values:

- size of the smallest array
- the value of the :*host_integer* in the optional FOR clause.

For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example I This example illustrates the use of the embedded SQL INSERT command:

```
EXEC SQL INSERT INTO emp (ename, empno, sal)
        VALUES (:ename, :empno, :sal);
```

Example II This example shows an embedded SQL INSERT command with a subquery:

```
EXEC SQL INSERT INTO new_emp (ename, empno, sal)
           SELECT ename, empno, sal FROM emp
           WHERE deptno = :deptno;
```

Related Topics

DECLARE DATABASE command on 4 – 282
INSERT command on 4 – 361

LOCK TABLE

Purpose

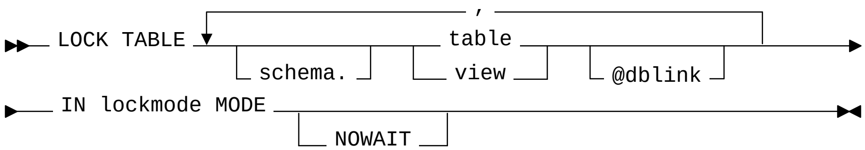
To lock one or more tables in a specified mode. This lock manually overrides automatic locking and permits or denies access to a table or view by other users for the duration of your operation.

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 Oracle7 in DBMS MAC mode, your DBMS label must dominate the creation label of the table or view or you must have READUP system privilege.

Syntax



Keywords and Parameters

schema is the schema containing the table or view. If you omit *schema*, Oracle7 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*, Oracle7 locks the view's base tables.

dblink is a database link to a remote Oracle7 database where the table or view is located. For information on specifying database links, see the section, "Referring to Objects in Remote Databases," on page 2 – 11. You can only lock tables and views on a remote database if you are using Oracle7 with the distributed option. All tables locked by a LOCK TABLE statement must be on the same database.

If you omit *dblink*, Oracle7 assumes the table or view is on the local database.

lockmode is one of the following:

- ROW SHARE
- ROW EXCLUSIVE
- SHARE UPDATE
- SHARE
- SHARE ROW EXCLUSIVE
- EXCLUSIVE

NOWAIT specifies that Oracle7 returns control to you immediately if the specified table is already locked by another user. In this case, Oracle7 returns a message indicating that the table is already locked by another user.

If you omit this clause, Oracle7 waits until the table is available, locks it, and returns control to you.

Usage Notes

Exclusive locks allow queries on the locked table but prohibit any other activity on it.

Share locks allow concurrent queries but prohibit updates to the locked table.

Row Share locks allow concurrent access to the locked table. They prohibit users from locking the entire table for exclusive access. ROW SHARE is synonymous with SHARE UPDATE.

Row Exclusive locks are the same as ROW SHARE locks, but also prohibit locking in SHARE mode. Row Exclusive locks are automatically obtained when updating, inserting, or deleting.

Share Row Exclusive locks are used to look at a whole table and to allow others to look at rows in the table but to prohibit others from locking the table in SHARE mode or updating rows.

Share Update locks are synonymous with ROW SHARE and included for compatibility with earlier versions of the Oracle7 RDBMS.

Some forms of locks can be placed on the same table at the same time, other locks only allow one lock per table. For example, multiple users can place SHARE locks on the same table at the same time, but only one user can place an EXCLUSIVE lock on a table at a time. For a complete description of the interaction of lock modes, see the "Data Concurrency" chapter of *Oracle7 Server Concepts.*

When you lock a table, you choose how other users can access it. A locked table remains locked until you either commit your transaction or roll it back entirely or to a savepoint before you locked the table.

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

Example I

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

```
LOCK TABLE emp
   IN EXCLUSIVE MODE
   NOWAIT
```

Example II

The following statement locks the remote ACCOUNTS table that is accessible through the database link BOSTON:

```
LOCK TABLE accounts@boston
   IN SHARE MODE
```

Related Topics

DELETE command on 4 – 286
INSERT command on 4 – 361
UPDATE command on 4 – 460
COMMIT command on 4 – 141
ROLLBACK command on 4 – 397
SAVEPOINT command on 4 – 404

NOAUDIT (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) described in the next section of this chapter.

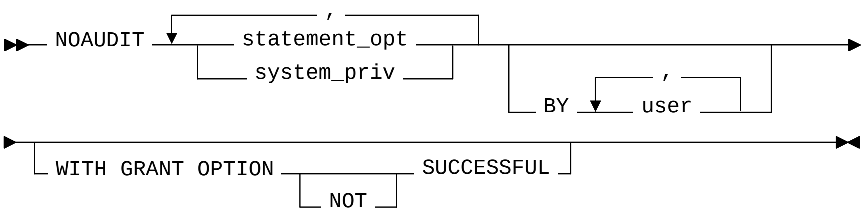
Prerequisites

You must have AUDIT SYSTEM system privilege.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the label at which the auditing option was set or you must satisfy one of the following 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

statement_opt is a statement option for which auditing is stopped. For a list of the statement options and the SQL statements they audit, see Table 4 – 7 beginning on page 4 – 130 and Table 4 – 8 on page 4 – 132.

system_priv is a system privilege for which auditing is stopped. For a list of the system privileges and the statements they authorize, see Table 4 – 7 on page 4 – 130.

BY stops auditing only for SQL statements issued by specified users in their subsequent sessions. If you omit this clause, Oracle7 stops auditing for all users' statements, except for the situation described in the section that follows.

WHENEVER SUCCESSFUL

stops auditing only for SQL statements that complete successfully.

NOT

stops auditing only for statements that result in Oracle7 errors.

If you omit the WHENEVER clause entirely, Oracle7 stops auditing for all statements, regardless of success or failure.

Usage Notes

A NOAUDIT statement (SQL Statements) reverses the effect of a previous AUDIT statement (SQL Statements). Note that the NOAUDIT statement must have the same syntax as the previous AUDIT statement and that it only reverses the effects of that particular statement. Therefore, if one AUDIT statement (statement A) enables auditing for a specific user, and a second (statement B) enables auditing for all users, then a NOAUDIT statement to disable auditing for all users (statement C) reverses statement B, but leaves statement A in effect and continues to audit the user that statement A specified. For information on auditing specific SQL commands, see the AUDIT command (SQL Statements) command on page 4 – 127.

Example I

If you have chosen auditing for every SQL statement that creates or drops a role, you can stop auditing of such statements by issuing the following statement:

```
NOAUDIT ROLE
```

Example II

If you have chosen auditing for any statement that queries or updates any table issued by the users SCOTT and BLAKE, you can stop auditing for SCOTT's queries by issuing the following statement:

```
NOAUDIT SELECT TABLE
    BY scott
```

Since the above statement only stops auditing SCOTT's queries, Oracle7 continues to audit BLAKE's queries and updates and SCOTT's updates.

Example III

To stop auditing on all statements that are authorized by DELETE ANY TABLE system privileges chosen for auditing, issue the following statement:

```
NOAUDIT ALL
```

Related Topics

AUDIT (SQL Statements) command on 4 – 127
NOAUDIT (Schema Objects) command on 4 – 374

NOAUDIT (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) described in the previous section of this chapter.

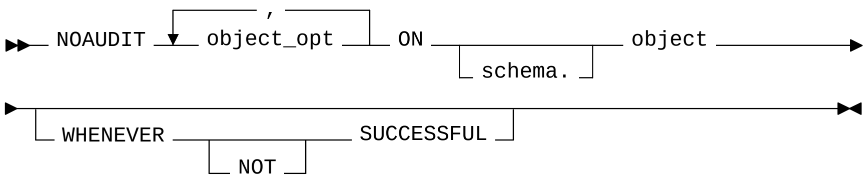
Prerequisites

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 Oracle7 in DBMS MAC mode, your DBMS label must match the label at which the auditing option was set or you must satisfy one of the following 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

object_opt — stops auditing for particular operations on the object. For a list of these options, see Table 4 – 9 on page 4 – 136.

ON — identifies the object on which auditing is stopped. If you do not qualify *object* with *schema*, Oracle7 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 Oracle7 errors.

If you omit the WHENEVER clause entirely, Oracle7 stops auditing for all statements, regardless of success or failure.

Usage Notes

For information on auditing specific schema objects, see the AUDIT command (Schema Objects) on page 4 – 134.

Example

If you have chosen auditing for every SQL statement that queries the EMP table in the schema SCOTT, you can stop auditing for such queries by issuing the following statement:

```
NOAUDIT SELECT
    ON scott.emp
```

You can stop auditing for such queries that complete successfully by issuing the following statement:

```
NOAUDIT SELECT
    ON scott.emp
    WHENEVER SUCCESSFUL
```

Since you only stopped auditing for successful queries, Oracle7 continues to audit queries resulting in Oracle7 errors.

Related Topics

AUDIT (Schema Objects) command on 4 – 134
NOAUDIT (SQL Statements) command on 4 – 372

OPEN (Embedded SQL)

Purpose

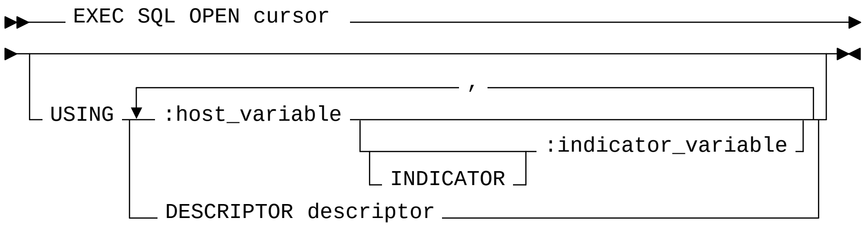
To open a cursor, evaluating the associated query and substituting the host variable names supplied by the USING clause into the WHERE clause of the query.

Prerequisites

You must declare the cursor with a DECLARE CURSOR embedded SQL statement before opening it.

Syntax

Syntax



Keywords and Parameters

cursor is the cursor to be opened.

USING specifies the host variables to be substituted into the WHERE clause of the associated query.

:*host_variable* specifies a host variable with an optional indicator variable to be substituted into the statement associated with the cursor.

DESCRIPTOR specifies a descriptor that describes the host variables to be substituted into the WHERE clause of the associated query. The *descriptor* must be initialized in a previous DESCRIBE statement.

The substitution is based on position. The host variable names specified in this statement can be different from the variable names in the associated query.

Usage Notes

The OPEN command defines the active set of rows and initializes the cursor just before the first row of the active set. The values of the host variables at the time of the OPEN are substituted in the statement. This command does not actually retrieve rows; rows are retrieved by the FETCH command.

Once you have opened a cursor, its input host variables are not reexamined until you reopen the cursor. To change any input host variables and therefore the active set, you must reopen the cursor.

All cursors in a program are in a closed state when the program is initiated or when they have been explicitly closed using the CLOSE command.

You can reopen a cursor without first closing it. For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example

This example illustrates the use of the OPEN command in a Pro*C embedded SQL program:

```
EXEC SQL DECLARE emp_cursor CURSOR FOR
   SELECT ename, empno, job, sal
   FROM emp
   WHERE deptno = :deptno;
EXEC SQL OPEN emp_cursor;
```

Related Topics

PREPARE command on 4 – 381
DECLARE CURSOR command on 4 – 280
FETCH command on 4 – 341
CLOSE command on 4 – 139

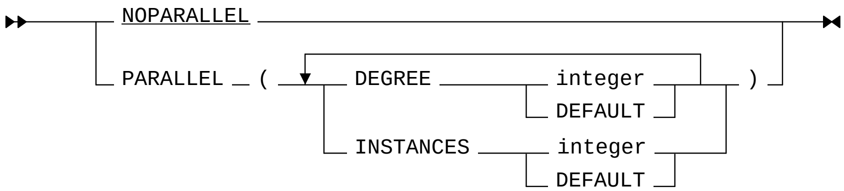
PARALLEL clause

Prerequisites

This clause can only be used in the following commands:

- ALTER CLUSTER
- ALTER DATABASE ... RECOVER
- ALTER INDEX ... REBUILD
- ALTER TABLE
- CREATE CLUSTER
- CREATE INDEX
- CREATE TABLE

Syntax



Keywords and Parameters

NOPARALLEL
: specifies serial execution of an operation. This is the default.

PARALLEL
: specifies parallel execution of an operation.

DEGREE
: determines the degree of parallelism for an operation on a single instance. That is, the number of query servers used in the parallel operation.

  integer
  : use *integer* query servers.

  DEFAULT
  : the number of query servers used is calculated from such things as the number of CPUs and the number of DEVICES storing tables to be scanned in parallel..

INSTANCES — determines the number of parallel server instances used in the parallel operation. This keyword is ignored if you do not have a parallel server.

integer	use *integer* instances
DEFAULT	use all available instances

Note: INSTANCES only applies to an instance using the Oracle7 Parallel Server.

Usage Notes

For more information on parallelized operations, see the “Parallel Query Option” chapter in *Oracle7 Server Tuning.*

Used in a CREATE command, the PARALLEL clause causes the creation of the object to be parallelized; if the CREATE command is CREATE TABLE, the PARALLEL clause sets the default degree of parallelism for queries on the table after creation.

Used in a command to alter an object, the PARALLEL clause changes the default degree of parallelism for queries on the object. In an ALTER DATABASE RECOVER command, the PARALLEL clause causes the recovery to be parallelized.

You cannot use the PARALLEL clause in an ALTER INDEX command unless you specify the REBUILD clause.

Specifying PARALLEL (DEGREE 1 INSTANCES 1) is equivalent to specifying NOPARALLEL.

A hint in a query can override a default of NOPARALLEL. Likewise, a hint in a query can override a default of PARALLEL.

CREATE SCHEMA

Although the PARALLEL clause syntax is allowed when creating a table, index or cluster in a CREATE SCHEMA statement, parallelism is not used and no error message is issued.

Example I

The following command creates a table using 10 query servers, 5 to scan scott.emp and another 5 to populate emp_dept:

```
CREATE TABLE emp_dept
    PARALLEL (DEGREE 5)
    AS SELECT * FROM scott.emp
            WHERE deptno = 10
```

Example II The following command creates an index using 10 query servers, 5 to scan scott.emp and another 5 to populate the emp_idx index:

```
CREATE INDEX emp_idx
    ON scott.emp (ename)
    PARALLEL 5
```

Example III The following command performs tablespace recovery using 5 recovery processes on 5 instances in a parallel server, for a total of 25 (5 * 5) query servers:

```
ALTER DATABASE
    RECOVER TABLESPACE binky
    PARALLEL (DEGREE 5 INSTANCES 5)
```

Example IV The following command changes the default number of query servers used to query the EMP table:

```
ALTER TABLE emp
    PARALLEL (DEGREE 9)
```

Example V The following command causes the index to be rebuilt from the existing index by using 6 query servers, 3 each to scan the old and to build the new index:

```
ALTER INDEX emp_idx
    REBUILD
    PARALLEL 3
```

Related Topics

ALTER CLUSTER command on page 4 – 16
ALTER DATABASE command on page 4 – 16
ALTER INDEX command on 4 – 33
ALTER TABLE command on 4 – 89
CREATE CLUSTER command on 4 – 164
CREATE INDEX command on 4 – 192
CREATE TABLE command on 4 – 245
Chapter “Parallel Query Option,” of *Oracle7 Server Tuning.*

PREPARE (Embedded SQL)

Purpose

To parse a SQL statement or PL / SQL block specified by a host variable and associate it with an identifier.

Prerequisites

None.

Syntax

```
>>- EXEC SQL PREPARE -+- statement_name -+- FROM -+- :host_string -+->◄
                      +- block_name -----+        +- 'text' -------+
```

Keywords and Parameters

statement_name
block_name
is the identifier to be associated with the prepared SQL statement or PL / SQL block. If this name has been previously assigned to another statement or block, the prior assignment is superseded.

:*host_string*
is a host variable whose value is the text of a SQL statement or PL / SQL block to be prepared.

'*text*'
is a string literal containing a SQL statement or PL / SQL block to be prepared.

Usage Notes

Any variables that appear in the :*host_string* or '*text*' are placeholders. The actual host variable names are assigned in the USING clause of the OPEN command (input host variables) or in the INTO clause of the FETCH command (output host variables).

A SQL statement is prepared only once, but can be executed any number of times.

Example

This example illustrates the use of a PREPARE statement in a Pro*C embedded SQL program:

```
EXEC SQL PREPARE my_statement FROM :my_string;
EXEC SQL EXECUTE my_statement
```

Related Topics

DECLARE CURSOR command on 4 – 280
OPEN command on 4 – 376
FETCH command on 4 – 341
CLOSE command on 4 – 139

RECOVER clause

Purpose

To perform media recovery.

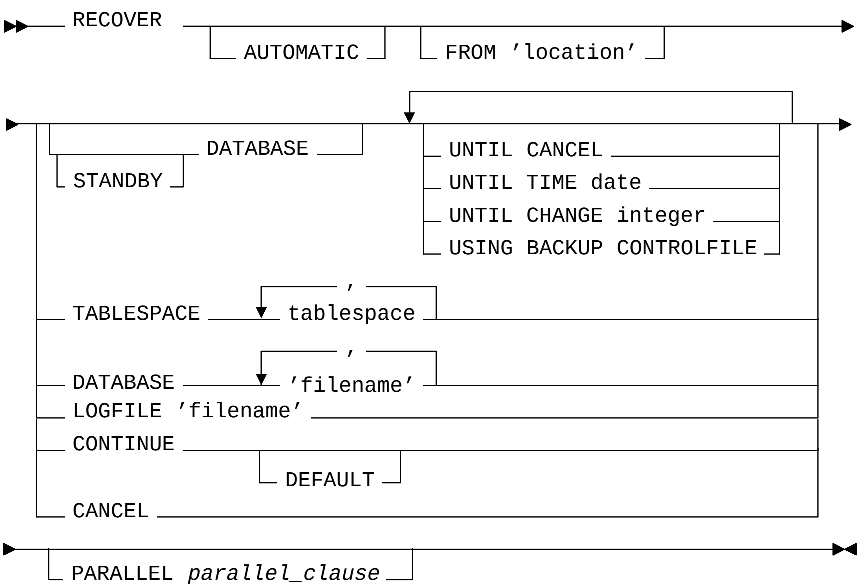
Prerequisites

The RECOVER clause must appear in an ALTER DATABASE statement. You must have the privileges necessary to issue this statement. For information on these privileges, see the ALTER DATABASE command on page 4 – 16.

You must also have the OSDBA role enabled. You cannot be connected to Oracle7 through the multi–threaded server architecture. Your instance must have the database mounted in exclusive mode.

> **Note:** It is recommended that you perform media recovery using Server Manager rather than using the ALTER DATABASE command with the RECOVER clause.

Syntax



Keywords and Parameters

AUTOMATIC
: automatically generates the names of the redo log files to apply during media recovery. If you omit this option, then you must specify the names of redo log files using the ALTER DATABASE ... RECOVER command with the LOGFILE clause.

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, Oracle7 assumes the archived redo log file group is in the location specified by the initialization parameter LOG_ARCHIVE_DEST.

STANDBY
: recovers the standby database using the controlfile and archived redo log files copied over from the primary database. For more information, see the *Oracle7 Server Administrator's Guide.*

DATABASE
: recovers the entire database. This is the default option. You can only use this option when the database is closed.

UNTIL CANCEL
: performs cancel–based recovery. This option recovers the database until you issue the ALTER DATABASE RECOVER command with the CANCEL clause.

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 before 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 offline.

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 offline.

LOGFILE — continues media recovery by applying the specified 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 Oracle7 has automatically generated.

CANCEL — terminates cancel–based recovery.

PARALLEL — specifies degree of parallelism to use when recovering. See *parallel_clause* on page 4 – 378.

Usage Notes

It is recommended that you use the Server Manager RECOVER command rather than the ALTER DATABASE command with the RECOVER clause to perform media recovery.

For most purposes, the RECOVER Server Manager command is easier to use than the ALTER DATABASE command. For information on this command, see *Oracle Server Manager User's Guide.*

For more information on media recovery, see the "Recovering a Database" chapter of *Oracle7 Server Administrator's Guide.*

You can use the ALTER DATABASE command with the RECOVER clause if you want to write your own specialized media recovery application using SQL.

Example I

The following statement performs complete recovery of the entire database:

```
ALTER DATABASE
    RECOVER AUTOMATIC DATABASE
```

Oracle7 automatically generates the names of redo log files to apply and prompts you with them. The following statement applies a suggested file:

```
ALTER DATABASE
    RECOVER CONTINUE DEFAULT
```

The following statement explicitly names a redo log file for Oracle7 to apply:

```
ALTER DATABASE
   RECOVER LOGFILE 'diska:arch0006.arc'
```

Example II The following statement performs time–based recovery of the database:

```
ALTER DATABASE AUTOMATIC
   RECOVER UNTIL TIME '1992–10–27:14:00:00'
```

Oracle7 recovers the database until 2:00pm on October 27, 1992.

Example III The following statement recovers the tablespace USER5:

```
ALTER DATABASE
   RECOVER TABLESPACE user5
```

Related Topics

ALTER DATABASE command on 4 – 16

RENAME

Purpose

To rename a table, view, sequence, or private synonym.

Prerequisites

The object must be in your own schema.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the object's creation label or you must satisfy one of the following 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 not comparable, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax

```
►►── RENAME old TO new ──────────────────────────────►◄
```

Keywords and Parameters

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.

Usage Notes

This command changes the name of a table, view, sequence, or private synonym for a table, view, or sequence. The new name must not already be used by another schema object in the same namespace and must follow the rules for naming schema objects defined in the section "Object Naming Rules" on page 2 – 3.

Integrity constraints, indexes, and grants on the old object are automatically transferred to the new object. Oracle7 invalidates all objects that depend on the renamed object, such as views, synonyms, and stored procedures and functions that refer to a renamed table.

You cannot use this command to rename public synonyms. To rename a public synonym, you must first drop it with the DROP SYNONYM command and then create another public synonym with the new name using the CREATE SYNONYM command.

You cannot use this command to rename columns. You can rename a column using the CREATE TABLE command with the AS clause. This example recreates the table STATIC, renaming a column from OLDNAME to NEWNAME:

```
CREATE TABLE temporary (newname, col2, col3)
   AS SELECT oldname, col2, col3 FROM static
DROP TABLE static
RENAME temporary TO static
```

Example

To change the name of table DEPT to EMP_DEPT:

```
RENAME dept TO emp_dept
```

Related Topics

CREATE SEQUENCE command on 4 – 224
CREATE SYNONYM command on 4 – 241
CREATE TABLE command on 4 – 245
CREATE VIEW command on 4 – 271

REVOKE (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) described in the next section of this chapter.

Prerequisites

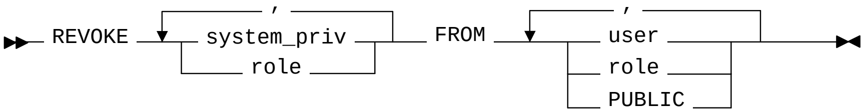
You must have been granted the system privilege or role with the ADMIN OPTION. Also, you can revoke any role if you have the GRANT ANY ROLE system privilege.

If you are using Trusted Oracle7 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 the following 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

It is recommended that you perform media recovery using the Server Manager RECOVER command rather than the ALTER DATABASE command with the RECOVER clause.

Syntax



Keywords and Parameters

system_priv — is a system privilege to be revoked. For a list of the system privileges, see Table 4 – 11 on page 4 – 351.

role — is a role to be revoked. For a list of the roles predefined by Oracle7, see Table 4 – 12 on page 4 – 352.

FROM — identifies users and roles from which the system privileges or roles are revoked.

PUBLIC	revokes the system privilege or role from all users.

Usage Notes

You can use this form of the REVOKE command to revoke both system privileges and roles from users, roles, and PUBLIC:

If you revoke a privilege from a user: Oracle7 removes the privilege from the user's privilege domain. Effective immediately, the user cannot exercise the privilege.

If you revoke a privilege from a role: Oracle7 removes the privilege from the role's privilege domain. Effective immediately, users with the role enabled cannot exercise the privilege. Also, other users who have been granted the role and subsequently enable the role cannot exercise the privilege.

If you revoke a privilege from PUBLIC: Oracle7 removes the privilege from the privilege domain of each user who has been granted the privilege through PUBLIC. Effective immediately, such users can no longer exercise the privilege. Note that the privilege is not revoked from users who have been granted the privilege directly or through roles.

If you revoke a role from a user: Oracle7 makes the role unavailable to the user. If the role is currently enabled for the user, the user can continue to exercise the privileges in the role's privilege domain as long as it remains enabled. However, the user cannot subsequently enable the role.

If you revoke a role from another role: Oracle7 removes the revoked role's privilege domain from the revokee role's privilege domain. Users who have been granted and have enabled the revokee role can continue to exercise the privileges in the revoked role's privilege domain as long as the revokee role remains enabled. However, other users who have been granted the revokee role and subsequently enable it cannot exercise the privileges in the privilege domain of the revoked role.

If you revoke a role from PUBLIC: Oracle7 makes the role unavailable to all users who have been granted the role through PUBLIC. Any user who has enabled the role can continue to exercise the privileges in its privilege domain as long as it remains enabled. However, users cannot subsequently enable the role. Note that the role is not revoked from users who have been granted the privilege directly or through other roles.

The REVOKE command can only revoke privileges and roles that have been granted directly with a GRANT statement. The REVOKE command cannot perform the following operations:

- revoke privileges or roles not granted to the revokee
- revoke roles granted through the operating system
- revoke privileges or roles granted to the revokee through roles

A system privilege or role cannot appear more than once in the list of privileges and roles to be revoked. A user, a role, or PUBLIC cannot appear more than once in the FROM clause.

Example I The following statement revokes DROP ANY TABLE system privilege from the users BILL and MARY:

```
REVOKE DROP ANY TABLE
    FROM bill, mary
```

BILL and MARY can no longer drop tables in schemas other than their own.

Example II The following statement revokes the role CONTROLLER from the user HANSON:

```
REVOKE controller
    FROM hanson
```

HANSON can no longer enable the CONTROLLER role.

Example III The following statement revokes the CREATE TABLESPACE system privilege from the CONTROLLER role:

```
REVOKE CREATE TABLESPACE
    FROM controller
```

Enabling the CONTROLLER role no longer allows users to create tablespaces.

Example IV To revoke the role VP from the role CEO, issue the following statement:

```
REVOKE vp
    FROM ceo
```

VP is no longer granted to CEO.

Related Topics

GRANT (System Privileges and Roles) command on 4 – 346
REVOKE (Object Privileges) command on 4 – 391

REVOKE (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) described in the previous section of this chapter.

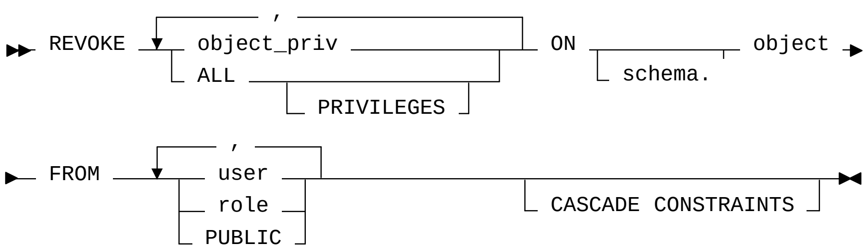
Prerequisites

You must have previously granted the object privileges to each user and role.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must match the label at which you granted the object privilege or you must satisfy one of the following 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

object_priv is an object privilege to be revoked. You can substitute any of the following 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 the following 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*, Oracle7 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.

Usage Notes

You can use this form of the REVOKE command to revoke object privileges from both users and roles:

If you revoke a privilege from a user: Oracle7 removes the privilege from the user's privilege domain. Effective immediately, the user cannot exercise the privilege.

If you revoke a privilege from a role: Oracle7 removes the privilege from the role's privilege domain. Effective immediately, users with the role enabled cannot exercise the privilege. Other users who have been granted the role cannot exercise the privilege after enabling the role.

If you revoke a privilege from PUBLIC: Oracle7 removes the privilege from the privilege domain of each user who has been granted the privilege through PUBLIC. Effective immediately, all such users are restricted from exercising the privilege. Note that the privilege is not revoked from users who have been granted the privilege directly or through roles.

You can only use the REVOKE command to revoke object privileges that you previously granted directly to the revokee. You cannot use the REVOKE command to perform the following operations:

- revoke object privileges that you did not grant to the revokee
- revoke privileges granted through the operating system
- revoke privileges granted to roles granted to the revokee

A privilege cannot appear more than once in the list of privileges to be revoked. A user, a role, or PUBLIC cannot appear more than once in the FROM clause.

Object Privileges

Each object privilege authorizes some operation on an object. By revoking an object privilege, you prevent the revokee from performing that operation. For a summary of the object privileges for each type of object, see Table 4 – 13 on page 4 – 357.

Revoking Multiple Identical Grants

Multiple users may grant the same object privilege to the same user, role, or PUBLIC. To remove the privilege from the grantee's privilege domain, all grantors must revoke the privilege. If even one grantor does not revoke the privilege, the grantee can still exercise the privilege by virtue of that grant.

Cascading Revokes

Revoking an object privilege that a user has either granted or exercised to define an object or a referential integrity constraint has the following cascading effects:

- If you revoke an object privilege from a user who has granted the privilege to other users or roles, Oracle7 also revokes the privilege from the grantees.
- If you revoke an object privilege from a user whose schema contains a procedure, function, or package that contains SQL statements that exercise the privilege, the procedure, function, or package can no longer be executed.
- If you revoke an object privilege on an object from a user whose schema contains a view on that object, Oracle7 invalidates the view.
- If you revoke REFERENCES privilege from a user who has exercised the privilege to define referential integrity constraints, you must specify the CASCADE CONSTRAINTS option. Oracle7 then revokes the privilege and drops the constraints.

Example I

You can grant DELETE, INSERT, SELECT, and UPDATE privileges on the table BONUS to the user PEDRO with the following statement:

```
GRANT ALL
    ON bonus
    TO pedro
```

To revoke DELETE privilege on BONUS from PEDRO, issue the following statement:

```
REVOKE DELETE
    ON bonus
    FROM pedro
```

To revoke the remaining privileges on BONUS that you granted to PEDRO, issue the following statement:

```
REVOKE ALL
    ON bonus
    FROM pedro
```

Example II You can grant SELECT and UPDATE privileges on the view REPORTS to all users by granting the privileges to the role PUBLIC:

```
GRANT SELECT, UPDATE
   ON reports
   TO public
```

The following statement revokes UPDATE privilege on REPORTS from all users:

```
REVOKE UPDATE
   ON reports
   FROM public
```

Users can no longer update the REPORTS view, although users can still query it. However, if you have also granted UPDATE privilege on REPORTS to any users (either directly or through roles), these users retain the privilege.

Example III You can grant the user BLAKE SELECT privilege on the ESEQ sequence in the schema ELLY with the following statement:

```
GRANT SELECT
   ON elly.eseq
   TO blake
```

To revoke SELECT privilege on ESEQ from BLAKE, issue the following statement:

```
REVOKE SELECT
   ON elly.eseq
   FROM blake
```

However, if the user ELLY has also granted SELECT privilege on ESEQ to BLAKE, BLAKE can still use ESEQ by virtue of ELLY's grant.

Example IV You can grant BLAKE the privileges REFERENCES and UPDATE on the EMP table in the schema SCOTT with the following statement:

```
GRANT REFERENCES, UPDATE
    ON scott.emp
    TO blake
```

BLAKE can exercise the REFERENCES privilege to define a constraint in his own DEPENDENT table that refers to the EMP table in the schema SCOTT:

```
CREATE TABLE dependent
(dependno    NUMBER,
 dependname  VARCHAR2(10),
 employee    NUMBER
             CONSTRAINT in_emp REFERENCES scott.emp(ename) )
```

You can revoke REFERENCES privilege on SCOTT.EMP from BLAKE, by issuing the following statement that contains the CASCADE CONSTRAINTS option:

```
REVOKE REFERENCES
    ON scott.emp
    FROM blake
    CASCADE CONSTRAINTS
```

Revoking BLAKE's REFERENCES privilege on SCOTT.EMP causes Oracle7 to drop the IN_EMP constraint because BLAKE required the privilege to define the constraint.

However, if BLAKE has also been granted REFERENCES privilege on SCOTT.EMP by a user other than you, Oracle7 does not drop the constraint. BLAKE still has the privilege necessary for the constraint by virtue of the other user's grant.

Related Topics

GRANT (Object Privileges) command on 4 – 355
REVOKE (System Privileges and Roles) command on 4 – 388

ROLLBACK

Purpose

To undo work done in the current transaction.

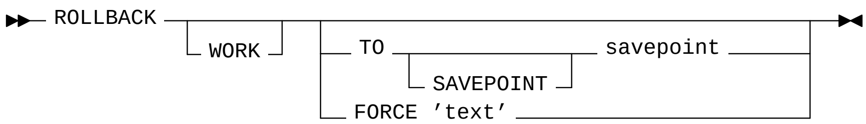
You can also use this command to manually undo the work done by an in–doubt distributed transaction.

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.

Syntax



Keywords and Parameters

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.

Usage Notes

A transaction (or a logical unit of work) is a sequence of SQL statements that Oracle7 treats as a single unit. A transaction begins with the first executable SQL statement after a COMMIT, ROLLBACK or connection to the database. A transaction ends with a COMMIT statement, a ROLLBACK statement, or disconnection (intentional or unintentional) from the database. Note that Oracle7 issues an implicit COMMIT statement before and after processing any Data Definition Language statement.

Using the ROLLBACK command without the TO SAVEPOINT clause performs the following operations:

- ends the transaction
- undoes all changes in the current transaction
- erases all savepoints in the transaction
- releases the transaction's locks

Using the ROLLBACK command with the TO SAVEPOINT clause performs the following operations:

- rolls back just the portion of the transaction after the savepoint.
- loses all savepoints created after that savepoint. Note that the named savepoint is retained, so you can roll back to the same savepoint multiple times. Prior savepoints are also retained.
- releases all table and row locks acquired since the savepoint. Note that other transactions that have requested access to rows locked after the savepoint must continue to wait until the transaction is committed or rolled back. Other transactions that have not already requested the rows can request and access the rows immediately.

It is recommended that you explicitly end transactions in application programs using either a COMMIT or ROLLBACK statement. If you do not explicitly commit the transaction and the program terminates abnormally, Oracle7 rolls back the last uncommitted transaction.

Example I The following statement rolls back your entire current transaction:

```
ROLLBACK
```

Example II The following statement rolls back your current transaction to savepoint SP5:

```
ROLLBACK TO SAVEPOINT sp5
```

Distributed Transactions

Oracle7 with the distributed option allows you to perform distributed transactions, or transactions that modify data on multiple databases. To commit or roll back a distributed transaction, you need only issue a COMMIT or ROLLBACK statement as you would any other transaction.

If there is a network failure during the commit process for a distributed transaction, the state of the transaction may be unknown, or in–doubt. After consultation with the administrators of the other databases involved in the transaction, you may decide to manually commit or roll back the transaction on your local database. You can manually roll back the transaction on your local database by issuing a ROLLBACK statement with the FORCE clause.

For more information on when to roll back in–doubt transactions, see *Oracle7 Server Distributed Systems, Volume I.*

You cannot manually roll back an in–doubt transaction to a savepoint.

A ROLLBACK statement with a FORCE clause only rolls back the specified transaction. Such a statement does not affect your current transaction.

Example III The following statement manually rolls back an in–doubt distributed transaction:

```
ROLLBACK WORK
    FORCE '25.32.87'
```

Related Topics

COMMIT command on 4 – 141
SAVEPOINT command on 4 – 404
SET TRANSACTION command on 4 – 445

ROLLBACK (Embedded SQL)

Purpose

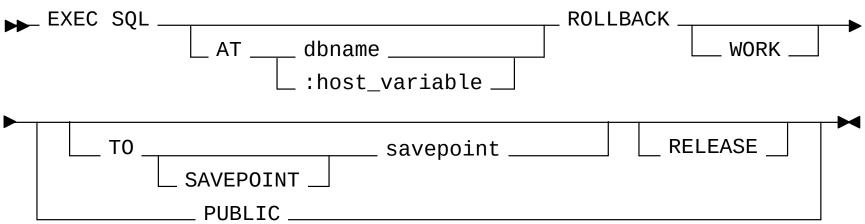
To end the current transaction, discard all changes in the current transaction, and release all locks and optionally release resources and disconnect from the database.

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.

Syntax



Keywords and Parameters

AT — identifies the database to which the ROLLBACK statement is issued. The database can be identified by either:

db_name	is a database identifier declared in a previous DECLARE DATABASE statement.
:*host_variable*	is a host variable whose value is a previously declared *db_name.*

If you omit this clause, the statement is issued to your default database.

WORK — is optional and has no effect on ROLLBACK.

TO — rolls back the transaction to a previously declared savepoint.

RELEASE — releases all resources and disconnects you from the database.

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.

Usage Notes

Always explicitly commit or rollback the last transaction in a program using the RELEASE option to disconnect from Oracle.

Oracle7 automatically rolls back your current transaction if the program terminates abnormally.

The ROLLBACK command has no effect on the contents of the host variables or on the control flow of the program.

Example

This example illustrates the use of the embedded SQL ROLLBACK command:

```
EXEC SQL ROLLBACK TO SAVEPOINT point4
```

Related Topics

COMMIT command on 4 – 141
DECLARE DATABASE command on 4 – 282
ROLLBACK command on 4 – 397
SAVEPOINT command on 4 – 404
SET TRANSACTION command on 4 – 445

SAVEPOINT

Purpose

To identify a point in a transaction to which you can later roll back.

Prerequisites

None.

Syntax

```
▶▶── SAVEPOINT savepoint ──────────────────────────────▶◀
```

Keywords and Parameters

savepoint is the name of the savepoint to be created.

Usage Notes

Savepoints are used with the ROLLBACK command to rollback portions of the current transaction.

Savepoints are useful in interactive programs, because you can create and name intermediate steps of a program. This allows you more control over longer, more complex programs. For example, you can use savepoints throughout a long complex series of updates, so that if you make an error, you need not resubmit every statement.

Savepoints are useful in application programs in a similar way. If a program contains several subprograms, you can create a savepoint before each subprogram begins. If a subprogram fails, it is easy to return the data to its state before the subprogram began and then re–execute the subprogram with revised parameters or perform a recovery action.

Savepoint names must be distinct within a given transaction. If you create a second savepoint with the same identifier as an earlier savepoint, the earlier savepoint is erased. After a savepoint has been created, you can either continue processing, commit your work, rollback the entire transaction, or rollback to the savepoint.

Transaction

A transaction (or a logical unit of work) is a sequence of SQL statements that Oracle7 treats as a single unit. A transaction begins with the first executable SQL statement after a COMMIT, ROLLBACK or connection to Oracle. A transaction ends with a COMMIT statement, a ROLLBACK statement, or disconnection (intentional or unintentional) from Oracle. Oracle7 issues an implicit COMMIT before and after any Data Definition Language statement.

Example

To update BLAKE's and CLARK's salary, check that the total company salary does not exceed 20,000, then re–enter CLARK's salary, enter:

```
UPDATE emp
    SET sal = 2000
    WHERE ename = 'BLAKE'
SAVEPOINT blake_sal
UPDATE emp
    SET sal = 1500
    WHERE ename = 'CLARK'
SAVEPOINT clark_sal
SELECT SUM(sal) FROM emp
ROLLBACK TO SAVEPOINT blake_sal
UPDATE emp
    SET sal = 1300
    WHERE ename = 'CLARK'
COMMIT
```

Related Topics

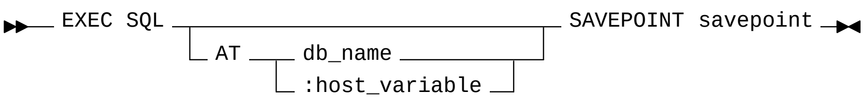
COMMIT command on 4 – 141
ROLLBACK command on 4 – 397
SET TRANSACTION command on 4 – 445

SAVEPOINT (Embedded SQL)

Purpose To identify a point in a transaction to which you can later roll back.

Prerequisites None.

Syntax



Keywords and Parameters

AT identifies the database on which the savepoint is created. The database can be identified by either:

db_name is a database identifier declared in a previous DECLARE DATABASE statement.

:*host_variable* is a host variable whose value is a previously declared *db_name.*

If you omit this clause, the savepoint is created on your default database.

savepoint is the name of the savepoint to be created.

Usage Notes For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Example This example illustrates the use of the embedded SQL SAVEPOINT command:

```
EXEC SQL SAVEPOINT save3
```

Related Topics

COMMIT command on 4 – 141
ROLLBACK command on 4 – 397
SAVEPOINT command on 4 – 404

SELECT

Purpose

To retrieve data from one or more tables, views, or snapshots.

Prerequisites

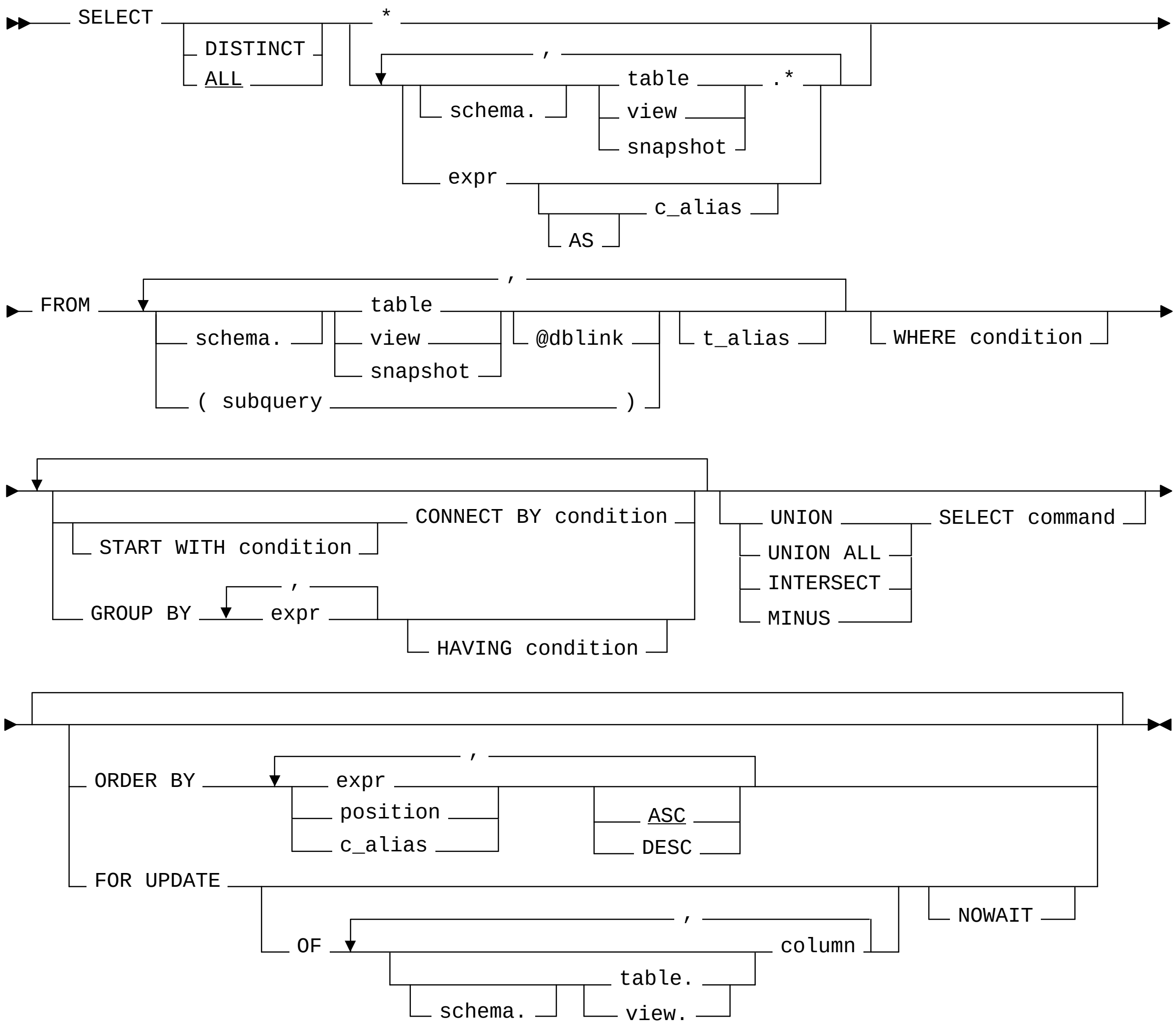
For you to select data from a table or snapshot, the table or snapshot must be in your own schema or you must have SELECT privilege on the table or snapshot.

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 Oracle7 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.

Syntax



SELECT
DISTINCT
ALL
*
,
schema.
table
view
snapshot
.*
expr
AS
c_alias
FROM
,
schema.
table
view
snapshot
@dblink
t_alias
WHERE condition
(subquery
)
START WITH condition
CONNECT BY condition
GROUP BY
,
expr
HAVING condition
UNION
UNION ALL
INTERSECT
MINUS
SELECT command
ORDER BY
,
expr
position
c_alias
ASC
DESC
FOR UPDATE
OF
,
schema.
table.
view.
column
NOWAIT

Keywords and Parameters

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. See the syntax description of *expr* on page 3 – 73. A column name in this list can only 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. The AS keyword is optional. The alias effectively renames the select list item for the duration of the query. The alias can be used in the ORDER BY clause, but not other clauses in the query.
schema	is the schema containing the selected table, view, or snapshot. If you omit *schema*, Oracle7 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 the complete or partial name for a database link to a remote database where the table, view, or snapshot is located. For more information on referring to database links, see the section “Referring to Objects in Remote Databases” on page 2 – 11. Note that this database need not be an Oracle7 database.

If you omit *dblink*, Oracle7 assumes that the table, view, or snapshot is on the local database.

subquery	is a subquery that is treated in the same manner as a view. For the syntax of subquery, see page 4 – 436. Oracle7 executes the subquery and then uses the resulting rows as a view in the FROM clause.
t_alias	provides a different name for the table, view, snapshot, or subquery for 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, Oracle7 returns all rows from the tables, views, or snapshots in the FROM clause. See the syntax description of *condition* on page 3 – 78.
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, Oracle7 returns summary rows for all groups. See the syntax description of *expr* on page 3 – 73 and the syntax description of *condition* on page 3 – 78.
UNION UNION ALL INTERSECT MINUS	combines the rows returned by two SELECT statement using a set operation. To reference a column, you must use an alias to name the column. The FOR UPDATE clause cannot be used with these set operators.

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.

FOR UPDATE — locks the selected rows.

OF — Only lock the select rows for a particular table in a join.

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, Oracle7 waits until the row is available and then returns the results of the SELECT statement.

Usage Notes

The list of expressions that appears after the SELECT keyword and before the FROM clause is called the *select list*. Each *expr* becomes the name of one column in the set of returned rows, and each *table*.* becomes a set of columns, one for each column in the table in the order they were defined when the table was created. The datatype and length of each expression is determined by the elements of the expression.

If two or more tables have some column names in common, you must qualify column names with names of tables. Otherwise, fully qualified column names are optional, although it is always better to explicitly qualify table and column references. Oracle7 often does less work with fully qualified table and column names.

You can use a column alias, *c_alias*, to label the preceding expression in the select list so that the column is displayed with a new heading. The alias effectively renames the select list item for the duration of the query. The alias can be used in the ORDER BY clause, but not other clauses in the query.

If you use the DISTINCT option to return only a single copy of duplicate rows, the total number of bytes in all select list expressions is limited to the size of a data block minus some overhead. This size is specified by the initialization parameter DB_BLOCK_SIZE.

You can use comments in a SELECT statement to pass instructions, or *hints*, to the Oracle7 optimizer. The optimizer uses hints to choose an execution plan for the statement. For more information on hints, see *Oracle7 Server Tuning*.

Example I

The following statement selects rows from the employee table with the department number of 40:

```
SELECT *
    FROM emp
    WHERE deptno = 40
```

Example II

The following statement selects the name, job, salary and department number of all employees except salesmen from department number 30:

```
SELECT ename, job, sal, deptno
    FROM emp
    WHERE NOT (job = ’SALESMAN’ AND deptno = 30)
```

Example III

The following statement selects from subqueries in the FROM clause and gives departments total employees and salaries as a percentage of all the departments:

```
SELECT a.deptno ”Department”,
       a.num_emp/b.total_count ”%Employees”,
       a.sal_sum/b.total_sal ”%Salary”
  FROM
 (SELECT deptno, COUNT(*) num_emp, SUM(SAL) sal_sum
    FROM scott.emp
    GROUP BY deptno) a,
 (SELECT COUNT(*) total_count, SUM(sal) total_sal
    FROM scott.emp) b ;
```

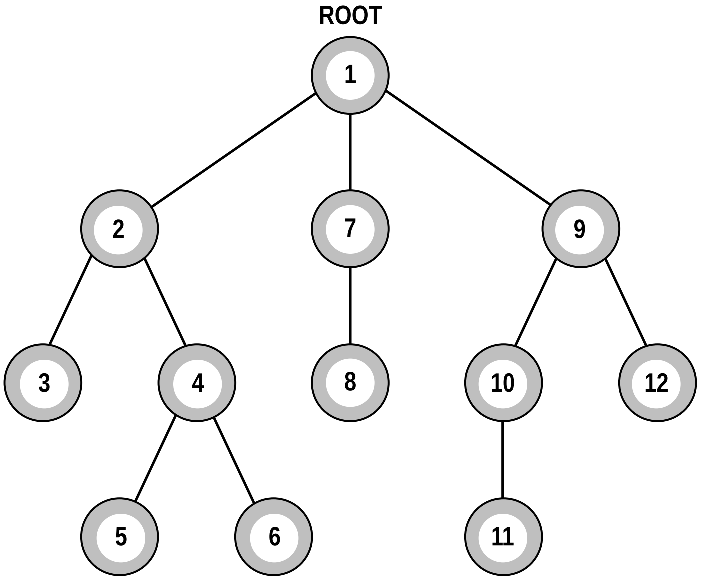
Hierarchical Queries

If a table contains hierarchical data, you can select rows in a hierarchical order using the following clauses:

START WITH	You can specify the root row(s) of the hierarchy using this clause.
CONNECT BY	You can specify the relationship between parent rows and child rows of the hierarchy using this clause.
WHERE	You can restrict the rows returned by the query without affecting other rows of the hierarchy using this clause.

Oracle7 uses the information from the above clause to form the hierarchy using the following steps:

1. Oracle7 selects the root row(s) of the hierarchy. These are the rows that satisfy the condition of the START WITH clause.
2. Oracle7 selects the child rows of each root row. Each child row must satisfy the condition of the CONNECT BY clause with respect to one of the root rows.
3. Oracle7 selects successive generations of child rows. Oracle7 first selects the children of the rows returned in step 2, and then the children of those children, and so on. Oracle7 always selects children by evaluating the CONNECT BY condition with respect to a current parent row.
4. If the query contains a WHERE clause, Oracle7 removes all rows from the hierarchy that do not satisfy the condition of the WHERE clause. Oracle7 evaluates this condition for each row individually, rather than removing all the children of a row that does not satisfy the condition.
5. Oracle7 returns the rows in the order shown in this diagram. In the diagram children appear below their parents.



SELECT statements performing hierarchical queries are subject to the following restrictions:

- A SELECT statement that performs a hierarchical query cannot also perform a join. A SELECT statement that performs a hierarchical query cannot select data from a view whose query performs a join.
- If you use the ORDER BY clause in a hierarchical query, Oracle7 orders rows by the ORDER BY clause, rather than in the order shown in step 5.

The following sections discuss the START WITH and CONNECT BY clauses.

START WITH Clause

The START WITH clause identifies the row(s) to be used as the root(s) of a hierarchical query. This clause specifies a condition that the roots must satisfy. If you omit this clause, Oracle7 uses all rows in the table as root rows. A START WITH condition can contain a subquery.

CONNECT BY Clause

The CONNECT BY clause specifies the relationship between parent and child rows in a hierarchical query. This clause contains a condition that defines this relationship. This condition can be any condition as defined by the syntax description of *condition* on page 3 – 78; however, some part of the condition must use the PRIOR operator to refer to the parent row. The part of the condition containing the PRIOR operator must have one of the following forms:

```
PRIOR expr comparison_operator expr
expr comparison_operator PRIOR expr
```

To find the children of a parent row, Oracle7 evaluates the PRIOR expression for the parent row and the other expression for each row in the table. Rows for which the condition is true are the children of the parent. The CONNECT BY clause can contain other conditions to further filter the rows selected by the query. The CONNECT BY clause cannot contain a subquery.

If the CONNECT BY clause results in a loop in the hierarchy, Oracle7 returns an error. A loop occurs if one row is both the parent (or grandparent or direct ancestor) and a child (or a grandchild or a direct descendent) of another row.

Example IV The following CONNECT BY clause defines a hierarchical relationship in which the EMPNO value of the parent row is equal to the MGR value of the child row:

```
CONNECT BY PRIOR empno = mgr
```

Example V In the following CONNECT BY clause, the PRIOR operator applies only to the EMPNO value. To evaluate this condition, Oracle7 evaluates EMPNO values for the parent row and MGR, SAL, and COMM values for the child row:

```
CONNECT BY PRIOR empno = mgr AND sal > comm
```

To qualify as a child row, a row must have a MGR value equal to the EMPNO value of the parent row and it must have a SAL value greater than its COMM value.

The LEVEL Pseudocolumn

SELECT statements that perform hierarchical queries can use the LEVEL pseudocolumn. LEVEL returns the value 1 for a root node, 2 for a child node of a root node, 3 for a grandchild, etc. For more information on LEVEL, see the section "Pseudocolumns" on page 2 – 38.

The number of levels returned by a hierarchical query may be limited by available user memory.

Example VI

The following statement returns all employees in hierarchical order. The root row is defined to be the employee whose job is 'PRESIDENT'. The child rows of a parent row are defined to be those who have the employee number of the parent row as their manager number.

```
SELECT LPAD(' ',2*(LEVEL-1)) || ename org_chart,
       empno, mgr, job
    FROM emp
    START WITH job = 'PRESIDENT'
    CONNECT BY PRIOR empno = mgr

ORG_CHART         EMPNO        MGR JOB
------------ ---------- ---------- ---------
KING               7839            PRESIDENT
  JONES            7566       7839 MANAGER
    SCOTT          7788       7566 ANALYST
      ADAMS        7876       7788 CLERK
    FORD           7902       7566 ANALYST
      SMITH        7369       7902 CLERK
  BLAKE            7698       7839 MANAGER
    ALLEN          7499       7698 SALESMAN
    WARD           7521       7698 SALESMAN
    MARTIN         7654       7698 SALESMAN
    TURNER         7844       7698 SALESMAN
    JAMES          7900       7698 CLERK
  CLARK            7782       7839 MANAGER
    MILLER         7934       7782 CLERK
```

The following statement is similar to the previous one, except that it does not select employees with the job 'ANALYST'.

```
SELECT LPAD(' ',2*(LEVEL-1)) || ename org_chart,
       empno, mgr, job
    FROM emp
    WHERE job != 'ANALYST'
    START WITH job = 'PRESIDENT'
    CONNECT BY PRIOR empno = mgr

ORG_CHART          EMPNO        MGR JOB
------------- ---------- ---------- ---------
KING                7839            PRESIDENT
JONES               7566       7839 MANAGER
ADAMS               7876       7788 CLERK
SMITH               7369       7902 CLERK
BLAKE               7698       7839 MANAGER
ALLEN               7499       7698 SALESMAN
WARD                7521       7698 SALESMAN
MARTIN              7654       7698 SALESMAN
TURNER              7844       7698 SALESMAN
JAMES               7900       7698 CLERK
CLARK               7782       7839 MANAGER
```

Oracle7 does not return the analysts SCOTT and FORD, although it does return employees who are managed by SCOTT and FORD.

The following statement is similar to the first one, except that it uses the LEVEL pseudocolumn to select only the first two levels of the management hierarchy:

```
SELECT LPAD(' ',2*(LEVEL-1)) || ename org_chart,
empno, mgr, job
    FROM emp
    START WITH job = 'PRESIDENT'
    CONNECT BY PRIOR empno = mgr AND LEVEL <= 2

ORG_CHART         EMPNO        MGR JOB
------------ ---------- ---------- ---------
KING               7839            PRESIDENT
  JONES            7566       7839 MANAGER
  BLAKE            7698       7839 MANAGER
  CLARK            7782       7839 MANAGER
```

GROUP BY Clause

You can use the GROUP BY clause to group selected rows and return a single row of summary information. Oracle7 collects each group of rows based on the values of the expression(s) specified in the GROUP BY clause.

If a SELECT statement contains the GROUP BY clause, the select list can only contain the following types of expressions:

- constants
- group functions
- the functions USER, UID, and SYSDATE
- expressions identical to those in the GROUP BY clause
- expressions involving the above expressions that evaluate to the same value for all rows in a group

Expressions in the GROUP BY clause can contain any columns in the tables, views, and snapshots in the FROM clause regardless of whether the columns appear in the select list.

The total number of bytes in all expressions in the GROUP BY clause is limited to the size of a data block minus some overhead. This size is specified by the initialization parameter DB_BLOCK_SIZE.

Example VII To return the minimum and maximum salaries for each department in the employee table, issue the following statement:

```
SELECT deptno, MIN(sal), MAX(sal)
    FROM emp
    GROUP BY deptno

    DEPTNO   MIN(SAL)   MAX(SAL)
---------- ---------- ----------
        10         10       5004
        20        804       3004
        30        954       2854
```

Example VIII To return the minimum and maximum salaries for the clerks in each department, issue the following statement:

```
SELECT deptno, MIN(sal), MAX(sal)
    FROM emp
    WHERE job = 'CLERK'
    GROUP BY deptno

    DEPTNO   MIN(SAL)   MAX(SAL)
---------- ---------- ----------
        10       1304       1304
        20        804       1104
        30        954        954
```

HAVING Clause

You can use the HAVING clause to restrict which groups of rows defined by the GROUP BY clause are returned by the query. Oracle7 processes the WHERE, GROUP BY, and HAVING clauses in the following manner:

1. If the statement contains a WHERE clause, Oracle7 removes all rows that do not satisfy it.
2. Oracle7 calculates and forms the groups as specified in the GROUP BY clause.
3. Oracle7 removes all groups that do not satisfy the HAVING clause.

Specify the GROUP BY and HAVING clauses after the WHERE and CONNECT BY clauses. If both the GROUP BY and HAVING clauses are specified, they can appear in either order.

Example IX To return the minimum and maximum salaries for the clerks in each department whose lowest salary is below $1,000, issue the following statement:

```
SELECT deptno, MIN(sal), MAX(sal)
    FROM emp
    WHERE job = 'CLERK'
    GROUP BY deptno
    HAVING MIN(sal) < 1000

    DEPTNO   MIN(SAL)   MAX(SAL)
---------- ---------- ----------
        20        804       1104
        30        954        954
```

Set Operators

The UNION, UNION ALL, INTERSECT, and MINUS operators combine the results of two queries into a single result. The number and datatypes of the columns selected by each component query must be the same, but the column lengths can be different. For information on the use of each set operator, see the section "Set Operators" on page 3 – 12.

If more than two queries are combined with set operators, adjacent pairs of queries are evaluated from left to right. You can use parentheses to specify a different order of evaluation.

The total number of bytes in all select list expressions of a component query is limited to the size of a data block minus some overhead. The size of a data block is specified by the initialization parameter DB_BLOCK_SIZE.

ORDER BY Clause

Without an ORDER BY clause, it is not guaranteed that the same query executed more than once will retrieve rows in the same order. You use the ORDER BY clause to order the rows selected by a query. The clause specifies either expressions or positions or aliases of expressions in the select list of the statement. Oracle7 returns rows based on their values for these expressions.

You can specify multiple expressions in the ORDER BY clause. Oracle7 first sorts rows based on their values for the first expression. Rows with the same value for the first expression are then sorted based on their values for the second expression, and so on. Oracle7 sorts nulls following all others in ascending order and preceding all others in descending order.

Sorting by position is useful in the following cases:

- To order by a lengthy select list expression, you can specify its position, rather than duplicate the entire expression, in the ORDER BY clause.
- For compound queries (containing set operators UNION, INTERSECT, MINUS, or UNION ALL), the ORDER BY clause must use positions, rather than explicit expressions. Also, the ORDER BY clause can only appear in the last component query. The ORDER BY clause orders all rows returned by the entire compound query.

The mechanism by which Oracle7 sorts values for the ORDER BY clause is specified either explicitly by the NLS_SORT initialization parameter or implicitly by the NLS_LANGUAGE initialization parameter. For information on these parameters, see the "National Language Support" chapter of *Oracle7 Server Reference*. You can also change the sort mechanism dynamically from one linguistic sort sequence to another using the ALTER SESSION command. You can also specify a specific sort sequence for a single query by using the NLSSORT function with the NLS_SORT parameter in the ORDER BY clause.

The ORDER BY clause is subject to the following restrictions:

- If the ORDER BY clause and the DISTINCT operator both appear in a SELECT statement, the ORDER BY clause cannot refer to columns that do not appear in the select list.
- The ORDER BY clause cannot appear in subqueries within other statements.
- The total number of bytes in all expressions in the ORDER BY clause is limited to the size of a data block minus some overhead. The size of a data block is specified by the initialization parameter DB_BLOCK_SIZE.

If you use the ORDER BY and GROUP BY clauses together, the expressions that can appear in the ORDER BY clause are subject to the same restrictions as the expressions in the select list, described in section "GROUP BY Clause" on page 4 – 416.

If you use the ORDER BY clause in a hierarchical query, Oracle7 uses the ORDER BY clause rather than the hierarchy to order the rows.

Example X To select all salesmen's records from EMP, and order the results by commission in descending order, issue the following statement:

```
SELECT *
   FROM emp
   WHERE job = 'SALESMAN'
   ORDER BY comm DESC
```

Example XI To select the employees from EMP ordered first by ascending department number and then by descending salary, issue the following statement:

```
SELECT ename, deptno, sal
    FROM emp
    ORDER BY deptno ASC, sal DESC
```

To select the same information as the previous SELECT and use the positional ORDER BY notation, issue the following statement:

```
SELECT ename, deptno, sal
    FROM emp
    ORDER BY 2 ASC, 3 DESC
```

FOR UPDATE Clause

The FOR UPDATE clause locks the rows selected by the query. Once you have selected a row for update, other users cannot lock or update it until you end your transaction. The FOR UPDATE clause signals that you intend to insert, update, or delete the rows returned by the query, but does not require that you perform one of these operations. A SELECT statement with a FOR UPDATE clause is often followed by one or more UPDATE statements with WHERE clauses.

The FOR UPDATE clause cannot be used with the following other constructs:

- DISTINCT operator
- GROUP BY clause
- set operators
- group functions

The tables locked by the FOR UPDATE clause must all be located on the same database. These locked tables must also be on the same database as any LONG columns and sequences referenced in the same statement.

If a row selected for update is currently locked by another user, Oracle7 waits until the row is available, locks it, and then returns control to you. You can use the NOWAIT option to cause Oracle7 to terminate the statement without waiting if such a row is already locked.

FOR UPDATE OF Note that the columns in OF clause only specify which tables' rows are locked. The specific columns of the table that you specify are not significant. If you omit the OF clause, Oracle7 locks the selected rows from all the tables in the query.

Example XII The following statement locks rows in the EMP table with clerks located in New York and locks rows in the DEPT table with departments in New York that have clerks:

```
SELECT empno, sal, comm
    FROM emp, dept
    WHERE job = 'CLERK'
      AND emp.deptno = dept.deptno
      AND loc = 'NEW YORK'
    FOR UPDATE
```

Example XIII The following statement only locks rows in the EMP table with clerks located in New York; no rows are locked in the DEPT table:

```
SELECT empno, sal, comm
    FROM emp, dept
    WHERE job = 'CLERK'
      AND emp.deptno = dept.deptno
      AND loc = 'NEW YORK'
    FOR UPDATE OF emp
```

Joins

A *join* is a query that combines rows from two or more tables, views, or snapshots. Oracle7 performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

Join Conditions

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a *join condition*. To execute a join, Oracle7 combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list.

To execute a join of three or more tables, Oracle7 first joins two of the tables based on the join conditions comparing their columns and then joins the result to another table based on join conditions containing columns of the joined tables and the new table. Oracle7 continues this process until all tables are joined into the result. The optimizer determines the order in which Oracle7 joins tables based on the join conditions, indexes on the tables, and, in the case of the cost–based optimization approach, statistics for the tables.

In addition to join conditions, the WHERE clause of a join query can also contain other conditions that refer to columns of only one table. These conditions can further restrict the rows returned by the join query.

Equijoins

An *equijoin* is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. Depending on the internal algorithm the optimizer chooses to execute the join, the total size of the columns in the equijoin condition in a single table may be limited to the size of a data block minus some overhead. The size of a data block is specified by the initialization parameter DB_BLOCK_SIZE.

Example XIV This equijoin returns the name and job of each employee and the number and name of the department in which the employee works:

```
SELECT ename, job, dept.deptno, dname
    FROM emp, dept
    WHERE emp.deptno = dept.deptno

ENAME      JOB       DEPTNO     DNAME
---------- --------- ---------- --------------
KING       PRESIDENT         10 ACCOUNTING
BLAKE      MANAGER           30 SALES
CLARK      MANAGER           10 ACCOUNTING
JONES      MANAGER           20 RESEARCH
FORD       ANALYST           20 RESEARCH
SMITH      CLERK             20 RESEARCH
ALLEN      SALESMAN          30 SALES
WARD       SALESMAN          30 SALES
MARTIN     SALESMAN          30 SALES
SCOTT      ANALYST           20 RESEARCH
TURNER     SALESMAN          30 SALES
ADAMS      CLERK             20 RESEARCH
JAMES      CLERK             30 SALES
MILLER     CLERK             10 ACCOUNTING
```

You must use a join to return this data because employee names and jobs are stored in a different table than department names. Oracle7 combines rows of the two tables according to this join condition:

```
emp.deptno = dept.deptno
```

Example XV The following equijoin returns the name, job, department number, and department name of all clerks:

```
SELECT ename, job, dept.deptno, dname
    FROM emp, dept
    WHERE emp.deptno = dept.deptno
      AND job = 'CLERK'

ENAME      JOB       DEPTNO     DNAME
---------- --------- ---------- --------------
SMITH      CLERK             20 RESEARCH
ADAMS      CLERK             20 RESEARCH
JAMES      CLERK             30 SALES
MILLER     CLERK             10 ACCOUNTING
```

This query is identical to Example XII except that it uses an additional WHERE clause condition to return only rows with a JOB value of 'CLERK':

```
job = 'CLERK'
```

Self Joins

A *self join* is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that are used to qualify column names in the join condition. To perform a self join, Oracle7 combines and returns rows of the table that satisfy the join condition.

Example XVI This query uses a self join to returns the name of each employee along with the name of the employee's manager:

```
SELECT e1.ename||' works for '||e2.ename
"Employees and their Managers"
    FROM emp e1, emp e2    WHERE e1.mgr = e2.empno

Employees and their Managers
-------------------------------
BLAKE works for KING
CLARK works for KING
JONES works for KING
FORD works for JONES
SMITH works for FORD
ALLEN works for BLAKE
WARD works for BLAKE
MARTIN works for BLAKE
SCOTT works for JONES
TURNER works for BLAKE
ADAMS works for SCOTT
JAMES works for BLAKE
MILLER works for CLARK
```

The join condition for this query uses the aliases E1 and E2 for the EMP table:

```
e1.mgr = e2.empno
```

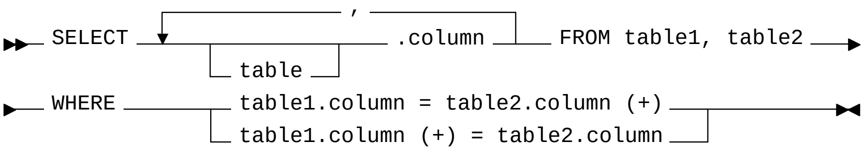
Cartesian Products

If two tables in a join query have no join condition, Oracle7 returns their Cartesian product. Oracle7 combines each row of one table with each row of the other. A Cartesian product always generates many rows and is rarely useful. For example, the Cartesian product of two tables each with a hundred rows has ten thousand rows. Always include a join condition unless you specifically need a Cartesian product. If a query joins three or more tables and there is no join condition for a specific pair, the optimizer may choose a join order that avoids producing an intermediate Cartesian product.

Outer Joins

The outer join extends the result of a simple join. An *outer join* returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition. Such rows are not returned by a simple join. To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition. For all rows in A that have no matching rows in B, Oracle7 returns NULL for any select list expressions containing columns of B.

This is the basic syntax of an outer join of two tables:



Outer join queries are subject to the following rules and restrictions:

- The (+) operator can only appear in the WHERE clause, not in the select list, and can only be applied to a column of a table or view.
- If A and B are joined by multiple join conditions, the (+) operator must be used in all of these conditions.
- The (+) operator can only be applied to a column, rather than to an arbitrary expression, although an arbitrary expression can contain a column marked with the (+) operator.
- A condition containing the (+) operator cannot be combined with another condition using the OR logical operator.
- A condition cannot use the IN comparison operator to compare a column marked with the (+) operator to another expression.
- A condition cannot compare a column marked with the (+) operator to a subquery.

If the WHERE clause contains a condition that compares a column from table B to a constant, the (+) operator must be applied to the column so that the rows from table A for which Oracle7 has generated NULLs for this column are returned.

In a query that performs outer joins of more than two pairs of tables, a single table can only be the NULL–generated table for one other table. For this reason, you cannot apply the (+) operator to columns of B in the join condition for A and B and the join condition for B and C.

Example XVII This query uses an outer join to extend the results of Example XII:

```
SELECT ename, job, dept.deptno, dname
    FROM emp, dept
    WHERE emp.deptno (+) = dept.deptno

ENAME      JOB           DEPTNO DNAME
---------- --------- ---------- --------------
CLARK      MANAGER           10 ACCOUNTING
KING       PRESIDENT         10 ACCOUNTING
MILLER     CLERK             10 ACCOUNTING
SMITH      CLERK             20 RESEARCH
ADAMS      CLERK             20 RESEARCH
FORD       ANALYST           20 RESEARCH
SCOTT      ANALYST           20 RESEARCH
JONES      MANAGER           20 RESEARCH
ALLEN      SALESMAN          30 SALES
BLAKE      MANAGER           30 SALES
MARTIN     SALESMAN          30 SALES
JAMES      CLERK             30 SALES
TURNER     SALESMAN          30 SALES
WARD       SALESMAN          30 SALES
                             40 OPERATIONS
```

In this outer join, Oracle7 returns a row containing the OPERATIONS department even though no employees work in this department. Oracle7 returns NULL in the ENAME and JOB columns for this row. The join query in Example X only selects departments that have employees.

The following query uses an outer join to extend the results of Example XV:

```
SELECT ename, job, dept.deptno, dname
   FROM emp, dept
   WHERE emp.deptno (+) = dept.deptno
     AND job (+) = 'CLERK'

ENAME      JOB           DEPTNO DNAME
---------- --------- ---------- --------------
MILLER     CLERK             10 ACCOUNTING
SMITH      CLERK             20 RESEARCH
ADAMS      CLERK             20 RESEARCH
JAMES      CLERK             30 SALES
                             40 OPERATIONS
```

In this outer join, Oracle7 returns a row containing the OPERATIONS department even though no clerks work in this department. The (+) operator on the JOB column ensures that rows for which the JOB column is NULL are also returned. If this (+) were omitted, the row containing the OPERATIONS department would not be returned because its JOB value is not 'CLERK'.

Example XVIII This example shows four outer join queries on the CUSTOMERS, ORDERS, LINEITEMS, and PARTS tables. These tables are shown here:

```
SELECT custno, custname
   FROM customers

CUSTNO     CUSTNAME
---------- --------------------
1          Angelic Co.
2          Believable Co.
3          Cabels R Us

SELECT orderno, custno,
   TO_CHAR(orderdate, 'MON-DD-YYYY') "ORDERDATE"
   FROM orders

ORDERNO    CUSTNO     ORDERDATE
---------- ---------- -----------
9001       1          OCT-13-1993
9002       2          OCT-13-1993
9003       1          OCT-20-1993
9004       1          OCT-27-1993
9005       2          OCT-31-1993
```

```
SELECT orderno, lineno, partno, quantity
    FROM lineitems

ORDERNO        LINENO     PARTNO   QUANTITY
---------- ---------- ---------- ----------
      9001          1        101         15
      9001          2        102         10
      9002          1        101         25
      9002          2        103         50
      9003          1        101         15
      9004          1        102         10
      9004          2        103         20

SELECT partno, partname
    FROM parts

PARTNO PARTNAME
------ --------
   101 X–Ray Screen
   102 Yellow Bag
   103 Zoot Suit
```

Note that the customer Cables R Us have placed no orders and that order number 9005 has no line items.

The following outer join returns all customers and the dates they placed orders. The (+) operator ensures that customers who placed no orders are also returned:

```
SELECT custname, TO_CHAR(orderdate, 'MON–DD–YYYY') "ORDERDATE"
    FROM customers, orders
    WHERE customers.custno = orders.custno (+)

CUSTNAME             ORDERDATE
-------------------- --------------
Angelic Co.          OCT–13–1993
Angelic Co.          OCT–20–1993
Angelic Co.          OCT–27–1993
Believable Co.       OCT–13–1993
Believable Co.       OCT–31–1993
Cables R Us
```

The following outer join builds on the result of the previous one by adding the LINEITEMS table to the FROM clause, columns from this table to the select list, and a join condition joining this table to the ORDERS table to the WHERE clause. This query joins the results of the previous query to the LINEITEMS table and returns all customers, the dates they placed orders, and the part number and quantity of each part they ordered. The first (+) operator serves the same purpose as in the previous query. The second (+) operator ensures that orders with no line items are also returned:

```
SELECT custname,
TO_CHAR(orderdate, 'MON-DD-YYYY') "ORDERDATE",
partno,
quantity
    FROM customers, orders, lineitems
    WHERE customers.custno = orders.custno (+)
      AND orders.orderno = lineitems.orderno (+)

CUSTNAME             ORDERDATE          PARTNO   QUANTITY
-------------------- -------------- ---------- ----------
Angelic Co.          OCT-13-1993           101         15
Angelic Co.          OCT-13-1993           102         10
Angelic Co.          OCT-20-1993           101         15
Angelic Co.          OCT-27-1993           102         10
Angelic Co.          OCT-27-1993           103         20
Believable Co.       OCT-13-1993           101         25
Believable Co.       OCT-13-1993           103         50
Believable Co.       OCT-31-1993
Cables R Us
```

The following outer join builds on the result of the previous one by adding the PARTS table to the FROM clause, the PARTNAME column from this table to the select list, and a join condition joining this table to the LINEITEMS table to the WHERE clause. This query joins the results of the previous query to the PARTS table to return all customers, the dates they placed orders, and the quantity and name of each part they ordered. The first two (+) operators serve the same purposes as in the previous query. The third (+) operator ensures that rows with NULL part numbers are also returned:

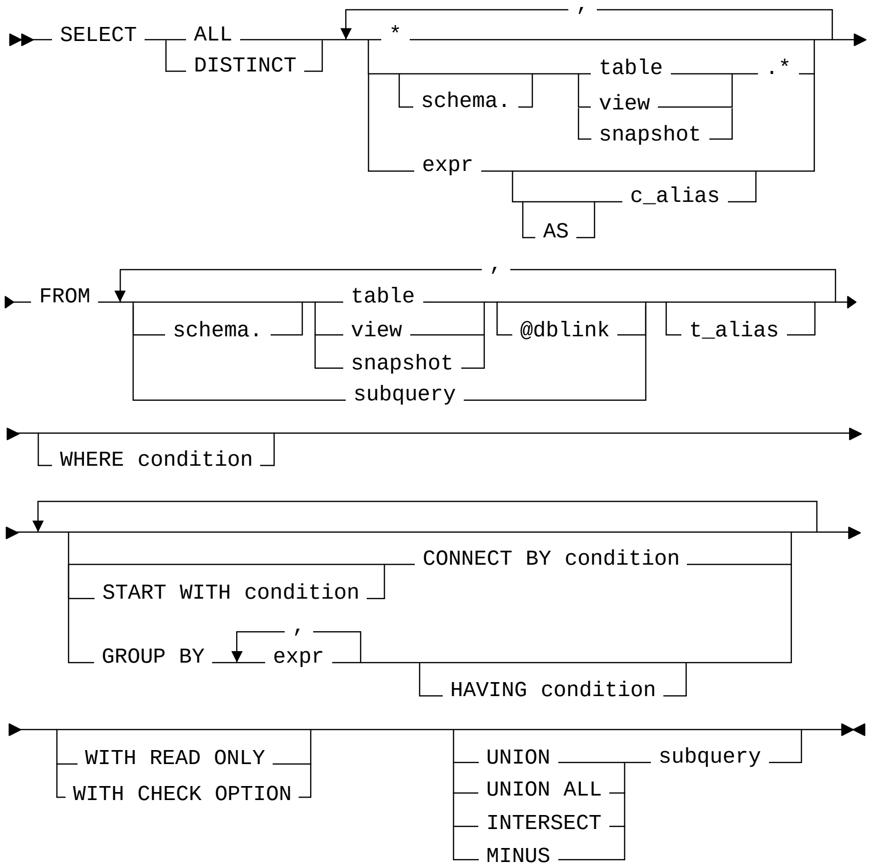
```
SELECT custname, TO_CHAR(orderdate, 'MON-DD-YYYY') "ORDERDATE",
    quantity, partname
    FROM customers, orders, lineitems, parts
    WHERE customers.custno = orders.custno (+)
      AND orders.orderno = lineitems.orderno (+)
      AND lineitems.partno = parts.partno (+)

CUSTNAME             ORDERDATE        QUANTITY PARTNAME
-------------------- -------------- ---------- ------------
Angelic Co.          OCT-13-1993            15 X-Ray Screen
Angelic Co.          OCT-13-1993            10 Yellow Bag
Angelic Co.          OCT-20-1993            15 X-Ray Screen
Angelic Co.          OCT-27-1993            10 Yellow Bag
Angelic Co.          OCT-27-1993            20 Zoot Suit
Believable Co.       OCT-13-1993            25 X-Ray Screen
Believable Co.       OCT-13-1993            50 Zoot Suit
Believable Co.       OCT-31-1993
Cables R Us
```

Subqueries

A *subquery* is a form of the SELECT command that appears inside another SQL statement. A subquery is sometimes called a nested query. The statement containing a subquery is called the *parent statement*. The rows returned by the subquery are used by the parent statement.

This is the syntax for a subquery:



Keywords and Parameters

WITH READ ONLY
: specifies that the subquery cannot be updated.

WITH CHECK OPTION
: specifies that, if the subquery is used in place of a table in an INSERT, UPDATE, or DELETE statement, changes to that table that would produce rows excluded from the subquery are prohibited. In other words, the following statement:

```
INSERT INTO (SELECT ename, deptno FROM Emp
               WHERE deptno < 10)
        VALUES ('Taylor', 20);
```

would be legal, but

```
INSERT INTO (SELECT ename, deptno FROM Emp
               WHERE deptno < 10
               WITH CHECK OPTION)
        VALUES ('Taylor', 20);
```

would be rejected.

Other keywords and parameters are as outlined after the SELECT syntax diagram in the beginning of this entry.

Usage Notes

Subqueries can be used for the following purposes:

- to define the set of rows to be inserted into the target table of an INSERT or CREATE TABLE statement
- to define the set of rows to be included in a view or snapshot in a CREATE VIEW or CREATE SNAPSHOT statement
- to define one or more values to be assigned to existing rows in an UPDATE statement
- to provide values for conditions in WHERE, HAVING, and START WITH clauses of SELECT, UPDATE, and DELETE statements
- to define a table to be operated on by a containing query. You do this by placing the subquery in the FROM clause of the containing query as you would a table name. You may use subqueries in place of tables in this way as well in INSERT, UDPATE, and DELETE statements. Subqueries so used can

employ correlation variables, but only those defined within the subquery itself, not outer references.

A subquery answers multiple part questions. For example, to determine who works in Taylor's department, you can first use a subquery to determine in which department Taylor works. You can then answer the original question with the parent SELECT statement.

A subquery is evaluated once for the entire parent statement, in contrast to a correlated subquery which is evaluated once per row processed by the parent statement.

A subquery can itself contain a subquery. Oracle7 places no limit on the level of query nesting.

Example XIX

To determine who works in Taylor's department, issue the following statement:

```
SELECT ename, deptno
    FROM emp
    WHERE deptno =
        (SELECT deptno
            FROM emp
            WHERE ename = 'TAYLOR')
```

Example XX

To give all employees in the EMP table a ten percent raise if they have not already been issued a bonus (if they do not appear in the BONUS table), issue the following statement:

```
UPDATE emp
    SET sal = sal * 1.1
    WHERE empno NOT IN (SELECT empno FROM bonus)
```

Example XXI

To create a duplicate of the DEPT table named NEWDEPT, issue the following statement:

```
CREATE TABLE newdept (deptno, dname, loc)
    AS SELECT deptno, dname, loc FROM dept
```

Correlated Subqueries

A *correlated subquery* is a subquery that is evaluated once for each row processed by the parent statement. The parent statement can be a SELECT, UPDATE, or DELETE statement. The following examples show the general syntax of a correlated subquery:

```
SELECT select_list
    FROM table1 t_alias1
    WHERE expr operator
        (SELECT column_list
            FROM table2 t_alias2
            WHERE t_alias1.column
                  operator t_alias2.column)
UPDATE table1 t_alias1
    SET column =
        (SELECT expr
            FROM table2 t_alias2
            WHERE t_alias1.column = t_alias2.column)
DELETE FROM table1 t_alias1
    WHERE column operator
        (SELECT expr
            FROM table2 t_alias2
            WHERE t_alias1.column = t_alias2.column)
```

This discussion focuses on correlated subqueries in SELECT statements, although it also applies to correlated subqueries in UPDATE and DELETE statements.

You can use a correlated subquery to answer a multi–part question whose answer depends on the value in each row processed by the parent statement. For example, a correlated subquery can be used to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Oracle7 performs a correlated subquery when the subquery references a column from a table from the parent statement.

Oracle7 resolves unqualified columns in the subquery by looking in the tables of the subquery, then in the tables of the parent statement, then in the tables of the next enclosing parent statement, and so on. Oracle7 resolves all unqualified columns in the subquery to the same table. If the tables in a subquery and parent query contain a column with the same name, a reference to the column of a table from the parent query must be prefixed by the table name or alias. To make your statements easier for you to read, always qualify the columns in a correlated subquery with the table, view, or snapshot name or alias.

In the case of an UPDATE statement, you can use a correlated subquery to update rows in one table based on rows from another table. For example, you could use a correlated subquery to roll up four quarterly sales tables into a yearly sales table.

In the case of a DELETE statement, you can use a correlated query to delete only those rows that also exist in another table.

Example XXII

The following statement returns data about employees whose salaries exceed the averages for their departments. The following statement assigns an alias to EMP, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal
    FROM emp x
    WHERE sal > (SELECT AVG(sal)
                    FROM emp
                    WHERE x.deptno = deptno)
    ORDER BY deptno
```

For each row of the EMP table, the parent query uses the correlated subquery to compute the average salary for members of the same department. The correlated subquery performs these steps for each row of the EMP table:

1. The DEPTNO of the row is determined.
2. The DEPTNO is then used to evaluate the parent query.
3. If that row’s salary is greater than the average salary for that row’s department, then the row is returned.

The subquery is evaluated once for each row of the EMP table.

Selecting from the DUAL Table

DUAL is a table automatically created by Oracle7 along with the data dictionary. DUAL is in the schema of the user SYS, but is accessible by the name DUAL to all users. It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value ’X’. Selecting from the DUAL table is useful for computing a constant expression with the SELECT command. Because DUAL has only one row, the constant is only returned once. Alternatively, you can select a constant, pseudocolumn, or expression from any table.

Example XXIII The following statement returns the current date:

```
SELECT SYSDATE FROM DUAL
```

You could select SYSDATE from the EMP table, but Oracle7 would return 14 rows of the same SYSDATE, one for every row of the EMP table. Selecting from DUAL is more convenient.

Using Sequences

The sequence pseudocolumns NEXTVAL and CURRVAL can also appear in the select list of a SELECT statement. For information on sequences and their use, see the CREATE SEQUENCE command on page 4 – 224 and the section "Pseudocolumns" on page 2 – 38.

Example XXIV The following statement increments the ZSEQ sequence and returns the new value:

```
SELECT zseq.nextval
    FROM dual
```

The following statement selects the current value of ZSEQ:

```
SELECT zseq.currval
    FROM dual
```

Distributed Queries

Oracle's distributed database management system architecture allows you to access data in remote databases using SQL*Net and an Oracle7 Server. You can identify a remote table, view, or snapshot by appending *@dblink* to the end of its name. The *dblink* must be a complete or partial name for a database link to the database containing the remote table, view, or snapshot. For more information on referring to database links, see the section "Referring to Objects in Remote Databases" on page 2 – 11.

Distributed queries are currently subject to this restriction all tables locked by a FOR UPDATE clause and all tables with LONG columns selected by the query must be located on the same database. For example, the following statement will cause an error:

```
SELECT emp_ny.*
    FROM emp_ny@ny, dept
    WHERE emp_ny.deptno = dept.deptno
      AND dept.dname = 'ACCOUNTING'
    FOR UPDATE OF emp_ny.sal
```

Also, you cannot issue the above statement because it selects LONG_COLUMN, a LONG value, from the EMP_REVIEW table on the NY database and locks the EMP table on the local database:

```
SELECT emp.empno, review.long_column, emp.sal
    FROM emp, emp_review@ny review
    WHERE emp.empno = emp_review.empno
    FOR UPDATE OF emp.sal
```

Example XXV This example shows a query which joins the DEPT table on the local database with the EMP table on the HOUSTON database:

```
SELECT ename, dname
    FROM emp@houston, dept
    WHERE emp.deptno = dept.deptno
```

Related Topics

DELETE command on 4 – 286
SELECT (Embedded SQL) command on 4 – 405
UPDATE command on 4 – 460

SELECT (Embedded SQL)

Purpose

To retrieve data from one or more tables, views, or snapshots, assigning the selected values to host variables.

Prerequisites

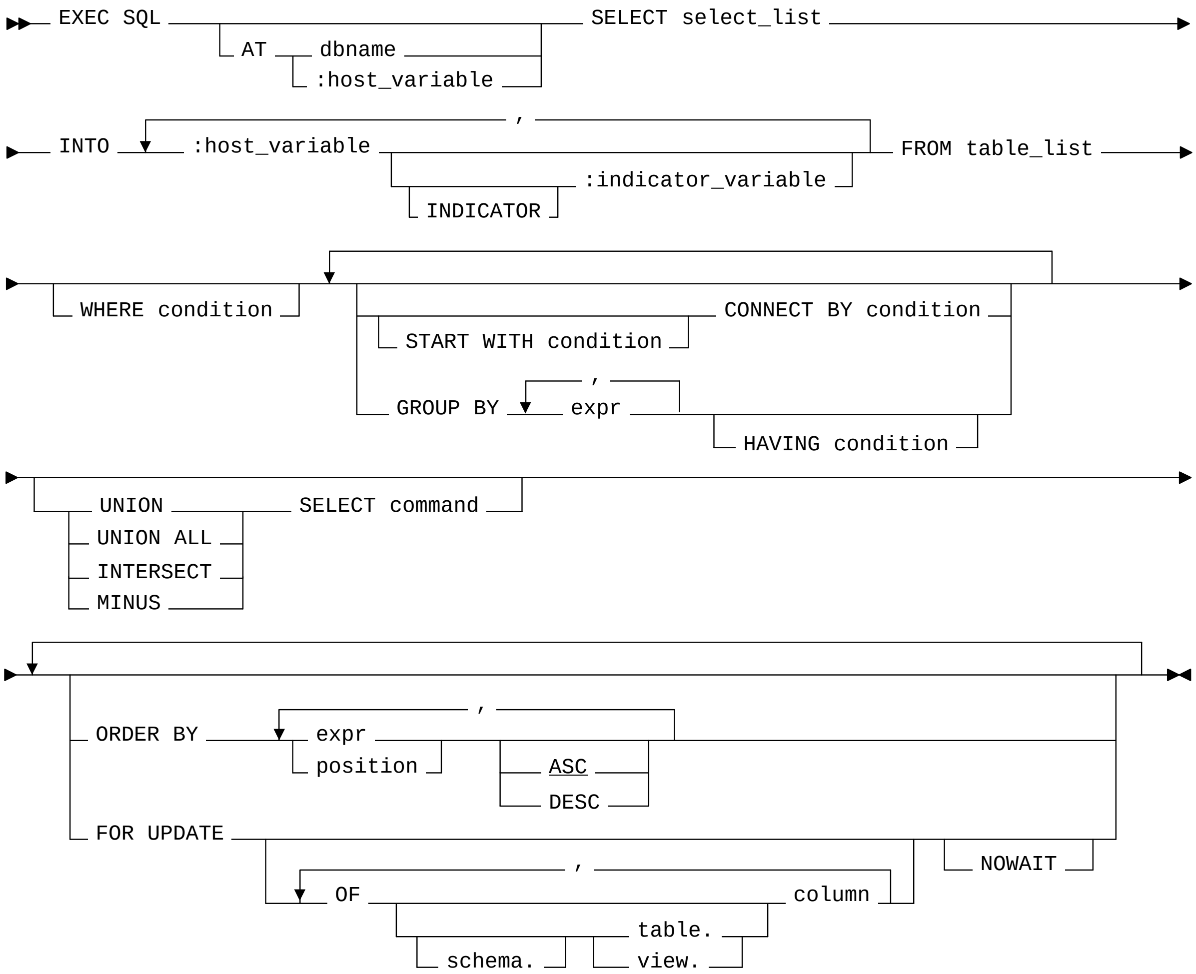
For you to select data from a table or snapshot, the table or snapshot must be in your own schema or you must have SELECT privilege on the table or snapshot.

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 Oracle7 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.

Syntax


EXEC SQL
AT
dbname
:host_variable
SELECT select_list
INTO
,
:host_variable
INDICATOR
:indicator_variable
FROM table_list
WHERE condition
START WITH condition
CONNECT BY condition
GROUP BY
,
expr
HAVING condition
UNION
UNION ALL
INTERSECT
MINUS
SELECT command
ORDER BY
,
expr
position
ASC
DESC
FOR UPDATE
,
OF
schema.
table.
view.
column
NOWAIT

Keywords and Parameters

AT
: identifies the database to which the SELECT statement is issued. The database can be identified by either:

  db_name
  : is a database identifier declared in a previous DECLARE DATABASE statement.

  :*host_variable*
  : is a host variable whose value is a previously declared *db_name.*

  If you omit this clause, the SELECT statement is issued to your default database.

select_list
: identical to the non–embedded SELECT command except that host variables can be used in place of literals.

INTO
: specifies output host variables and optional indicator variables to receive the data returned by the SELECT statement. Note that these variables must be either all scalars or all arrays, but arrays need not have the same size.

WHERE
: restricts the rows returned to those for which the condition is TRUE. See the syntax description of *condition* on page 3 – 78. The *condition* can contain host variables, but cannot contain indicator variables. These host variables can be either scalars or arrays.

All other keywords and parameters are identical to the non–embedded SQL SELECT command.

Usage Notes

If no rows meet the WHERE clause condition, no rows are retrieved and Oracle7 returns an error code through the SQLCODE component of the SQLCA.

You can use comments in a SELECT statement to pass instructions, or *hints*, to the Oracle7 optimizer. The optimizer uses hints to choose an execution plan for the statement. For more information on hints, see *Oracle7 Server Tuning*.

Example I

This example illustrates the use of the embedded SQL SELECT command:

```
EXEC SQL SELECT ename, sal + 100, job
    INTO :ename, :sal, :job
    FROM emp
    WHERE empno = :empno
```

Related Topics

DECLARE DATABASE command on 4 – 282
DECLARE CURSOR command on 4 – 280
EXECUTE command on 4 – 332
FETCH command on 4 – 341
PREPARE command on 4 – 397

SET ROLE

Purpose

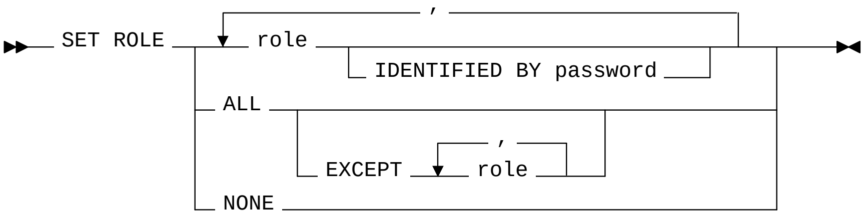
To enable and disable roles for your current session.

Prerequisites

You also must already have been granted the roles that you name in the SET ROLE statement.

If you are using Trusted Oracle7 in DBMS MAC mode, your DBMS label must dominate the label of roles granted to you.

Syntax



Keywords and Parameters

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.

Default Privilege Domain

At logon Oracle7 establishes your default privilege domain by enabling your default roles. Your default privilege domain contains all privileges granted explicitly to you and all privileges in the privilege domains of your default roles. You can then perform any operations authorized by the privileges in your default privilege domain.

Changing Your Privilege Domain

During your session, you can change your privilege domain with the SET ROLE command. The SET ROLE command changes the roles currently enabled for your session. You can change your enabled roles any number of times during a session. The number of roles that can be concurrently enabled is limited by the initialization parameter MAX_ENABLED_ROLES.

You can use the SET ROLE command to enable or disable any of the following roles:

- roles that have been granted directly to you
- roles granted to you through other roles

You cannot use the SET ROLE command to enable roles that you have not been granted either directly or through other roles.

Your current privilege domain is also changed in the following cases:

- if you are granted a privilege
- if one of your privileges is revoked
- if one of your enabled roles is revoked
- if the privilege domain of one of your enabled roles is changed

If none of the above conditions occur and you do not issue the SET ROLE command, your default privilege domain remains in effect for the duration of your session. In the last two cases, the change in your privilege domain does not take effect until you logon to Oracle7 again or issue a SET ROLE statement.

You can determine which roles are in your current privilege domain at any time by examining the SESSION_ROLES data dictionary view.

To change your default roles, use the ALTER USER command.

Example I To enable the role GARDENER identified by the password MARIGOLDS for your current session, issue the following statement:

```
SET ROLE gardener IDENTIFIED BY marigolds
```

Example II To enable all roles granted to you for the current session, issue the following statement:

```
SET ROLE ALL
```

Example III To enable all roles granted to you except BANKER, issue the following statement:

```
SET ROLE ALL EXCEPT banker
```

Example IV To disable all roles granted to you for the current session, issue the following statement:

```
SET ROLE NONE
```

Related Topics

ALTER USER command on 4 – 108
CREATE ROLE command on 4 – 215

SET TRANSACTION

Purpose

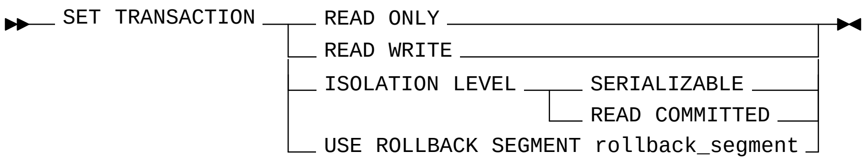
For the current transaction:

- establish as a read–only or read–write transaction
- establish the isolation level
- assign the transaction to a specified rollback segment

Prerequisites

If you use a SET TRANSACTION statement, it must be the first statement in your transaction. However, every transaction need not have a SET TRANSACTION statement.

Syntax



Keywords and Parameters

READ ONLY — establishes the current transaction as a read–only transaction.

READ WRITE — establishes the current transaction as a read–write transaction.

ISOLATION LEVEL

specifies how transactions containing database modifications are handled.

SERIALIZABLE

use the serializable transaction isolation mode as specified in SQL92. That is, if a serializable transaction attempts to execute a DML statement that updates any resource that may have been updated in an uncommitted transaction at the start of the serializable transaction, then the DML statement fails. The COMPATIBLE initialization parameter must be set to 7.3.0 or higher for SERIALIZABLE mode to work.

READ COMMITTED

use the default Oracle transaction behavior. Thus, if the transaction contains DML that require row locks held by another transaction, then the DML statement will wait until the row locks are released.

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.

Usage Notes

The operations performed by a SET TRANSACTION statement affect only your current transaction, not other users or other transactions. Your transaction ends whenever you issue a COMMIT or ROLLBACK statement. Note also that Oracle7 implicitly commits the current transaction before and after executing a Data Definition Language statement.

Establishing Read–only Transactions

The default state for all transactions is statement level read consistency. You can explicitly specify this state by issuing a SET TRANSACTION statement with the READ WRITE option.

You can establish transaction level read consistency by issuing a SET TRANSACTION statement with the READ ONLY option. After a transaction has been established as read–only, all subsequent queries in that transaction only see changes committed before the transaction began. Read–only transactions are very useful for reports that run multiple queries against one or more tables while other users update these same tables.

Only the following statements are permitted in a read–only transaction:

- SELECT (except statements with the FOR UPDATE clause)
- LOCK TABLE
- SET ROLE
- ALTER SESSION
- ALTER SYSTEM

INSERT, UPDATE, and DELETE statements and SELECT statements with the FOR UPDATE clause are not permitted. Any Data Definition Language statement implicitly ends the read–only transaction.

The read consistency that read–only transactions provide is implemented in the same way as statement–level read consistency. Every statement by default uses a consistent view of the data as of the time the statement is issued. Read–only transactions present a consistent view of the data as of the time that the SET TRANSACTION READ ONLY statement is issued. Read–only transactions provide read consistency is for all nodes accessed by distributed queries and local queries.

You cannot toggle between transaction level read consistency and statement level read consistency in the same transaction. A SET TRANSACTION statement can only be issued as the first statement of a transaction.

Example I The following statements could be run at midnight of the last day of every month to count how many ships and containers the company owns. This report would not be affected by any other user who might be adding or removing ships and / or containers.

```
COMMIT
SET TRANSACTION READ ONLY
SELECT COUNT(*) FROM ship
SELECT COUNT(*) FROM container
COMMIT
```

The last COMMIT statement does not actually make permanent any changes to the database. It ends the read–only transaction.

Assigning Transactions to Rollback Segments

If you issue a Data Manipulation Language statement in a transaction, Oracle7 assigns the transaction to a rollback segment. The rollback segment holds the information necessary to undo the changes made by the transaction. You can issue a SET TRANSACTION statement with the USE ROLLBACK SEGMENT clause to choose a specific rollback segment for your transaction. If you do not choose a rollback segment, Oracle7 chooses one randomly and assigns your transaction to it.

SET TRANSACTION allows you to assign transactions of different types to rollback segments of different sizes:

- Assign OLTP transactions, or small transactions containing only a few Data Manipulation Language statements that modify only a few rows, to small rollback segments if there are no long–running queries concurrently reading the same tables. Small rollback segments are more likely to remain in memory.
- Assign transactions that modify tables that are concurrently being read by long–running queries to large rollback segments so that the rollback information needed for the read consistent queries is not overwritten.
- Assign transactions with bulk Data Manipulation Language statements, or statements that insert, update, or delete large amounts of data, to rollback segments large enough to hold the rollback information for the transaction.

Example II The following statement assigns your current transaction to the rollback segment OLTP_5:

```
SET TRANSACTION USE ROLLBACK SEGMENT oltp_5
```

Related Topics

COMMIT command on 4 – 141
ROLLBACK command on 4 – 397
SAVEPOINT command on 4 – 404

STORAGE clause

Purpose

To specify storage characteristics for tables, indexes, clusters, and rollback segments, and the default storage characteristics for tablespaces.

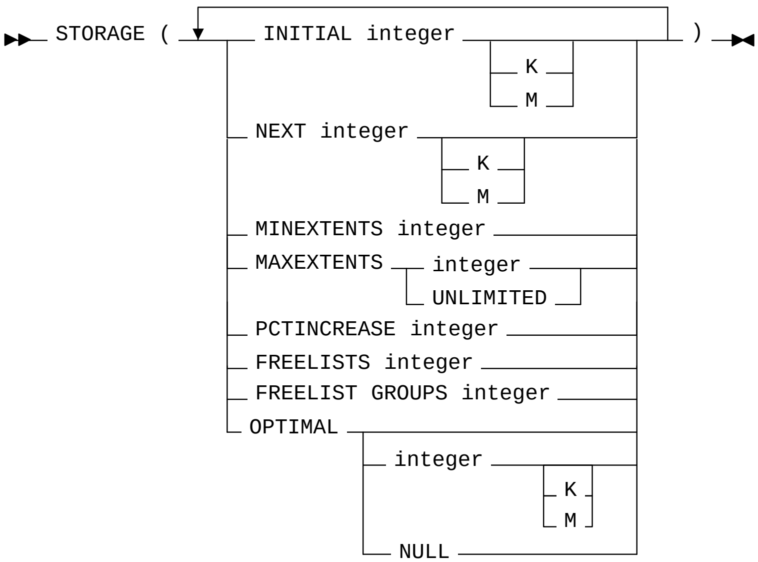
Prerequisites

The STORAGE clause can appear in commands that create or alter any of the following 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 use the appropriate create or alter command.

Syntax



Keywords and Parameters

INITIAL
: specifies the size in bytes of the object's first extent. Oracle7 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. Oracle7 rounds values up to the next multiple of the data block size for values less than 5 data blocks. Oracle7 rounds values up to the next multiple of 5 data blocks.

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. Oracle7 rounds values up to the next multiple of the data block size for values less than 5 data blocks. For values greater than 5 data blocks, Oracle7 rounds up to a value than minimizes fragmentation, as described in the "Data Blocks, Extents, and Segments" chapter of *Oracle7 Server Concepts*.

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.

Oracle7 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 to allocate when the object 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 Oracle7 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 Oracle7 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 Oracle7 can allocate for the object. The minimum value is 1. The default and maximum values vary depending your data block size.

UNLIMITED — specifies that extents should automatically be allocated as needed. You should not use this option for rollback segments.

FREELIST GROUPS — for objects other than tablespaces, specifies the number of groups of free lists for a table, cluster, or index. The default and minimum value for this parameter is 1. Only use this parameter if you are using Oracle7 with the Parallel Server option in parallel mode.

FREELISTS — for objects other than tablespaces, 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, Oracle7 returns an error message 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.

OPTIMAL

specifies an optimal size in bytes for a rollback segment. Not applicable to other kinds of objects. You can also use K or M to specify this size in kilobytes or megabytes. Oracle7 tries to maintain this size for the rollback segment by dynamically deallocating extents when their data is no longer needed for active transactions. Oracle7 deallocates as many extents as possible without reducing the total size of the rollback segment below the OPTIMAL value.

NULL

specifies no optimal size for the rollback segment, meaning that Oracle7 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. Oracle7 rounds values to the next multiple of the data block size.

Usage Notes

The STORAGE parameters affect both how long it takes to access data stored in the database and how efficiently space in the database is used. For a discussion of the effects of these parameters, see the "Tuning I/O" chapter of *Oracle7 Server Tuning*.

When you create a tablespace, you can specify values for the STORAGE parameters. These values serve as default STORAGE parameter values for segments allocated in the tablespace.

When you create a cluster, index, rollback segments, snapshot, snapshot log, or table, you can specify values for the STORAGE parameters for the segments allocated to these objects. If you omit any STORAGE parameter, Oracle7 uses the value of that parameter specified for the tablespace.

When you alter a cluster, index, rollback segment, snapshot, snapshot log, or table, you can change the values of STORAGE parameters. These new values only affect future extent allocations. For this reason, you cannot change the values of the INITIAL and MINEXTENTS parameter. If you change the value of the NEXT parameter, the next allocated extent will have the specified size, regardless of the size of the most–recently allocated extent and the value of the PCTINCREASE parameter. If you change the value of the PCTINCREASE parameter, Oracle7 calculates the size of the next extent using this new value and the size of the most recently allocated extent.

When you alter a tablespace, you can change the values of STORAGE parameters. These new values serve as default values only to subsequently allocated segments (or subsequently created objects).

ROLLBACK SEGMENTS and MAXEXTENTS UNLIMITED

It is not good practice to create or alter a rollback segment to use MAXEXTENTS UNLIMITED. Rogue transactions containing inserts, updates, or deletes, that continue for a long time will continue to create new extents until a disk is full.

A rollback segment created without specifying the storage option has the same storage options as the tablespace that the rollback segment is created in. Thus, if the tablespace is created with MAXEXTENT UNLIMITED, then the rollback segment would also have the same default.

Example I

The following statement creates a table and provides STORAGE parameter values:

```
CREATE TABLE dept
    (deptno  NUMBER(2),
     dname   VARCHAR2(14),
     loc     VARCHAR2(13) )
    STORAGE (INITIAL 100K  NEXT     50K
             MINEXTENTS 1  MAXEXTENTS 50  PCTINCREASE 5 )
```

Oracle7 allocates space for the table based on the STORAGE parameter values for the following reasons:

- Because the MINEXTENTS value is 1, Oracle7 allocates 1 extent for the table upon creation.
- Because the INITIAL value is 100K, the first extent's size is 100 kilobytes.
- If the table data grows to exceed the first extent, Oracle7 allocates a second extent. Because the NEXT value is 50K, the second extent's size is 50 kilobytes.
- If the table data subsequently grows to exceed the first two extents, Oracle7 allocates a third extent. Because the PCTINCREASE value is 5, the calculated size of the third extent is 5% larger than the second extent, or 52.5 kilobytes. If the data

block size is 2 kilobytes, Oracle7 rounds this value to 52 kilobytes.

If the table data continues to grow, Oracle7 allocates more extents, each 5% larger than the previous one.

- Because the MAXEXTENTS value is 50, Oracle7 can allocate as many as 50 extents for the table.

Example II The following statement creates a rollback segment and provides STORAGE parameter values:

```
CREATE ROLLBACK SEGMENT rsone
    STORAGE ( INITIAL  10K  NEXT 10K
              MINEXTENTS 2  MAXEXTENTS 25
              OPTIMAL 50K )
```

Oracle7 allocates space for the rollback segment based on the STORAGE parameter values:

- Because the MINEXTENTS value is 2, Oracle7 allocates 2 extents for the rollback segment upon creation.
- Because the INITIAL value is 10K, the first extent's size is 10 kilobytes.
- Because the NEXT value is 10K, the second extent's size is 10 kilobytes.
- If the rollback data exceeds the first two extents, Oracle7 allocates a third extent. Because the PCTINCREASE value for rollback segments is always 0, the third extent is the same size as the second extent, 10 kilobytes.

  If the rollback data continues to grow, Oracle7 allocates more extents, each the same size as the previous one, 10 kilobytes.
- Because the MAXEXTENTS value is 25, Oracle7 can allocate as many as 25 extents for the rollback segment.
- Because the OPTIMAL value is 50K, Oracle7 deallocates extents if the rollback segment exceeds 50 kilobytes. Note that Oracle7 only deallocates extents that contain data for transactions that are no longer active.

Related Topics

CREATE CLUSTER command on 4 – 164
CREATE INDEX command on 4 – 192
CREATE ROLLBACK SEGMENT command on 4 – 218
CREATE TABLE command on 4 – 245
CREATE TABLESPACE command on 4 – 254

TRUNCATE

Purpose

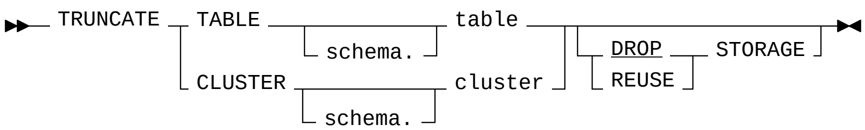
To remove all rows from a table or cluster and reset the STORAGE parameters to the values when the table or cluster was created.

Prerequisites

The table or cluster must be in your schema or you must have DELETE 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 not comparable or higher than your DBMS label, you must have READUP system privilege.

Syntax



Keywords and Parameters

TABLE
specifies the schema and name of the table to be truncated. If you omit *schema*, Oracle7 assumes the table is in your own schema. This table cannot be part of a cluster.

When you truncate a table, Oracle7 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*, Oracle7 assumes the cluster is in your own schema. You can only truncate an indexed cluster, not a hash cluster.

When you truncate a cluster, Oracle7 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. STORAGE values are not reset to the values when the table or cluster was created. 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, Oracle7 uses the DROP STORAGE option by default.

Usage Notes

You can use the TRUNCATE command to quickly remove all rows from a table or cluster. Removing rows with the TRUNCATE command is faster than removing them with the DELETE command for the following reasons:

- The TRUNCATE command is a Data Definition Language command and generates no rollback information.
- Truncating a table does not fire the table's DELETE triggers.
- Truncating the master table of a snapshot does not record any changes in the table's snapshot log.

The TRUNCATE command allows you to optionally deallocate the space freed by the deleted rows. The DROP STORAGE option deallocates all but the space specified by the table's MINEXTENTS parameter.

Deleting rows with the TRUNCATE command is also more convenient than dropping and recreating a table for the following reasons:

- Dropping and recreating invalidates the table's dependent objects, while truncating does not.
- Dropping and recreating requires you to regrant object privileges on the table, while truncating does not.
- Dropping and recreating requires you to recreate the table's indexes, integrity constraints, and triggers and respecify its STORAGE parameters, while truncating does not.

When you truncate a table, NEXT is automatically reset to the last extent deleted.

You cannot individually truncate a table that is part of a cluster. You must either truncate the cluster, delete all rows from the table, or drop and recreate the table.

You cannot truncate the parent table of an enabled referential integrity constraint. You must disable the constraint before truncating the table.

If you truncate the master table of a snapshot, Oracle7 does not record the removed rows in the snapshot log. For this reason, a fast refresh does not remove the rows from the snapshot. Snapshots based on a truncated table must be refreshed completely for Oracle7 to remove their rows.

You cannot roll back a TRUNCATE statement.

Example I

The following statement deletes all rows from the EMP table and returns the freed space to the tablespace containing EMP:

```
TRUNCATE TABLE emp
```

The above statement also deletes all data from all indexes on EMP and returns the freed space to the tablespaces containing them.

Example II

The following statement deletes all rows from all tables in the CUST cluster, but leaves the freed space allocated to the tables:

```
TRUNCATE CLUSTER cust
   REUSE STORAGE
```

The above statement also deletes all data from all indexes in the tables in CUST.

Related Topics

DELETE command on 4 – 286
DROP CLUSTER command on 4 – 301
DROP TABLE command on 4 – 318

TYPE (Embedded SQL)

Purpose

To perform *user–defined type equivalencing*, or to assign an Oracle7 external datatype to a whole class of host variables by equivalencing the external datatype to a user–defined datatype.

Prerequisites

The user–defined datatype must be previously declared in an embedded SQL program.

Syntax

```
>>— EXEC SQL TYPE type IS datatype ————————————————————————><
```

Keywords and Parameters

type is the user–defined datatype to be equivalenced with an Oracle7 external datatype.

datatype is an Oracle7 external datatype recognized by the Oracle Precompilers (not an Oracle7 internal datatype). The datatype may include a length, precision, or scale. This external datatype is equivalenced to the user–defined *type* and assigned to all host variables assigned the *type*. For a list of external datatypes, see *Programmer's Guide to the Oracle Precompilers.*

Usage Notes

User defined type equivalencing is one kind of datatype equivalencing. You can only perform user–defined type equivalencing with the embedded SQL TYPE command in a Pro*C or Pro*Pascal Precompiler program. You may want to use datatype equivalencing for one of the following purposes:

- to automatically null–terminate a character host variable
- to store program data as binary data in the database
- to override default datatype conversion

For more information on using the TYPE command to perform user–defined type equivalencing, see *Programmer's Guide to the Oracle Precompilers.*

All Oracle Precompilers also support the embedded SQL VAR command for host variable equivalencing.

Example I This example shows an embedded SQL TYPE statement in a Pro*C Precompiler program:

```
struct screen {short len;
          char  buff[4002];
              };
typedef struct screen graphics;

EXEC SQL BEGIN DECLARE SECTION;
   EXEC SQL TYPE graphics IS VARRAW (4002);
   graphics crt;  –– host variable of type graphics
   ...
EXEC SQL END DECLARE SECTION;
```

Example II This example shows an embedded SQL TYPE statement in a Pro*Pascal Precompiler program:

```
Type
   OraDate = Record
          Cent, Year, Month, Day, Hour, Min, Sec: Byte
    End;

Var
   EXEC SQL BEGIN DECLARE SECTION;
     EXEC SQL TYPE OraDate IS DATE;
     Birthday: OraDate;  –– host variable of type OraDate
     ...
   EXEC SQL END DECLARE SECTION;
```

Related Topics

VAR command (Embedded SQL) on 4 – 469

UPDATE

Purpose

To change existing values in a table or in a view's base table.

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.

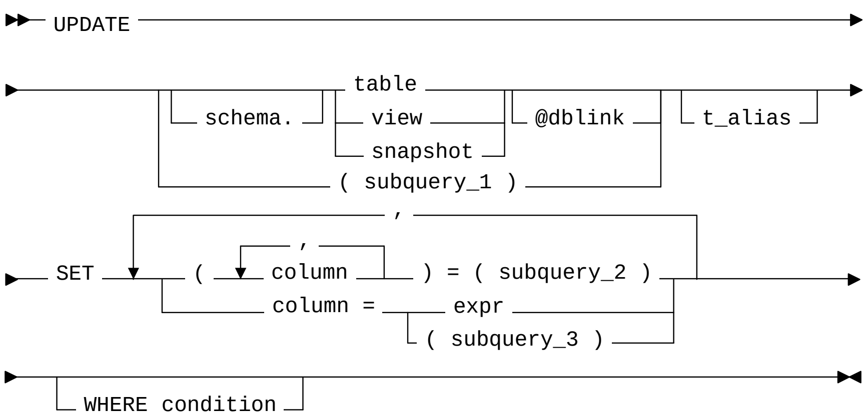
For you to update values in the base table of a view, the owner of the schema containing the view must have UPDATE privilege on the base table. Also, if the view is in a schema other than your own, you must have UPDATE privilege on the view.

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 Oracle7 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

schema — is the schema containing the table or view. If you omit *schema,* Oracle7 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,* Oracle7 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. For information on referring to database links, see the section, "Referring to Objects in Remote Databases," on page 2 – 11. You can only use a database link to update a remote table or view if you are using Oracle7 with the distributed option.

If you omit *dblink,* Oracle7 assumes the table or view is on the local database.

alias — provides a different name for the table, view, or subquery to be referenced elsewhere in the statement.

subquery_1 — is a subquery that Oracle treats in the same manner as a view. For the syntax of *subquery,* see page 4 – 431.

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. See the syntax description of *expr* on page 3 – 73.

subquery_2 — is a subquery that returns new values that are assigned to the corresponding columns. For the syntax of *subquery,* see page 4 – 436.

subquery_3 — is a subquery that return a new value that is assigned to the corresponding column. For the syntax of *subquery,* see page 4 – 436.

WHERE — restricts the rows updated to those for which the specified condition is TRUE. If you omit this clause, Oracle7 updates all rows in the table or view. See the syntax description of *condition* on page 3 – 78.

Usage Notes

The SET clause determines which columns are updated and what new values are stored in them.

The WHERE clause determines the rows in which values are updated. If the WHERE clause is not specified, all rows are updated. For each row that satisfies the WHERE clause, the columns to the left of the equals (=) operator in the SET clause are set to the values of the corresponding expressions on the right. The expressions are evaluated as the row is updated.

You can use comments in an UPDATE statement to pass instructions, or *hints*, to the Oracle7 optimizer. The optimizer uses hints to choose an execution plan for the statement. For more information, see *Oracle7 Server Tuning*.

Issuing an UPDATE statement against a table fires any UPDATE triggers associated with the table.

Updating Views

If a view was created with the WITH CHECK OPTION, you can only update the view if the resulting data satisfies the view's defining query.

You cannot update a view if the view's defining query contains one of the following constructs:

- join
- set operator
- GROUP BY clause
- group function
- DISTINCT operator

Subqueries

If the SET clause contains a subquery, it must return exactly one row for each row updated. Each value in the subquery result is assigned respectively to the columns in the parenthesized list. If the subquery returns no rows, then the column is assigned a null. Subqueries may select from the table being updated.

The SET clause may mix assignments of expressions and subqueries.

Correlated Update

If a subquery refers to columns from the updated table, Oracle7 evaluates the subquery once for each row, rather than once for the entire update. Such an update is called a correlated update. The reference to columns from the updated table is usually accomplished by means of a table alias.

Potentially, each row evaluated by an UPDATE statement could be updated with a different value as determined by the correlated subquery. Normal UPDATE statements update each row with the same value.

Example I

The following statement gives null commissions to all employees with the job TRAINEE:

```
UPDATE emp
    SET comm = NULL
    WHERE job = 'TRAINEE'
```

Example II

The following statement promotes JONES to manager of Department 20 with a $1,000 raise (assuming there is only one JONES):

```
UPDATE emp
    SET job = 'MANAGER', sal = sal + 1000, deptno = 20
    WHERE ename = 'JONES'
```

Example III

The following statement increases the balance of bank account number 5001 in the ACCOUNTS table on a remote database accessible through the database link BOSTON:

```
UPDATE accounts@boston
    SET balance = balance + 500
    WHERE acc_no = 5001
```

Example IV

This example shows the following syntactic constructs of the UPDATE command:

- both forms of the SET clause together in a single statement
- a correlated subquery
- a WHERE clause to limit the updated rows

```
UPDATE emp a
    SET deptno =
        (SELECT deptno
            FROM dept
            WHERE loc = 'BOSTON'),
        (sal, comm) =
        (SELECT 1.1*AVG(sal), 1.5*AVG(comm)
            FROM emp b
            WHERE a.deptno = b.deptno)
    WHERE deptno IN
        (SELECT deptno
            FROM dept
            WHERE loc = 'DALLAS'
               OR loc = 'DETROIT')
```

The above UPDATE statement performs the following operations:

- updates only those employees who work in Dallas or Detroit
- sets DEPTNO for these employees to the DEPTNO of Boston
- sets each employee's salary to 1.1 times the average salary of their department
- sets each employee's commission to 1.5 times the average commission of their department

Related Topics

DELETE command on 4 – 286
INSERT command on 4 – 361

UPDATE (Embedded SQL)

Purpose

To change existing values in a table or in a view's base table.

Prerequisites

For you to update values in a table or snapshot, the table must be in your own schema or you must have UPDATE privilege on the table.

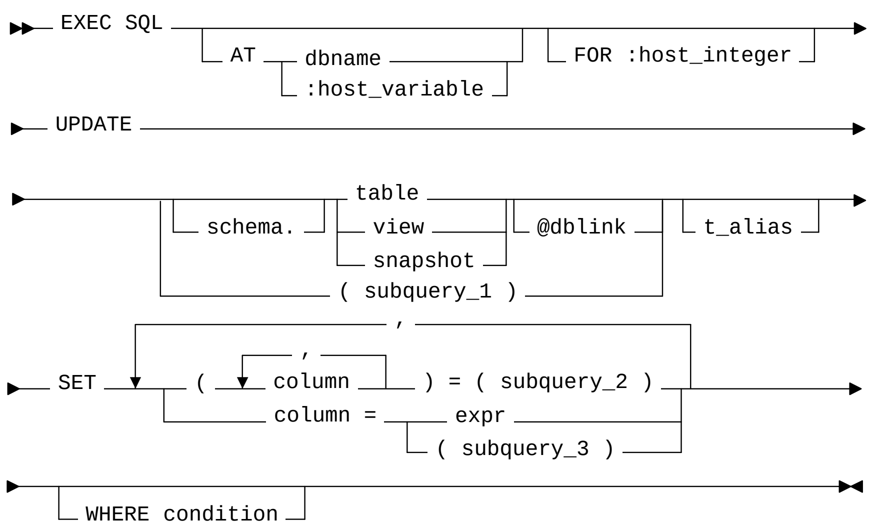
For you to update values in the base table of a view, the owner of the schema containing the view must have UPDATE privilege on the base table. Also, if the view is in a schema other than your own, you must have UPDATE privilege on the view.

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 Oracle7 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 not comparable to your DBMS label, you must have READUP, WRITEUP, and WRITEDOWN system privileges.

Syntax



Keywords and Parameters

AT
: identifies the database to which the UPDATE statement is issued. The database can be identified by either:

  db_name
  : is a database identifier declared in a previous DECLARE DATABASE statement.

  :*host_variable*
  : is a host variable whose value is a previously declared *db_name*.

  If you omit this clause, the UPDATE statement is issued to your default database.

FOR :*host_integer*
: limits the number of times the UPDATE statement is executed if the SET and WHERE clauses contain array host variables. If you omit this clause, Oracle7 executes the statement once for each component of the smallest array.

schema
: is the schema containing the table or view. If you omit *schema*, Oracle7 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*, Oracle7 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. For information on referring to database links, see the section "Referring to Objects in Remote Databases," on page 2 – 11. You can only use a database link to update a remote table or view if you are using Oracle7 with the distributed option.

subquery_1
: is a subquery that Oracle treats in the same manner as a view. For the syntax of *subquery*, see page 4 – 431.

  If you omit *dblink*, Oracle7 assumes the table or view is on the local database.

t_alias
: is a name used to reference the table, view, or subquery elsewhere in the statement.

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. See the syntax description of *expr* on page 3 – 73.

subquery_2 is a subquery that returns new values that are assigned to the corresponding columns. For the syntax of *subquery*, see page 4 – 436.

subquery_3 is a subquery that return a new value that is assigned to the corresponding column. For the syntax of *subquery*, see page 4 – 436.

WHERE specifies which rows of the table or view are updated:

condition updates only rows for which this condition is true. This condition can contain host variables and optional indicator variables. See the syntax description of *condition* on page 3 – 78.

CURRENT OF updates only the row most recently fetched by the *cursor*. The *cursor* cannot be associated with a SELECT statement that performs a join unless its FOR UPDATE clause explicitly locks only one table.

If you omit this clause entirely, Oracle7 updates all rows of the table or view.

Usage Notes

Host variables in the SET and WHERE clauses must be either all scalars or all arrays. If they are scalars, Oracle7 executes the UPDATE statement only once. If they are arrays, Oracle7 executes the statement once for each set of array components. Each execution may update zero, one, or multiple rows.

Array host variables can have different sizes. In this case, the number of times Oracle7 executes the statement is determined by the smaller of the following values:

- the size of the smallest array
- the value of the :*host_integer* in the optional FOR clause

The cumulative number of rows updated is returned through the third element of the SQLERRD component of the SQLCA. When arrays are used as input host variables, this count reflects the total number of updates for all components of the array processed in the UPDATE statement. If no rows satisfy the condition, no rows are updated and Oracle7 returns an error message through the SQLCODE element of the SQLCA. If you omit the WHERE clause, all rows are updated and Oracle7 raises a warning flag in the fifth component of the SQLWARN element of the SQLCA.

You can use comments in an UPDATE statement to pass instructions, or *hints*, to the Oracle7 optimizer. The optimizer uses hints to choose an execution plan for the statement. For more information on hints, see *Oracle7 Server Tuning*.

For more information on this command, see *Programmer's Guide to the Oracle Precompilers*.

Examples

The following examples illustrate the use of the embedded SQL UPDATE command:

```
EXEC SQL UPDATE emp
    SET sal = :sal, comm = :comm INDICATOR :comm_ind
    WHERE ename = :ename;

EXEC SQL UPDATE emp
    SET (sal, comm) =
        (SELECT AVG(sal)*1.1, AVG(comm)*1.1
            FROM emp)
    WHERE ename = 'JONES';
```

Related Topics

DECLARE DATABASE command on 4 – 282
UPDATE command on 4 – 460

VAR (Embedded SQL)

Purpose

To perform *host variable equivalencing*, or to assign a specific Oracle7 external datatype to an individual host variable, overriding the default datatype assignment.

Prerequisites

The host variable must be previously declared in the Declare Section of the embedded SQL program.

Syntax

```
▶▶— EXEC SQL VAR host_variable IS datatype ——————————————▶◀
```

Keywords and Parameters

host_variable is the host variable to be assigned an Oracle7 external datatype.

datatype is an Oracle7 external datatype recognized by the Oracle Precompilers (not an Oracle7 internal datatype). The datatype may include a length, precision, or scale. This external datatype is assigned to the *host_variable*. For a list of external datatypes, see *Programmer's Guide to the Oracle Precompilers.*

Usage Notes

Host variable equivalencing is one kind of datatype equivalencing. You may want to use datatype equivalencing for one of the following purposes:

- to automatically null–terminate a character host variable
- to store program data as binary data in the database
- to override default datatype conversion

For more information on using the VAR command to perform host variable equivalencing, see *Programmer's Guide to the Oracle Precompilers.* The Pro*C and Pro*Pascal Precompilers also support the embedded SQL TYPE command for user–defined type equivalencing.

Example This example equivalences the host variable DEPT_NAME to the datatype STRING and the host variable BUFFER to the datatype RAW(2000):

```
EXEC SQL BEGIN DECLARE SECTION;

    ...

    dept_name CHARACTER(15);  –– default datatype is CHAR

    EXEC SQL VAR dept_name IS STRING; –– reset to STRING

    ...

    buffer CHARACTER(200); –– default datatype is CHAR

    EXEC SQL VAR buffer IS RAW(200); –– refer to RAW

EXEC SQL END DECLARE SECTION;
```

Related Topics

TYPE command (Embedded SQL) on 4 – 458

WHENEVER (Embedded SQL)

Purpose

To specify the action to be taken when if error an warning results from executing an embedded SQL program.

Prerequisites

None.

Syntax

```
>>__ EXEC SQL WHENEVER ____ NOT FOUND ______ ____ CONTINUE ______________ ><
                         |__ SQLERROR ____|  |__ GOTO label_name __|
                         |__ SQLWARNING __|  |__ STOP _____________|
                                             |__ DO routine_call __|
```

Keywords and Parameters

NOT FOUND
: identifies any exception condition that results in a return code of +100 in SQLCODE, (or +1403 in Version 5 compatibility mode).

SQLERROR
: identifies a condition that results in a negative return code.

SQLWARNING
: identifies a non–fatal warning condition.

CONTINUE
: indicates that the program should progress to the next statement.

GOTO
: indicates that the program should branch to the statement named by *label_name*.

STOP
: stops program execution.

DO
: indicates that the program should call a host language routine. The syntax of *routine_call* depends on your host language. See your language–specific Supplement to the Oracle Precompilers Guide.

Usage Notes

The WHENEVER command allows your program to transfer control to an error handling routine in the event an embedded SQL statement results in an error or warning.

The scope of a WHENEVER statement is positional, rather than logical. A single WHENEVER statement applies to all embedded SQL statements that physically follow it in the Precompiler source file, not in the flow of the program logic. A WHENEVER statement remains in effect until it is superseded by another WHENEVER statement checking for the same condition.

For more information on this command, see *Programmer's Guide to the Oracle Precompilers.*

Do not confuse the WHENEVER embedded SQL command with the WHENEVER SQL*Plus command.

Example The example illustrates the use of the WHENEVER command in a Pro*C embedded SQL program:

```
EXEC SQL WHENEVER NOT FOUND;
...
EXEC SQL WHENEVER SQLERROR GOTO sqlerror:
...
sql_error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK RELEASE;
```

Related Topics

None

APPENDIX

A

Differences From Previous Versions

This appendix lists differences between the current and previous releases of Oracle.

Differences Between Oracle7 Release 7.2 and Release 7.3

New SQL Functions

The following transcendental functions are new:

- acos
- asin
- atan
- atan2

ALTER CLUSTER DEALLOCATE UNUSED

You can deallocate unused space from a cluster and save specified kilobytes for future use. For example,

```
ALTER CLUSTER detroit.sales
    DEALLOCATE USUSED KEEP 100K;
```

ALTER DATABASE CREATE STANDBY CONTROLFILE AS <filename>

You can specify the controlfile for a standby database. For example,

```
ALTER DATABASE stocks CREATE STANDBY CONTROLFILE AS controlfile;
```

ALTER DATABASE MOUNT STANDBY DATABASE [EXCLUSIVE, PARALLEL]

You can mount a database or its corresponding standby database as either EXCLUSIVE or as PARALLEL. For example,

```
ALTER DATABASE stocks MOUNT STANDBY DATABASE EXCLUSIVE;
ALTER DATABASE stocks MOUNT STANDBY DATABASE PARALLEL;
```

ALTER DATABASE RECOVER STANDBY DATABASE

You can recover the standby database. For example,

```
ALTER DATABASE stocks RECOVER STANDBY DATABASE;
```

ALTER DATABASE ACTIVATE STANDBY DATABASE

You can activate a standby database. For example,

```
ALTER DATABASE stocks ACTIVATE STANDBY DATABASE;
```

ALTER INDEX ALLOCATE EXTENT

You can allocate an extent to an index after creation.

ALTER INDEX DEALLOCATE UNUSED

You can deallocate unused space from an index and save specified kilobytes for future use. For example,

```
ALTER INDEX april.sales
    DEALLOCATE USUSED KEEP 100K;
```

ALTER INDEX REBUILD

You can use an existing index as the data source of a fast re–create index. This changes the index's storage characteristics. ALTER INDEX has the new parameter, REBUILD. For example,

```
ALTER INDEX salesmen REBUILD UNRECOVERABLE TABLESPACE detroit;
```

ALTER SESSION HASH_JOIN_ ENABLED

You can use hash–join to improve the performance of join operations. There are three new parameters:

- HASH_JOIN_ENABLED turns the feature on or off.
- HASH_AREA_SIZE specifies the maximum amount of memory in bytes to be used for the hash join. If not specified, hash join uses twice the SORT_AREA_SIZE value.
- HASH_MULTIBLOCK_IO_COUNT determines how many blocks hash join should read and write at once. If not specified, hash join uses the value for DB_FILE_MULTIBLOCK_READ_COUNT.
- REMOTE_DEPENDENCIES_MODE specifies how the session handles dependencies of remote stored procedures, by TIMESTAMP or by SIGNATURE..

For example,

```
ALTER SESSION HASH_JOIN_ENABLED = TRUE;
ALTER SESSION HASH_AREA_SIZE = 1000K;
ALTER SESSION HASH_MULTIBLOCK_IO_COUNT = 50;
```

ALTER SESSION SET

You can change dynamic initialization parameters while an instance is running. ALTER SESSION changes the parameter for the duration of the session, or until you re–execute ALTER SESSION. For example,

```
ALTER SESSION SET COMMIT_POINT_STRENGTH 100;
```

ALTER SYSTEM SET REMOTE_ DEPENDENCIES_ MODE

You can alter the system so that timestamp mismatches are now ignored if the user requests that invalidation be based on signatures, rather than by a timestamp with the REMOTE_DEPENDENCIES_MODE parameter.

```
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE = SIGNATURE;
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE = TIMESTAMP;
```

ALTER SYSTEM SET

You can change global value of a dynamic initialization parameter. New sessions use the changed value. For example,

```
ALTER SYSTEM SET COMMIT_POINT_STRENGTH=100;
```

ALTER TABLE DEALLOCATE UNUSED

You can release unused space from a segment and return it to the database system. For example,

```
ALTER TABLE emp DEALLOATE UNUSED KEEP 100K;
```

ALTER TABLESPACE COALESCE

You can improve performance by coalescing available free space (extents) in the tablespace into larger, contiguous extents on a per file basis. For example,

```
ALTER TABLESPACE inventory COALESCE;
```

ALTER TABLESPACE [PERMANENT, TEMPORARY]

You can alter a tablespace to be a permanent or temporary to use with multiple sort operations.

```
ALTER TABLESPACE inventory PERMANENT;
ALTER TABLESPACE inventory TEMPORARY;
```

For more information, see CREATE TABLESPACE [PERMANENT, TEMPORARY] on 4 – 254.

ALTER TRIGGER trigger_name COMPILE

You can now create a compiled trigger that is stored in pcode form, thereby eliminating the need for recompilation during loads and removing RPI calls at execution time. COMPILE and DEBUG are new parameters of the ALTER TRIGGER command. Because triggers now have remote dependencies, they can become invalid if a depended–on object changes. The COMPILE option allows a user to manually recompile an invalid trigger object.

```
ALTER TRIGGER reorder COMPILE;
```

ALTER TRIGGER trigger_name DEBUG

The DEBUG option allows PL / SQL information to be generated during trigger recompilation.

```
ALTER TRIGGER reorder DEBUG;
```

ANALYZE TABLE

You can create histograms on columns that have highly–skewed database distribution and are frequently used in WHERE clauses of queries. You create a histogram with the ANALYZE TABLE command. For example

```
ANALYZE TABLE emp COMPUTE STATISTICS FOR COLUMNS salary SIZE 50;
```

The SIZE keyword states the maximum number of buckets for the histogram.

CREATE TABLESPACE [PERMANENT, TEMPORARY]

You can create a permanent or temporary tablespace to use with multiple sort operations.

A tablespace can be defined as temporary during creation, or it can be made temporary later. The CREATE TABLESPACE command is expanded to include the TEMPORARY and PERMANENT options:

Specifying TEMPORARY defines the tablespace as a *temporary tablespace.* All sorts in a temporary tablespace share a single sort segment and allocate space using the sort segment table. However, no permanent objects can be stored in the temporary tablespace.

Specifying PERMANENT allows the permanent objects to be stored in the tablespace. However, if this tablespace is used for sorting, no caching is done, so sort performance may suffer. For example,

```
CREATE TABLESPACE inventory PERMANENT;
CREATE TABLESPACE inventory TEMPORARY;
```

CREATE TRIGGER trigger_name COMPILE

You can now create a compiled trigger that is stored in pcode form, thereby eliminating the need for recompilation during loads and removing RPI calls at execution time.

```
ALTER TRIGGER reorder COMPILE;
```

SET TRANSACTION READ ONLY

In previous releases, you could use the SET TRANSACTION READ ONLY command to design application transactions that had a consistent view of their data during query–only application transactions. The new isolation level provided by serializable transaction isolation preserves the transaction–consistent view of data that is provided by SET TRANSACTION READ ONLY. Serializable transaction isolation now allows transactions to execute DML statements and to see their own changes while shielding them from visibility of other transactions' changes–either in–flight or committed.

```
SET TRANSACTION ISOLATION_LEVEL SERIALIZABLE;
```

or

```
SET TRANSACTION ISOLATION_LEVEL READ COMMITTED;
```

The SQL command syntax for the ALTER SESSION command is extended as follows:

```
ALTER SESSION SET ISOLATION_LEVEL=SERIALIZABLE
```

or

```
ALTER SESSION SET ISOLATION_LEVEL=READ COMMITTED
```

STORAGE Clause

In Release 7.2 and earlier releases of Oracle7, the number of extents that could be allocated to a single segment was limited by the database block size. The entire extent map had to fit within half of the segment header block. For a 2 Kb block, the maximum number of extents per segment was 121.

The following are changes in space management:

- MAXEXTENTS is no longer limited by the number of extents that fit into a single database block.
- A new keyword, UNLIMITED, is now supported as a valid value for MAXEXTENTS. For example,

```
CREATE TABLESPACE emp MAXEXTENTS UNLIMITED;
```

Differences Between Oracle7 Release 7.1 and Release 7.2

ALTER DATABASE BACKUP CONTROLFILE TO TRACE

It is now possible to write SQL commands to the database's trace file that can be used to re–create the database. For example:

```
ALTER DATABASE BACKUP CONTROLFILE
    TO TRACE
      NORESETLOGS ;
```

ALTER DATABASE CLEAR LOGFILE

It is now possible to reinitialize redo log files during recovery. For example:

```
ALTER DATABASE CLEAR UNARCHIVED
    LOGFILE 'somefile'
    UNRECOVERABLE DATAFILE;
```

ALTER DATABASE DATAFILE datafile END BACKUP

It is now possible to avoid unnecessary media recovery (when the database was closed without finishing an online backup) using the following command:

```
ALTER DATABASE DATAFILE 'file' END BACKUP;
```

ALTER DATABASE DATAFILE datafile RESIZE

It is now possible to dynamically change the size of a datafile. For example:

```
ALTER DATABASE DATAFILE 'file' RESIZE 10M ;
```

ALTER ROLLBACK SEGMENT SHRINK

It is now possible to shrink a rollback segment to an optimum size using the following command:

```
ALTER ROLLBACK SEGMENT name SHRINK TO size ;
```

ALTER SESSION SET INSTANCE

In a parallel server environment while connected to one instance it is now possible to mimic that the session is connected to another instance. For example:

```
ALTER SESSION SET INSTANCE = 3;
```

ALTER SESSION SET NLS_CALENDAR

It is now possible to redefine the language calendar for a session. For example:

```
ALTER SESSION SET NLS_CALENDAR = gregorian;
```

ALTER TABLE ... DISABLE TABLE LOCK

It is now possible to allow or disallow users to use a table lock using the following commands:

```
ALTER TABLE table_name DISABLE TABLE LOCK;
ALTER TABLE table_name ENABLE TABLE LOCK;
```

ALTER TABLESPACE ... ADD DATAFILE ... AUTOEXTEND

It is now possible for datafiles to be automatically extended when more space is required. For example:

```
ALTER TABLESPACE temp ADD DATAFILE 'file' AUTOEXTEND ON;
```

This feature is of most use in a parallel server environment where a table lock can affect system performance.

CREATE CLUSTER ... HASH IS

It is now possible to use your own PL/SQL functions to calculate the hash key. For example:

```
CREATE CLUSTER cloudy (deptno number(2))
    HASHKEY 20 HASH IS my_hash(deptno);
```

CREATE DATABASE DATAFILE datafile AUTOEXTEND

It is now possible to create a database with datafiles that will be automatically extended when more space is required. For example:

```
CREATE DATABASE
    DATAFILE 'file' 10M AUTOEXTEND ON;
```

CREATE INDEX ... UNRECOVERABLE

It is now possible to create an index quickly in ARCHIVELOG mode by avoiding the overhead required to save recovery information. For example:

```
CREATE INDEX tmp_idx
    ON emp(ename)
    UNRECOVERABLE;
```

CREATE TABLE ... UNRECOVERABLE

It is now possible to create a table quickly in ARCHIVELOG mode by avoiding the overhead required to save recovery information. For example:

```
CREATE TABLE quick_emp
    UNRECOVERABLE
    AS SELECT * FROM emp WHERE deptno = 10;
```

CREATE TABLESPACE DATAFILE datafile AUTOEXTEND

It is now possible to create a tablespace with datafiles that will be automatically extended when more space is required. For example:

```
CREATE TABLESPACE DATAFILE 'file' SIZE 10M AUTOEXTEND ON;
```

expr

It is now possible to use a user defined PL/SQL function in the same manner as a SQL expression. For example:

```
SELECT my_fun(ename) FROM emp;
```

INSERT INTO subquery

It is now possible to use a subquery in the INTO clause of an insert statement similar to how views are used. For example:

```
INSERT INTO (SELECT * FROM dept)
    VALUES (50, 'DEVELOPMENT', 'BELMONT');
```

SELECT FROM subquery

It is now possible to use a subquery in the FROM clause of a select statement similar to how views are used. For example:

```
SELECT *
    FROM    (SELECT * FROM dept) a,
            emp b
    WHERE a.deptno = b.deptno
```

TO_CHAR

A number format model using '9's now returns a zero for the value zero. For example:

```
SELECT TO_CHAR(0,'999') num FROM DUAL;

NUM
----
   0
```

UPDATE subquery

It is now possible to use a subquery in an update statement similar to how views are used. For example:

```
UPDATE (SELECT * FROM dept)
    SET deptno = 50
    WHERE deptno = 60
```

Differences Between Oracle7, Release 7.0 and Release 7.1

ALTER CLUSTER

This command has a PARALLEL clause and a CACHE clause to support the parallel query option.

ALTER DATABASE

This command has a RESET COMPATIBILITY option for compatibility control.

You must have ALTER DATABASE system privilege and your instance must have the database open for you to issue this command.

The RECOVER option of this command has changed to include a PARALLEL clause for use with the parallel recovery feature.

ALTER SESSION

This command has a new SET FLAGGER option to support flagging of SQL extensions that go beyond the SQL92 standard for SQL. The SET FLAGGER option has four additional options: entry, intermediate, full, and off.

This command also has a new option for closing cached cursors used by PL/SQL. Using the ALTER SESSION command with this option overrides the initialization parameter CLOSE_CACHED_OPEN_CURSORS for your current session.

This command also has a new option for specifying the size of the session cursor cache. The syntax is:

```
ALTER SESSION SET SESSION_CACHED_CURSORS = integer
```

The integer specified can be any positive integer, but the maximum value is operating–system dependent.

ALTER TABLE

This command has a PARALLEL clause and a CACHE clause to support the parallel query option.

ALTER TABLESPACE

This command has READ ONLY and READ WRITE options to support read–only tablespaces.

This command has BEGIN BACKUP and END BACKUP options to support the parallel server option.

CREATE CLUSTER

This command has a PARALLEL clause and a CACHE clause to support the parallel query option.

CREATE INDEX

This command has a PARALLEL clause to support the parallel query option.

CREATE TABLE

This command has a PARALLEL clause and a CACHE clause to support the parallel query option.

SELECT

There is new syntax and functionality in the following parts of the SELECT command:

- SELECT list
- ORDER BY clause

SELECT List

Column aliases in the SELECT list can optionally be separated from their expressions by the new AS keyword, as in this example:

```
SELECT empno, ename AS name
   FROM emp
```

ORDER BY Clause

The ORDER BY clause can now reference column expression aliases defined in the SELECT list. These column expression aliases effectively rename the SELECT list items for the duration of the expression.

Differences Between Oracle Version 6 and Oracle7, Release 7.0

This section indicates differences between Oracle Version 6 and Oracle7, Release 7.0, and contains the following sections:

- terminology introduced in release 7.0
- reserved words
- Oracle datatypes
- commands
- SQL functions
- format elements
- operators
- comments
- namespaces
- system privileges
- optional components of Oracle7
- compatibility modes

Terminology Introduced in Release 7.0

Some new terms have been introduced in Oracle7 that describe features of Oracle Version 6. These are new terms that better explain old concepts:

initialization parameters — The term *initialization parameter* now describes parameters that you use to specify configuration settings when starting an instance.

In Version 6 manuals, these parameters were commonly called INIT.ORA parameters.

schema — The term *schema* now describes the collection of objects owned by a user. Every user owns a schema in which objects can be created. The name of that schema is the same as the name of the user. The name of an object can be qualified by the schema in which the object exists. For example, the table EMP in the schema of the user SCOTT can be identified by SCOTT.EMP.

In Version 6 manuals, there was no distinction between a user and the collection of objects owned by the user. The name of an object could be qualified with the name of the user who owned it.

server processes

The term *server process* now describes a process that handles requests from user processes. A server process can be either dedicated to one user process or shared among many user processes, depending on the configuration of your instance.

In Version 6 manuals, these processes were called shadow processes.

Session Control commands

The term *Session Control commands* now describes a category of SQL commands that manage the properties of a session. This category includes the ALTER SESSION command (described in Version 6 manuals as a Data Definition Language command) and the new SET ROLE command.

system change number (SCN)

The term *system change number* now describes values that identify committed transactions.

In Version 6 manuals, these values were called system commit numbers. The new term is still abbreviated SCN.

System Control commands

The term *System Control commands* now describes a category of SQL commands that manage the properties of your Oracle instance. This category includes the new ALTER SYSTEM command.

Transaction Control commands

The term *Transaction Control commands* now describes a category of SQL commands that manage changes made by Data Manipulation Language commands. This category includes the COMMIT, ROLLBACK, and SAVEPOINT commands (described in Version 6 as Data Manipulation Language commands) and the SET TRANSACTION command (described in Version 6 manuals as a Data Definition Language command).

Reserved Words

This section lists changes to the SQL reserved words in Oracle7:

- new reserved words in Oracle7
- previously reserved words now obsolete

A complete list of all the SQL reserved words for Oracle7, begins on page 2 – 4.

New Reserved Words

Oracle7 has new SQL reserved words:

ROWLABEL	This reserved word is the name of a column automatically created by Trusted Oracle7 for all tables in the database. This column holds the label for each row in the table. For more information on ROWLABEL, see *Trusted Oracle7 Server Administrator's Guide*. In the standard Oracle7 Server, ROWLABEL is also a reserved word and always evaluates to null.
VARCHAR2	This reserved word is a datatype for variable length character strings. For more information on this datatype, see the section "Oracle Datatypes" beginning on page A – 15 and the section "Character Datatypes" on page 2 – 20.

Do not use these words to name objects or their parts in Oracle7.

Obsolete Reserved Words

Previous versions of Oracle contained SQL reserved words that are no longer reserved in Oracle7:

- GRAPHIC
- IF
- VARGRAPHIC

You can use these words as names of schema objects or object parts in Oracle7.

Oracle Datatypes

Oracle7 has new datatypes and changes to existing datatypes. This section discusses how Oracle7 treats these types of data:

- numeric data
- character data
- LONG data
- label data

Numeric Datatypes

Oracle7 returns an error if a numeric expression evaluates to a value greater than or equal to 10^{126} or less than or equal to -10^{126}. Oracle Version 6 returned a tilde (~) for a value outside these limits.

Character Datatypes

This section discusses the differences in Oracle Version 6 and Oracle7 character datatypes. For information on upgrading to Oracle7 with respect to these differences, see *Oracle7 Server Migration*.

In Oracle Version 6

Oracle Version 6 supported one datatype for character strings:

CHAR — Values of this datatype were variable length character strings of maximum length 255 characters. Oracle Version 6 compared CHAR values using non–padded comparison semantics.

Oracle Version 6 also supported these synonyms for the CHAR datatype:

- CHARACTER
- VARCHAR

In Oracle7

Oracle7 supports two datatypes for character strings:

CHAR — Values of this datatype are fixed length character strings of maximum length 255 characters. Oracle7 compares CHAR values using blank–padded comparison semantics. Note that the Oracle7 CHAR datatype is not equivalent to the Oracle Version 6 CHAR datatype.

VARCHAR2 — Values of this datatype are variable length character strings of maximum length 2000. Oracle7 compares VARCHAR2 values using non–padded comparison semantics. The VARCHAR2 datatype is equivalent to the Oracle Version 6 CHAR datatype except for the difference in maximum lengths.

☞ **Attention:** Oracle Version 6 only had the CHAR datatype available. In Version 6, VARCHAR and VARCHAR2 were synonyms for CHAR. Thus, the default datatype of character strings was CHAR. In Oracle7, the default character type is VARCHAR2.

Oracle7 also supports these synonyms for the CHAR and VARCHAR2 datatypes:

CHARACTER	This datatype is synonymous with the Oracle7 CHAR datatype.
VARCHAR	This datatype is currently synonymous with the VARCHAR2 datatype. However, Oracle Corporation recommends that you use VARCHAR2 rather than VARCHAR. In a future version of Oracle, VARCHAR may be a separate datatype used for variable length character strings compared with different comparison semantics.

For complete information on the Oracle7 datatypes, including the differences between blank–padded and non–padded comparison semantics, see the sections, "Character Datatypes," on page 2 – 20, and "Datatype Comparison Rules," on page 2 – 29.

LONG Datatype

The LONG datatype has new properties and fewer restrictions:

- The maximum length a LONG value is now 2 gigabytes, or $2^{31} - 1$ bytes, increased from 65,535 bytes.
- You can now use a distributed query to select a LONG column from a remote table or view.

For more information on the LONG datatype, see the section "LONG Datatype" on page 2 – 23.

Label Data

Labels are used by the Trusted Oracle7 to mediate access to information. The new MLSLABEL datatype is used to store representations of labels. For more information on these datatypes, see *Trusted Oracle7 Server Administrator's Guide.*

New Commands

These commands are new to the SQL language for Oracle7.

CREATE FUNCTION ALTER FUNCTION DROP FUNCTION	These commands have been added for stored functions.
CREATE PACKAGE CREATE PACKAGE BODY ALTER PACKAGE DROP PACKAGE	These commands have been added for stored packages.
CREATE PROCEDURE ALTER PROCEDURE DROP PROCEDURE	These commands have been added for stored procedures.
CREATE TRIGGER ALTER TRIGGER DROP TRIGGER	These commands have been added for database triggers.
ALTER VIEW	This command has been added to recompile views.
CREATE PROFILE ALTER PROFILE DROP PROFILE ALTER RESOURCE COST	These commands have been added for resource limits.
CREATE ROLE ALTER ROLE DROP ROLE SET ROLE CREATE USER DROP USER	These commands have been added for security.
CREATE SNAPSHOT ALTER SNAPSHOT DROP SNAPSHOT CREATE SNAPSHOT LOG ALTER SNAPSHOT LOG DROP SNAPSHOT LOG	These commands have been added for. snapshots.
ALTER SYSTEM	This command has been added to perform various specialized operations on an instance.

ANALYZE	This command has been added to collect statistics for cost–based optimization.
CREATE CONTROLFILE	This command has been added for recovery.
CREATE SCHEMA	This command has been added to added to issue multiple Data Definition Language statements in the same transaction.
TRUNCATE	This command has been added to added to quickly remove all rows from a table or cluster.

For complete information on each of these commands, see Chapter 4 “Commands” of this manual.

For a list of new embedded SQL commands for Oracle7, see *Programmer’s Guide to the Oracle Precompilers.*

Existing Commands with New Functionality

These commands were part of the SQL language for Oracle Version 6, but they have new syntax or functionality in Oracle7. For complete information on these commands, see the section describing the command in Chapter 4 of this manual. For a list of embedded SQL commands with new syntax or functionality for Oracle7, see *Programmer's Guide to the Oracle Precompilers.*

ALTER CLUSTER

This command has a new ALLOCATE EXTENT clause for dynamic free space management.

The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:

- In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and generated an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
- In Oracle7, if you specify a value greater than the maximum, Oracle generates an error immediately.

For complete information on this parameter, see the section describing the STORAGE clause on page 4 – 449.

ALTER DATABASE

This command now allows you to specify multiple copies of redo log files and has new clauses to manipulate multiple copies of redo log files:

- ADD LOGFILE MEMBER
- DROP LOGFILE MEMBER

This command also has these new clauses for managing multiple redo log files for multiple instances of the Oracle7 Parallel Server in parallel mode:

- ENABLE THREAD
- DISABLE THREAD

The ADD LOGFILE clause of this command also has a new THREAD parameter for this purpose.

This command also has a new PARALLEL option that replaces the SHARED option from Oracle Version 6.

This command also has the new BACKUP CONTROLFILE, CREATE DATAFILE, and RECOVER clauses for backup and recovery.

This command also has the new RENAME GLOBAL_NAME to change the database's global name.

This command also has a new SET clause to change the MAC mode or to establish the labels DBHIGH and DBLOW with Trusted Oracle7. For more information on this clause, see *Trusted Oracle7 Server Administrator's Guide*.

The CLOSE and DISMOUNT options of this command that were supported in previous versions are no longer supported. You should use the Server Manager SHUTDOWN command instead. For information on this command, see *Oracle Server Manager User's Guide*.

ALTER INDEX

The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:

- In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and generated an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
- In Oracle7, if you specify a value greater than the maximum, Oracle generates an error immediately.

For complete information on this parameter, see the section describing the STORAGE clause on page 4 – 449.

ALTER ROLLBACK SEGMENT

You need no longer specify the PUBLIC keyword to alter a public rollback segment, although Oracle still accepts this keyword for backward compatibility.

The STORAGE clause of this command has new syntax and functionality. For a summary of these changes, see the CREATE ROLLBACK SEGMENT command later in this list.

ALTER SESSION

This command has new parameters for National Language Support:

- NLS_LANGUAGE
- NLS_TERRITORY
- NLS_DATE_FORMAT
- NLS_DATE_LANGUAGE
- NLS_NUMERIC_CHARACTERS
- NLS_ISO_CURRENCY
- NLS_CURRENCY
- NLS_SORT

The equal sign (=) following the SQL_TRACE parameter is optional. Equal signs following all other parameters are mandatory.

This command also has a new GLOBAL_NAMES parameter to enable and disable global name resolution for remote objects. For more information on global name resolution, see Chapter "Database Administration" of *Oracle7 Server Distributed Systems, Volume I*.

This command also has a new LABEL parameter to change your DBMS session label and to change your default label format with Trusted Oracle7. For more information on this command, see *Trusted Oracle7 Server Administrator's Guide*.

This command also has a new OPTIMIZER_GOAL parameter to change:

- the optimization approach between the rule–based approach and the cost–based approach
- the goal of the cost–based approach between best throughput and best response time

In future versions of Oracle, the rule–based approach will not be available and this parameter will only specify the goal of the cost–based approach.

This command also has a new CLOSE DATABASE LINK clause to explicitly close an open database link.

This command also has a new ADVISE clause for sending advice for forcing in–doubt distributed transactions to remote databases.

This command also has a new COMMIT IN PROCEDURE clause for permitting or prohibiting COMMIT and ROLLBACK commands in procedures and stored functions.

ALTER TABLE

This command has a new ALLOCATE EXTENT clause for dynamic free space management.

The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:

- In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and generated an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
- In Oracle7, if you specify a value greater than the maximum, Oracle generates an error immediately.

For complete information on this parameter, see the section describing the STORAGE clause on page 4 – 449.

This command also has these new clauses to enable and disable integrity constraints and database triggers:

- ENABLE
- DISABLE

The CONSTRAINT clause of the ALTER TABLE command also has new syntax and functionality. For a summary of these changes, see the CREATE TABLE command later in this list.

DEFAULT values for columns were not enforced by Oracle Version 6. Oracle7 does enforce them. Oracle7 also ensures that a column is long enough to hold its DEFAULT value.

This command also has a new DROP clause for dropping integrity constraints.

For information on the ENABLE, DISABLE, CONSTRAINT, and DROP clauses, see the sections describing them in Chapter 4 “Commands” of this manual.

ALTER TABLESPACE

This command has a new OFFLINE TEMPORARY option. Also, the ONLINE option generates an error message if the tablespace requires media recovery, rather than performing the media recovery transparently.

The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:

- In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and returned an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
- In Oracle7, if you specify a value greater than the maximum, Oracle returns an error message immediately.

For information on this parameter, see the section describing the STORAGE clause on page 4 – 449.

ALTER USER

This command has new clauses to assign tablespaces, profiles, and default roles to users:

- QUOTA
- PROFILE
- DEFAULT ROLE

AUDIT (SQL Statements)

This form of the AUDIT command has many new system auditing options to support auditing of system operations with finer granularity.

AUDIT (Schema Objects)

This form of the AUDIT command has new object auditing options to support auditing of stored procedures, functions, and packages.

COMMIT

This command has new clauses for managing distributed transactions:

- COMMENT
- FORCE

CREATE CLUSTER

This command has these new parameters to create hash clusters:

- HASH
- HASHKEYS

The STORAGE clause of this command has new syntax and functionality:

- The maximum value of the MAXEXTENTS parameter of the STORAGE varies depending on your data block size:
  - In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and returns an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
  - In Oracle7, if you specify a value greater than the maximum, Oracle returns an error message immediately.
- This clause has these new parameters for managing free space:
  - FREELIST GROUPS
  - FREELISTS

For complete information on these parameters, see the section describing the STORAGE clause on page 4 – 449.

CREATE DATABASE

This command now allows you to specify redo log file groups containing multiple copies. This command also has these new parameters:

MAXLOGMEMBERS	This parameter specifies the maximum number of members in a single redo log file group.
MAXLOGHISTORY	This parameter specifies the maximum number of archived redo log file groups for automatic media recovery of the Oracle7 Parallel Server.
CHARACTER SET	This parameter specifies the database character set.

CREATE DATABASE LINK

The name of a database link must correspond to the name and domain of the remote database to which it connects. For more information on naming and referring to database links, see the section “Referring to Objects in Remote Databases” on page 2 – 11.

The USING clause of this command is now optional. This clause specifies the connect string to a remote database.

The USING clause also supports the specification of a secondary database for a read–only mount with Trusted Oracle7. For information on using this command with read–only mounts, see *Trusted Oracle7 Server Administrator’s Guide.*

When you issue a SQL statement that contains a database link, Oracle must determine both of these things before connecting to the remote database:

- a username and password (specified by the CONNECT TO clause of a CREATE DATABASE LINK statement)
- a database string (specified by the USING clause of a CREATE DATABASE LINK statement)

Oracle finds these things by first searching for private database links in your own schema with the same name as the database link in the statement, and then, if necessary, searching for a public database link with the same name.

Oracle always determines the username and password from the first matching database link (either private or public). If the first matching database link has an associated username and password, Oracle uses it. If it does not have an associated username and password, Oracle uses your current username and password.

If the first matching database link has an associated database string, Oracle uses it. If not, Oracle searches for the next matching (public) database link. If there is no matching database link, or if no matching link has an associated database string, Oracle returns an error message.

CREATE INDEX

Enforcing uniqueness among column values is now performed by integrity constraints. Oracle Corporation recommends that you use UNIQUE integrity constraints rather than unique indexes. Unique indexes may not be supported in future versions of Oracle.

The STORAGE clause of this command has new syntax and functionality:

- The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:
  - In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and returned an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
  - In Oracle7, if you specify a value greater than the maximum, Oracle returns an error message immediately.
- This clause has the new FREELISTS parameter for managing free space.

For complete information on these parameters, see the section describing the STORAGE clause on page 4 – 449.

CREATE ROLLBACK SEGMENT

This command has these changes to the STORAGE clause parameters:

- The PCTINCREASE parameter can no longer be specified for rollback segments. Rollback segments automatically have a PCTINCREASE value of 0.
- The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:
  - In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and returned an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
  - In Oracle7, if you specify a value greater than the maximum, Oracle returns an error message immediately.
- There is a new parameter OPTIMAL.

For complete information on these parameters, see the section describing the STORAGE clause on page 4 – 449.

CREATE TABLE

This command has these new clauses to enable and disable integrity constraints and triggers:

- ENABLE
- DISABLE

The CONSTRAINT clause of the CREATE TABLE command has new syntax and functionality:

- The optional CONSTRAINT identifier must appear at the beginning of the CONSTRAINT clause in Oracle7, rather than at the end as in Oracle Version 6.
- The new ON DELETE CASCADE option allows deletions of referenced key values from the parent table that have dependent rows in the child table and causes Oracle to delete the dependent rows to maintain referential integrity.
- The new DISABLE option allows you to disable an integrity constraint upon creation.
- The new USING INDEX option allows you to specify parameter values and storage characteristics for the index that Oracle7 uses to enforce a UNIQUE or PRIMARY KEY constraint.
- The new EXCEPTIONS INTO clause allows you to identify existing rows that violate a constraint.

Furthermore, Oracle Version 6 only enforced NOT NULL constraints. Oracle7 enforces all types of integrity constraints.

DEFAULT values for columns were not enforced by Oracle Version 6. Oracle7 does enforce them. Oracle7 also ensures columns are long enough to hold their DEFAULT values.

The STORAGE clause of this command has new syntax and functionality:

- The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:
  - In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and generated an error only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
  - In Oracle7, if you specify a value greater than the maximum, Oracle generates an error immediately.
- This clause has these new parameters for managing free space:
  - FREELIST GROUPS
  - FREELISTS

For complete information on the ENABLE, DISABLE, CONSTRAINT, and STORAGE clauses, see the sections describing them in Chapter 4 "Commands" of this manual.

CREATE TABLESPACE

The STORAGE clause of this command has new syntax and functionality:

- The maximum value of the MAXEXTENTS parameter of the STORAGE clause varies depending on your data block size:
  - In Oracle Version 6, if you specified a value that exceeded the maximum, Oracle stored the specified value in the data dictionary and returned an error message only if there is an attempt to allocate more extents than the maximum MAXEXTENTS value.
  - In Oracle7, if you specify a value greater than the maximum, Oracle returns an error message immediately.
- This clause has these new parameters for managing free space:
  - FREELIST GROUPS
  - FREELISTS

For complete information on these parameters, see the section describing the STORAGE clause on page 4 – 449.

CREATE VIEW

This command has these new options:

OR REPLACE	This option allows you to redefine a view without dropping and recreating it and regranting object privileges previously granted on it.
FORCE	This option allows you to create a view even if the tables, views, and snapshots that it queries do not exist.
NOFORCE	This option prevents you from creating a view if the tables, views, and snapshots that it queries do not exist. This is the default option and is equivalent to the behavior of Version 6.

The authorization of this command is slightly different in Oracle7 than in Oracle Version 6. In Oracle Version 6, a user granted the DBA system privilege could create a view based on any table in any schema. In Oracle7, a user granted the predefined DBA role can only create a view if the owner of the schema to contain the view is granted privileges to select, insert, update, or delete rows from the base table. These privileges must be granted directly, rather than through roles.

DELETE

This command now allows you to delete rows from a remote table or view using a database link.

DROP CLUSTER

This command has a new CASCADE CONSTRAINTS option to allow you to drop referential integrity constraints from tables outside the dropped cluster that refer to primary and unique keys in the tables of the cluster.

DROP ROLLBACK SEGMENT

You need no longer specify the PUBLIC keyword to drop a public rollback segment, although Oracle7 still accepts this keyword for backward compatibility.

DROP TABLE

This command has a new CASCADE CONSTRAINTS option to allow you to drop referential integrity constraints that refer to primary and unique keys in a dropped table.

EXPLAIN PLAN

The INTO clause of this command can now contain a remote table qualified by a database link.

The SQL statement in the FOR clause can now contain bind variables. Oracle assumes these bind variables are of datatype VARCHAR2.

GRANT (System Privileges and Roles)

In Oracle7, this form of the GRANT command is the same as Form I in Oracle Version 6. It also has many new system privileges to support security management with finer granularity. This form of the GRANT command can also administer roles.

In Oracle Version 6, the GRANT command (Form I) was also used to create users and change passwords. In Oracle7, you can use the CREATE USER and ALTER USER commands to perform these tasks. Oracle Corporation recommends that you use the CREATE USER and ALTER USER commands rather than the GRANT command. Using the GRANT command for these purposes may not be supported in future versions of Oracle. For information on using the GRANT command for these purposes, see the *SQL Language Reference Manual* for Oracle Version 6.

In Oracle Version 6, the GRANT command (Form II) gave users access to tablespaces. In Oracle7, you can only perform this task with the new TABLESPACE clause of the CREATE USER and ALTER USER commands.

GRANT (Object Privileges)

In Oracle7, this form of the GRANT command is the same as Form III in Oracle Version 6. This form of the command grants privileges on specific objects. In Oracle7, this form has new object privileges for security management of stored procedures, functions, and packages.

INSERT

This command now allows you to insert rows into a remote table or view using a database link.

LOCK TABLE

This command now allows you to lock a remote table or view using a database link.

NOAUDIT

Changes to the NOAUDIT command correspond directly to the changes to the AUDIT command listed earlier in this section.

REVOKE

Changes to the REVOKE command correspond directly to the changes to the GRANT command listed earlier in this section.

ROLLBACK

This command has a new FORCE clause for managing distributed transactions.

SELECT

Oracle7 places fewer restrictions on distributed queries than Oracle Version 6. For complete information on distributed queries, see the section, "Distributed Queries," on page 4 – 436.

In Oracle Version 6, you could specify a column of a remote table in the select list using this syntax:

```
table@dblink.column
```

Since Oracle7 interprets all characters following @ to be the complete name of a database link, you cannot use this syntax in Oracle7. For example, you can issue this query in Oracle Version 6, but not in Oracle7:

```
SELECT emp@boston.ename
    FROM emp@boston
```

Oracle7 interprets 'boston.ename' to be the complete name of a database link. In Oracle7, you can instead issue one of these equivalent queries also accepted by Oracle Version 6:

```
SELECT e.ename
    FROM emp@boston e
SELECT ename
    FROM emp@boston
```

You can also issue this equivalent query that was not acceptable in Oracle Version 6:

```
SELECT emp.ename@boston
    FROM emp@boston
```

Also, in Oracle Version 6, you could qualify a *table.column* expression with a schema in the select list regardless of whether the table was qualified with a schema in the FROM clause. In Oracle7, you can only qualify a *table.column* expression with a *schema* if the table is qualified with a *schema* in the FROM clause. For example, you could issue this query in Oracle Version 6, but not in Oracle7:

```
SELECT scott.emp.ename
    FROM emp
```

Oracle7 places more restrictions on the WHERE clause conditions of SELECT statements that perform outer joins:

- The OR logical operator cannot combine two conditions if either contains the outer join operator (+). Also, a condition cannot use the IN logical operator to compare a column marked with the (+) operator to another expression. If you have applications that issue queries with such conditions, replace them with equivalent queries that use the UNION or UNION ALL set operators instead.
- If a condition compares a column marked with the (+) operator to a subquery, Oracle7 returns an error message. Oracle Version 6 ignored the (+) operator in such conditions. If you have applications that issue queries with such conditions, remove the (+) operator from them and they will behave in Oracle7 as they did in Oracle Version 6.

SET TRANSACTION

This command has these new options:

READ WRITE	This option establishes the current transaction as a read–write transaction in which data can be both queried and modified, as opposed to a read–only transaction in which data can only be queried and not modified. Oracle establishes a read–write transaction by default if you do not issue a SET TRANSACTION statement.
USE ROLLBACK SEGMENT	This option allows you to assign your current transaction to a specific rollback segment.

UPDATE

This command now allows you to update values in remote tables and views using a database link.

VALIDATE INDEX

Validating indexes is now also performed by the new ANALYZE command. Oracle Corporation recommends that you use the ANALYZE command rather than the VALIDATE INDEX command. The VALIDATE INDEX command may not be supported in future versions of Oracle. For information on the VALIDATE INDEX command, see the *SQL Language Reference Manual* for Oracle Version 6.

SQL Functions

This section lists:

- new SQL functions added for Oracle7
- existing SQL functions with new functionality

New SQL Functions

These new SQL functions have been added for Oracle7:

- SIN
- COS
- TAN
- SINH
- COSH
- TANH
- EXP
- LN
- LOG
- CONCAT
- INSTRB
- LENGTHB
- SUBSTRB
- NLS_INITCAP
- NLS_LOWER
- NLS_UPPER
- TO_MULTI_BYTE
- TO_SINGLE_BYTE

These new SQL functions have been added for Trusted Oracle7:

- GLB
- LUB
- TO_LABEL
- GREATEST_LB
- LEAST_UB

Existing SQL Functions with New Functionality

These functions have been enhanced for Oracle7:

- The POWER function now allows non–integral exponents.
- The NLSSORT function now accepts the optional NLS_SORT parameter for National Language Support.
- The TO_CHAR function now accepts the optional parameters NLS_DATE_LANGUAGE, NLS_NUMERIC_CHARACTERS, NLS_CURRENCY, and NLS_ISO_CURRENCY for National Language Support.
- In Trusted Oracle7, the TO_CHAR function converts values with the datatypes MLSLABEL or RAW MLSLABEL to values with the datatype VARCHAR2.
- The TO_DATE function now accepts the optional NLS_DATE_LANGUAGE parameter for National Language Support.
- The TO_NUMBER function now accepts the parameters NLS_NUMERIC_CHARACTERS, NLS_CURRENCY, and NLS_ISO_CURRENCY for National Language Support.

For complete information on these functions, see the section "Functions" on page 3 – 17.

Format Models

These new number format elements have been added to SQL for Oracle7:

- D
- G
- L
- C
- RN

These new date format elements have been added to SQL for Oracle7:

- IYYY, IYY, IY, I
- IW
- RM
- RR

If you used National Language Support in Oracle Version 6, the WW date format element may behave differently in Oracle7. In Version 6, depending on the *territory* component of the value of the LANGAUGE initialization parameter, WW returned a week number based on either the ISO standard or the number of days from January 1. In Oracle7, WW always returns a week number based on the number of days from January 1, regardless of the value of the NLS_TERRITORY initialization parameter, and the new IW date format element returns the ISO standard week number. If your Version 6 application used WW to return the ISO standard week number, replace WW with IW.

Oracle7 also has a new format model modifier FX and new functionality for the FM format model modifier. For information on format models, see the section "Format Models" on page 3 – 59.

Operators

This section describes:

- new operators
- existing operators with changes in functionality

New Operators

These new operators have been added to SQL for Oracle7:

SOME	This new comparison operator is synonymous with the ANY comparison operator.
UNION ALL	This new set operator combines two queries and returns all rows returned by either query, including all duplicate rows. The UNION ALL operator is similar to the UNION operator, except the UNION operator returns only one copy of duplicate rows.

Existing Operators with Functional Changes

The functionality of these existing operators has changed for Oracle7:

–	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. If you have applications that issue SQL statements with such arithmetic expressions, separate the minus signs with a space or a parenthesis.
LIKE	The LIKE operator accepts the new ESCAPE option, which allows you to use the characters % and _ literally, rather than as special pattern matching characters, within a pattern.
(+)	The outer join operator is subject to new restrictions listed in the section describing the SELECT command earlier in this chapter.

Comments

Oracle7 supports comments within SQL statements beginning with — as well as comments beginning with /*. For more information on comments within SQL statements, see the section "Comments" beginning on page 2 – 43.

Namespaces

This section describes:

- changes to namespaces for schema objects
- changes to namespaces for other objects

Changes to Namespaces for Schema Objects

Figure A – 1 shows the namespaces for schema objects in Oracle Version 6:

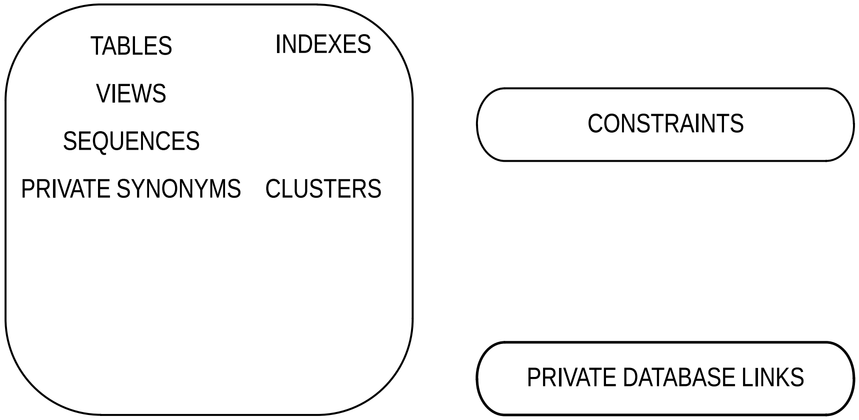


Figure A – 1 Namespaces for Schema Objects in Oracle Version 6

For Oracle7, changes have been made to these namespaces:

- Stand–alone procedures, stand–alone stored functions, packages, and snapshots have been added to the namespace containing tables.
- Indexes have been moved from the namespace containing tables to a new namespace.
- Clusters have been moved from the namespace containing tables to a new namespace.
- Database triggers have been added in a new namespace.

These changes are shown in bold in Figure A – 2.

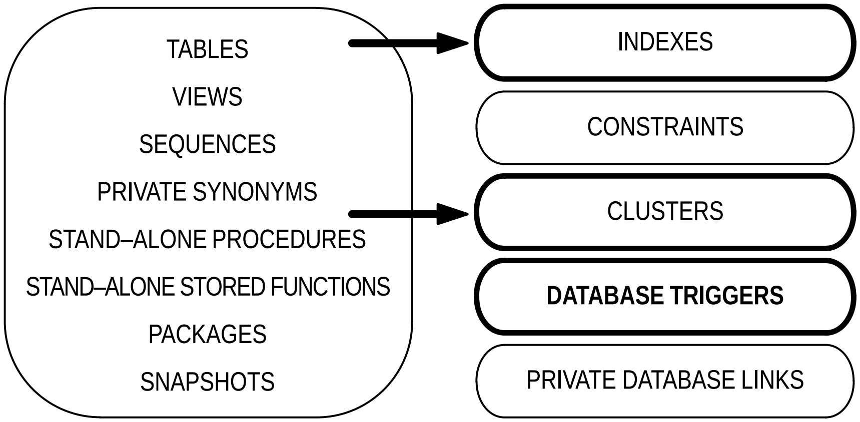


Figure A – 2 Changes in Namespaces for Schema Objects for Oracle7

Changes to Namespaces for Other Objects

Figure A – 3 shows the namespaces for other objects in Oracle Version 6:

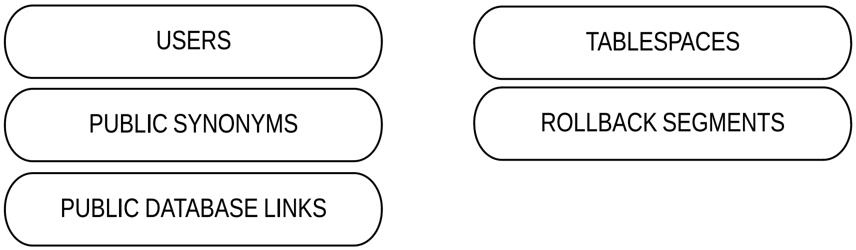


Figure A – 3 Namespaces for Other Objects in Oracle Version 6

For Oracle7, changes have been made to these namespaces:

- Roles have been added to the namespace containing users.
- Profiles have been added to a new namespace.

These changes are shown in bold in Figure A – 4.

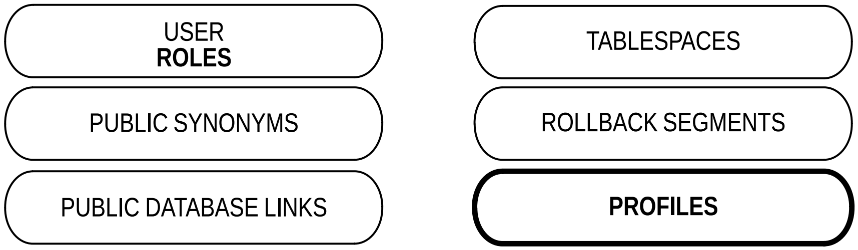


Figure A – 4 Changes in Namespaces for Other Objects in Oracle7

Changes to the Optional Components of Oracle

This section discusses the differences in the optional components between Oracle Version 6 and Oracle7.

With Oracle Version 6, the transaction processing option was available. This option included these features:

- row–level locking
- PL/SQL

With Oracle7, the transaction processing option is obsolete. However, these options are available:

procedural option	This option includes PL/SQL and allows you to use anonymous PL/SQL blocks, stored procedures, stored functions, stored packages, and database triggers.
distributed option	This option allows you to issue Data Manipulation Language (DELETE, EXPLAIN PLAN, LOCK TABLE, INSERT, and UPDATE) statements that modify data on remote databases.
Parallel Server option	This option allows multiple Oracle instances to mount an Oracle7 database in parallel mode. This functionality was also available in Oracle Version 6.2.

To use snapshots, you must have both the procedural option and the distributed option. All other features of Oracle7 (including row–level locking) are available in all installations and do not require one of these options.

Compatibility Modes

The compatibility mode controls Oracle7's behavior in a few areas for which there are minor differences between Oracle Version 6 and Oracle7. Oracle7 can operate in these compatibility modes:

V7 compatibility mode
: In this mode, Oracle interprets SQL exactly as described in this manual.

V6 compatibility mode
: In this mode, Oracle interprets SQL as described in this manual, with some exceptions for compatibility with Oracle Version 6.

Table 4 – 14 describes the differences between V6 and V7 compatibility modes:

V6 Compatibility Mode	V7 Compatibility Mode
If you define a column of datatype CHAR, Oracle creates the column with the Oracle7 VARCHAR2 datatype, which is equivalent to the Oracle Version 6 CHAR datatype. The column is a variable–length character string with non–padded comparison semantics and a maximum length of 2000 bytes.	If you define a column of datatype CHAR, Oracle creates the column with the Oracle7 CHAR datatype, which is not equivalent to the Oracle Version 6 CHAR datatype. The column is fixed–length character string with blank–padded comparison semantics and a maximum length of 255 bytes.
The optimal CONSTRAINT identifier can only appear at the end of a CONSTRAINT clause.	The optional CONSTRAINT identifier can only appear at the beginning of a CONSTRAINT clause.
By default, PRIMARY KEY, UNIQUE, referential integrity, and CHECK constraints are disabled upon creation. NOT NULL constraints are enabled upon creation by default.	By default, all integrity constraints are enabled upon creation.

Table 4 – 14 Differences Between V6 and V7 Compatibility Modes

V6 Compatibility Mode	V7 Compatibility Mode
If you specify a PCTINCREASE value for a rollback segment, Oracle ignores this value and uses a value of 0.	If you specify a PCTINCREASE value for a rollback segment, Oracle returns an error.
If you specify a MAXEXTENTS value that exceeds the maximum possible value based on the data block size, Oracle ignores the specified value and uses the maximum possible value.	If you specify a MAXEXTENTS value that exceeds the maximum possible value based on the data block size, Oracle returns an error.

Table 4 – 14 Differences Between V6 and V7 Compatibility Modes

There are additional differences between the V6 and V7 compatibility modes that are specific to the Oracle Precompilers and the Oracle Call Interfaces (OCIs). For information on these differences, see *Programmer's Guide to the Oracle Precompilers* and *Programmer's Guide to the Oracle Call Interface.*

Migrating to Oracle7

You may want to establish V6 compatibility mode when you initially upgrade to Oracle7 in order ease the migration of your existing Oracle Version 6 applications. Establishing V6 compatibility mode reduces (but does not eliminate) the number of changes you may have to make to your applications before running them on Oracle7. Note that there is some SQL syntax supported by Oracle Version 6 that is not supported by Oracle7 in either V6 or V7 compatibility mode. If you have existing applications that you have run on Oracle Version 6, see *Oracle7 Server Migration* for a list of the changes that you must make to these applications before running them on Oracle7.

You should eventually upgrade your applications so that they can be run in V7 compatibility mode, rather than V6 compatibility mode.

Establishing and Switching Between Compatibility Modes

By default, all sessions on Oracle7 initially run in V7 compatibility mode. Some Oracle application tools allow you to establish and switch between compatibility modes for your sessions. For information on how to establish and switch between compatibility modes, see the manual for the specific tool. For example, to find out how to switch between compatibility modes with SQL*Plus, see *SQL*Plus User's Guide and Reference.*

APPENDIX

B

Oracle and Standard SQL

This appendix discusses the following topics:

- Oracle's conformance to the SQL standards established by industry standards governing bodies
- Oracle's extensions to standard SQL
- locating extensions to standard SQL with the FIPS Flagger

Conformance with Standard SQL

This section declares Oracle's conformance to the SQL standards established by these organizations:

- American National Standards Institute (ANSI)
- International Standards Organization (ISO)
- United States Federal Government

Conformance with these standards is measured by the National Institute of Standards and Technology (NIST) "SQL Test Suite". NIST is an organization of the government of the United States of America.

ANSI and ISO Compliance

Oracle7 conforms to Entry level conformance defined in the ANSI document, X3.135–1992, "Database Language SQL." You can obtain a copy of the ANSI standard from this address:

American National Standards Institute
1430 Broadway
New York, NY 10018
USA

The ANSI and ISO SQL standards require conformance claims to state the type of conformance and the implemented facilities. The Oracle7 Server, the Oracle Precompilers Version 1.5, and SQL*Module Version 1.0 provide conformance with the ANSI X3.135–1992 / ISO 9075–1992 standard:

- Compliance at Entry Level
  (including both SQL–DDL and SQL–DML)
- Module Language
- Embedded SQL Ada
- Embedded SQL C
- Embedded SQL COBOL
- Embedded SQL FORTRAN
- Embedded SQL Pascal
- Embedded SQL PL/I
- Full implementation of the Integrity Enhancement Feature

FIPS Compliance

Oracle complies completely with FIPS PUB 127–2 for Entry SQL. In addition, the following information is provided for Section 16, "Special Procurement Considerations." Oracle complies completely with FIPS PUB 127, providing SQL conformance as described above. In addition, this information is provided regarding Section 13 "Special Procurement Considerations" of FIPS PUB 127.

Section 16.2 Programming Language Interfaces
The Oracle Precompilers support the use of Embedded SQL. SQL*Module supports the use of Module Language. Support is provided for Ada, C, COBOL, FORTRAN, and Pascal.

Section 16.3 Style of Language Interface
Oracle with SQL*Module supports Module Language for Ada, C, COBOL, FORTRAN, and Pascal. Oracle with the Oracle Precompilers supports Ada, C, COBOL, FORTRAN, and Pascal. The languages supported may vary depending on your operating system.

Section 16.5 Interactive Direct SQL
Oracle7 with SQL*Plus Version 3.1 (as well as other Oracle tools) supports "direct invocation" of the following SQL commands, meeting the requirements of FIPS PUB 127–2:

- CREATE TABLE command
- CREATE VIEW command
- GRANT command
- INSERT command
- SELECT command, with ORDER BY clause but not INTO clause
- UPDATE command: searched
- DELETE command: searched
- COMMIT WORK command
- ROLLBACK WORK command

Most other SQL commands described in this Manual are also supported interactively.

Section 16.6 Sizing for Database Constructs

Table 4 – 15 lists requirements identified in FIPS PUB 127–1 and how they are met by Oracle7.

Length of an identifier (in bytes)	18	30
Length of CHARACTER datatype (in bytes)	240	255
Decimal precision of NUMERIC datatype	15	38
Decimal precision of DECIMAL datatype	15	38
Decimal precision of INTEGER datatype	9	38
Decimal precision of SMALLINT datatype	4	38
Binary precision of FLOAT datatype	20	126
Binary precision of REAL datatype	20	63
Binary precision of DOUBLE PRECISION datatype	30	126
Columns in a table	100	254
Values in an INSERT statement	100	254
Set clauses in an UPDATE statement (Note 1)	20	254
Length of a row (Note 2, 3)	2000	2(254) + 2^{31} +253(2000)
Columns in a UNIQUE constraint	6	16
Length of a UNIQUE constraint (Note 2)	120	(Note 4)
Length of foreign key column list (Note 2)	120	(Note 4)
Columns in a GROUP BY clause	6	255 (Note 5)
Sort specifications in ORDER BY clause	6	255 (Note 5)
Columns in a referential integrity constraint	6	16
Tables referenced in a SQL statement	10	No limit
Cursors simultaneously open	10	(Note 6)
Items in a SELECT list	100	255

Table 4 – 15 Sizing for Database Constructs

[1] The number of set clauses in an UPDATE statement refers to the number items separated by commas following the SET keyword.

[2] The FIPS PUB defines the length of a collection of columns to be the sum of: twice the number of columns, the length of each character column in bytes, decimal precision plus 1 of each exact numeric column, binary precision divided by 4 plus 1 of each approximate numeric column.

[3] The Oracle limit for the maximum row length is based on the maximum length of a row containing a LONG value of length 2 gigabytes and 253 VARCHAR2 values, each of length 2000 bytes.

[4] The Oracle limit for a UNIQUE key is half the size of an Oracle data block (specified by the initialization parameter DB_BLOCK_SIZE) minus some overhead.

[5] Oracle places no limit on the number of columns in a GROUP BY clause or the number of sort specifications in an ORDER BY clause. However, the sum of the sizes of all the expressions in either a GROUP BY or an ORDER BY clause is limited to the size of an Oracle data block (specified by the initialization parameter DB_BLOCK_SIZE) minus some overhead.

[6] The Oracle limit for the number of cursors simultaneously opened is specified by the initialization parameter OPEN_CURSORS. The maximum value of this parameter depends on the memory available on your operating system and exceeds 100 in all cases.

Section 16.7 Character Set Support
Oracle supports the ASCII character set (FIPS PUB 1–2) on most computers and the EBCDIC character set on IBM mainframe computers. Oracle supports both single–byte and multi–byte character sets.

Extensions to Standard SQL

This section lists the additional features supported by Oracle that extend beyond standard SQL "Database Language SQL with Integrity Enhancement". This section provides information on these parts of the SQL language:

- commands
- functions
- operators
- pseudocolumns
- datatypes
- names of schema objects
- values

For information on the extensions to standard embedded SQL "Database Language Embedded SQL" supported by the Oracle Precompilers, see *Programmer's Guide to the Oracle Precompilers.*

Commands

This section describes these additional commands and additional syntax and functionality of standard commands. Oracle supports these commands that are not part of standard SQL:

```
ALTER CLUSTER
ALTER DATABASE
ALTER FUNCTION
ALTER INDEX
ALTER PACKAGE
ALTER PROCEDURE
ALTER PROFILE
ALTER RESOURCE COST
ALTER ROLLBACK SEGMENT
ALTER ROLE
ALTER SEQUENCE
ALTER SESSION
ALTER SNAPSHOT
ALTER SNAPSHOT LOG
ALTER SYSTEM
ALTER TABLE
ALTER TABLESPACE
ALTER TRIGGER
ALTER USER
ALTER VIEW

ANALYZE

AUDIT

COMMENT

CREATE CONTROLFILE
CREATE CLUSTER
CREATE DATABASE
CREATE DATABASE LINK
CREATE FUNCTION
CREATE INDEX
CREATE PACKAGE
CREATE PACKAGE BODY
CREATE PROCEDURE
CREATE PROFILE
CREATE ROLLBACK SEGMENT
CREATE ROLE
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 ROLLBACK SEGMENT
DROP ROLE
DROP SEQUENCE
DROP SNAPSHOT
DROP SNAPSHOT LOG
DROP SYNONYM
DROP TABLE
DROP TABLESPACE

EXPLAIN PLAN

NOAUDIT

RENAME

REVOKE

SAVEPOINT

SET TRANSACTION

TRUNCATE
```

Additional Parts of Standard Commands

Oracle supports additional syntax for some commands that are part of standard SQL.

COMMIT

The COMMIT command supports these additional clauses:

- COMMENT clause
- FORCE clause

Also, standard SQL requires a COMMIT statement to include the WORK keyword. Oracle allows your COMMIT statements to either include or omit this keyword. Note that this keyword adds no functionality to the command.

CREATE TABLE

The CREATE TABLE command supports these additional parameters and clauses:

- PCTFREE parameter
- PCTUSED parameter
- INITRANS parameter
- MAXTRANS parameter
- TABLESPACE parameter
- STORAGE clause
- CLUSTER clause
- ENABLE clause
- DISABLE clause
- AS clause

CONSTRAINT Clause The CONSTRAINT clause of the CREATE TABLE command supports these additional options and identifiers:

- ON DELETE CASCADE option
- ENABLE option
- DISABLE option
- CONSTRAINT identifier

CREATE VIEW The CREATE VIEW command supports this additional syntax:

- OR REPLACE option
- FORCE and NOFORCE options
- CONSTRAINT identifier with the WITH CHECK OPTION

If you omit column names from a CREATE VIEW statement, the column aliases that appear in the defining query are used for columns of the view. Standard SQL does not support column aliases in SELECT statements.

DELETE The DELETE command supports this additional syntax:

- Database links to delete rows from tables and views on remote databases
- Table aliases for use with correlated queries

Also, standard SQL requires a DELETE statement to include the FROM keyword. Oracle allows your DELETE statements to either include or omit this keyword. Note that this keyword adds no functionality to the command.

GRANT The GRANT command (System Privileges and Roles) is an extension to standard SQL.

The GRANT command (Object Privileges) supports other privileges on other objects in addition to the DELETE, INSERT, REFERENCES, SELECT, and UPDATE privileges on tables and views supported by standard SQL. This command also supports granting object privileges to roles.

INSERT The INSERT command supports the use of database links to insert rows into tables and views on remote databases.

The INSERT command supports a subquery in the INTO clause, similar to inserting into a view.

ROLLBACK The ROLLBACK command supports these additional clauses:

- TO clause
- FORCE clause

Also, standard SQL requires a ROLLBACK statement to include the WORK keyword. Oracle allows your ROLLBACK statements to either include or omit this keyword. Note that this keyword adds no functionality to the command.

SELECT The SELECT command supports these additional clauses and syntax:

- START WITH clause
- CONNECT BY clause
- FOR UPDATE clause
- Database links for querying tables, views, and snapshots on remote databases
- Outer join operator (+) for performing outer joins
- Column aliases in the select list
- NULL in the select list

GROUP BY Clause The GROUP BY clause of the SELECT command supports this additional syntax and functionality:

- A SELECT statement that selects from a view whose defining query contains group functions or a GROUP BY clause can contain group functions and GROUP BY, HAVING, and WHERE clauses.
- A SELECT statement can perform a join involving a view whose defining query contains a GROUP BY clause.

ORDER BY Clause The ORDER BY clause of the SELECT command supports this additional syntax and functionality:

- This clause can also specify any expression involving any columns in any tables or views that appear in the FROM clause, rather than only select list expressions or positions of select list expressions.
- This clause can qualify a column name with its table or view name, using the syntax *table.column* or *view.column*.

Queries Queries, or forms of the SELECT command that appear inside other SQL statements, support this additional functionality:

- Queries can contain the GROUP BY clause.
- Queries can select from views whose defining queries contain the GROUP BY clause.

UPDATE The UPDATE command supports this additional syntax:

- Database links to update data in tables and views on remote databases
- Table aliases for use with correlated queries
- Parenthesized lists of columns on the left side of the SET clause, rather than only single columns
- Queries on the right side of the SET clause, rather than only expressions

The UPDATE command also supports this additional functionality:

- An UPDATE statement that updates a view can contain a query.
- A query within an UPDATE statement can refer to the table or view being updated.
- If the columns of a view are based on both columns of the base table and expressions containing columns of the base table, an UPDATE statement can update values based on columns, but not values based on expressions. Standard SQL prohibits all updates to such views.

Functions

This section describes additional functions and additional functionality of standard functions.

Additional Functions

The only standard SQL functions are AVG, COUNT, MAX, MIN, and SUM. Oracle supports many additional functions that are not part of standard SQL. See section “Functions” on page 3 – 17.

Additional Functionality of Standard Functions

You can nest group functions in the select list of a SELECT statement, as in this example:

```
SELECT MIN(MAX(sal))
   FROM emp
   GROUP BY deptno
```

The depth of nesting cannot be more than that shown in the example.

You can also use a group function in a SELECT statement that queries a view whose defining query contains group functions or a GROUP BY clause.

Operators

This section describes additional operators and additional functionality of standard operators.

Additional Operators

Oracle supports these operators that are not part of standard SQL:

- | | character operator (character concatenation)
- !=, ^=, and ¬= comparison operators (inequality)
- MINUS set operator
- INTERSECT set operator
- (+) operator (outer join)
- PRIOR operator

Additional Functionality of Standard Operators

Oracle supports additional functionality for standard SQL operators:

- The left member of an expression containing the IN operator can be a parenthesized list of expressions, rather than only a single expression.
- Any expression, rather than only a column, can be used with the comparison operators IS NULL and IS NOT NULL.
- The pattern used with the LIKE operator can be any expression of datatype CHAR or VARCHAR2, rather than only a text literal.

Pseudocolumns

Pseudocolumns are values that behave like columns of a table but are not actually stored in the table. Pseudocolumns are supported by Oracle, but are not part of standard SQL. For a list of pseudocolumns, see the section "Pseudocolumns" on page 2 – 38.

Datatypes

Oracle supports these additional datatypes that are not part of standard SQL:

- DATE
- NUMBER
- VARCHAR2
- LONG
- RAW
- LONG RAW
- ROWID

Oracle also supports automatic conversion of values from one datatype to another that is not part of standard SQL.

Names of Schema Objects

Oracle supports additional functionality for names of schema objects:

- Oracle supports names of maximum length 30 bytes, rather than 18 characters.
- Oracle allows you to enter names in either lowercase or uppercase, rather than only in lowercase. However, note that names are not case–sensitive unless they are in double quotes.
- Oracle supports names in double quotes. Quoted identifiers allow you to use:
  - names that are reserved words
  - names that are case–sensitive
  - names that contain spaces
- Oracle supports names that contain the special characters # and $ and repeated underscores (__).

Values

Oracle allows you to use either uppercase "E" or lowercase "e" for exponential notation of numeric values, rather than only "E".

FIPS Flagger

In your Oracle applications, you can use the extensions listed in the previous sections just as you can use standard SQL. If you are concerned with the portability of your applications to other implementations of SQL, use Oracle's FIPS Flagger to locate Oracle extensions to standard SQL in your embedded SQL programs. The FIPS Flagger is part of the Oracle Precompilers and the SQL*Module compiler. For information on how to use the FIPS Flagger, see *Programmer's Guide to the Oracle Precompilers* or *SQL*Module User's Guide and Reference.*

APPENDIX

C

Operating System–Specific Dependencies

This manual occasionally refers to other Oracle7 manuals that contain detailed information for using Oracle7 only on a specific operating system. These Oracle7 manuals are often called installation or user's guides, although the exact name may vary among operating systems.

This appendix lists all the references in this manual to operating system–specific Oracle manuals.

For the information on these topics appropriate for your operating system, see your Oracle7 installation or user's guide.

Topic	*Page*
Concatenation operator (usually ||)	
CREATE CONTROLFILE	4 – 173
MAXDATAFILES default, maximum	
MAXINSTANCES default, maximum	
MAXLOGFILES default, minimum, maximum	
MAXLOGMEMBERS default, maximum	
MAXLOGHISTORY default	
CREATE DATABASE	4 – 178
DATAFILE default	
LOGFILE default	
MAXDATAFILES default, maximum	
MAXINSTANCES default, maximum	
MAXLOGFILES default, minimum, maximum	
MAXLOGMEMBERS default, maximum	
MAXLOGHISTORY default	
CHARACTERSET supported and default	
CREATE TABLESPACE	4 – 254
REUSE and raw devices	
CREATE USER	4 – 267
OS_ROLES initialization parameter	
IDENTIFIED EXTERNALLY	
SET ROLE	
database administration roles	
CREATE TRIGGER	4 – 257
DBMSTDX.SQL	
filenames	4 – 343

Topic	*Page*
SQL scripts	
CATEXP.SQL	4 – 353
DBMSSNAP.SQL	4 – 230
	4 – 238
DBMSSTDX.SQL	4 – 188
	4 – 198
	4 – 202
	4 – 206
	4 – 257
SQL.BSQ	4 – 352
UTLCHAIN.SQL	4 – 339
UTLEXCPT.SQL	4 – 329
UTLSAMPL.SQL	pref viii
UTLXPLAN.SQL	4 – 339
STORAGE clause	4 – 449
INITIAL maximum	
NEXT maximum	
MINEXTENTS maximum	
PCTINCREASE maximum	
OPTIMAL maximum	
ROWID component lengths	2 – 27

Index

B

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Reader's Comment Form

Oracle7™ Server SQL Reference
Part No. A32538–1

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

Please send your comments to:

Oracle7 Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065 U.S.A.
Fax: (415) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

Thank you for helping us improve our documentation.