Oracle® Database Lite

SQL Reference

10*g* (10.2.0)

Part No. B15917-01

June 2005

ORACLE®

Oracle Database Lite SQL Reference 10*g* (10.2.0)

Part No. B15917-01

Contents

2 SQL Operators

3 SQL Functions

4 SQL Commands

Send Us Your Comments

Oracle Database Lite SQL Reference 10*g* (10.2.0)

Part No. B15917-01

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 title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: helplite_us@oracle.com
- FAX: (650) 506-7355. Attn: Oracle Database Lite 10*g*
- Postal service:

  Oracle Corporation
  Oracle Database Lite Documentation Manager
  500 Oracle Parkway, Mailstop 1op2
  Redwood Shores, CA 94065
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This preface introduces the *Oracle Database Lite SQL Reference*. This reference describes the Structured Query Language (SQL) used to manage information in an Oracle Database Lite database.

Oracle SQL is a superset of the SQL-92 standard defined by the American National Standards Institute (ANSI) and the International Standards Organization (ISO).

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

The following topics are discussed:

- Chapter 1, "Using SQL"

  Lists and defines important differences between Oracle Lite SQL and Oracle SQL, and explains how to use SQL.

- Chapter 2, "SQL Operators"

Describes the following types of Oracle Lite SQL operators: arithmetic, character, comparison, logical, set, and other.

- Chapter 3, "SQL Functions"

  Describes the following types of Oracle Lite SQL functions: number, character, character returning number values, date, conversion, group, other.

- Chapter 4, "SQL Commands"

  Describes the following types of Oracle Lite SQL commands: Data Definition Language (DDL), Data Manipulation Language (DML), transaction control, clauses, and pseudocolumns.

- Appendix A, "Oracle Database Lite Keywords and Reserved Words"

  Contains a list and description of Oracle Lite keywords and reserved words.

- Appendix B, "SQL Limitations For Oracle Database Lite"

  Contains the SQL limitations for the Oracle Lite database.

- Appendix C, "Oracle Database Lite Datatypes"

  Contains a list and description of Oracle Lite datatypes.

- Appendix D, "Oracle Database Lite Literals"

  Contains a list and description of Oracle Lite literals.

- Appendix E, "Index Creation Options"

  Describes additional options for the CREATE INDEX command.

- Appendix F, "Syntax Diagram Conventions"

  Explains the syntax diagrams and document conventions used in the *Oracle Database Lite SQL Reference*.

1 Using SQL

This document discusses how SQL is used with Oracle Database Lite. Topics include:

1.1 SQL Overview

Oracle Database Lite uses the SQL (Structured Query Language) database language to store and retrieve data. It includes the following categories of SQL statements:

- DDL (Data Definition Language)

  Used to create, alter, or drop database objects, such as schemas, tables, columns, views, and sequences. For example, statements that use the commands, `ALTER`, `CREATE`, `DROP`, `GRANT`, and `REVOKE`.

- DML (Data Manipulation Language)

  Used to query and manipulate data in existing schema objects. For example, statements that use the commands, `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.

- TCL (Transaction Control Language)

  These statements manage changes made in DML statements. For example, statements that use the commands, `COMMIT`, `ROLLBACK`, and `SAVEPOINT`.

- Clause

  Subsets of commands that modify commands. Oracle Lite supports `CONSTRAINT` and `DROP` clauses.

- Pseudocolumns

  Values generated from commands that behave like columns of a table but are not actually stored in the table. Oracle Database Lite supports the `LEVEL` and `ROWNUM` pseudocolumns.

- Functions

  Operate on data to transform or aggregate it. For example, `TO_DATE` to transform a date column into a particular format, and `SUM` to total all values for a column.

1.1.1 Examples

This reference provides SQL statement examples. All examples are based on the default Oracle Database Lite objects.

1.1.2 Oracle SQL and SQL-92

Oracle Database Lite uses Oracle SQL as its default SQL language. Oracle SQL handles computation results and date data in a different manner than SQL-92. The differences between Oracle SQL and SQL-92 are listed in Table 1–1.

Table 1–1 Differences Between Oracle SQL and SQL-92

Oracle SQL	SQL-92
Division yields a double precision result such as 3.333. For example 8/3 yields 2.666.	Division yields datatypes of operands such as 3. For example, 8/3 yields 2.
`DATE` datatype stores full timestamp information but only displays the date portion.	`DATE` datatype stores and displays date but no timestamp information.

Although Oracle Database Lite uses Oracle SQL, by default it supports several SQL-92 features including:

- Column datatypes: `TIME`, `TIMESTAMP`, `TINYINIT`, and `BIT`
- `CASE` expression
- `CAST` expression

1.1.2.1 Running SQL-92 on Oracle Lite

As mentioned in the preceding section, Oracle Database Lite uses Oracle SQL by default. However, if you want to support SQL-92 by default instead of Oracle SQL, you can change the SQL compatibility parameter in the `POLITE.INI` file to SQL-92. To change the parameter, add the following in the `POLITE.INI` file.

```
SQLCOMPATIBILITY=SQL92
```

See the *Oracle Database Lite Administration and Deployment Guide* for more information about the `POLITE.INI` file.

1.2 Oracle Lite SQL and Oracle SQL Comparison

The SQL language supported by Oracle Database Lite is a subset of the SQL language supported by Oracle. Oracle Database Lite supports some additional SQL-92 database objects, functions, and commands.

1.2.1 Objects

The differences between database objects supported by Oracle Database Lite and those supported by Oracle are listed in Table 1–2. See "Oracle Database Lite Database Object Naming Conventions" for more information:

Table 1–2 Differences Between Oracle Database Lite and Oracle-Supported Database Objects

Supported by Oracle Database Lite	Supported by Oracle
Tables, views, indexes, sequences, schemas, snapshots.	All database objects.
A name identifier up to 128 characters for columns, indexes, tables, and schemas. User name identifiers can be up to 30 characters.	A name identifier up to 31 characters.

1.2.2 Operators

Chapter 2, "SQL Operators", lists the operators supported by Oracle Database Lite. In general, the Oracle Database Lite supports all operators supported by Oracle.

Except for datatype-related differences, the corresponding operators always work identically.

1.2.3 Functions

Chapter 3, "SQL Functions" lists the functions supported by Oracle Database Lite. The functions listed in Table 1–3 produce different results in Oracle and Oracle Database Lite.

Table 1–3 Function Behavior in Oracle Database Lite and Oracle

Function	Supported by Oracle Lite	Supported by Oracle
`ROWID`	16 characters long	18 characters long
`TO_CHAR`	does not accept 'nlsparams'	accepts 'nlsparams'
`TO_DATE`	does not accept 'nlsparams'	accepts 'nlsparams'
`TO_NUMBER`	does not accept 'nlsparams'	accepts 'nlsparams'

1.2.4 Commands

Some Oracle commands have a more limited functionality in Oracle Database Lite. The Oracle command parameters that are not supported by Oracle Database Lite are listed in Table 1–4.

Table 1–4 Oracle Command Parameters Not Supported by Oracle Database Lite

Command	Element Unsupported by Oracle Lite
`CREATE TABLE`	Index clause for table and column constraints.
	Exceptions into clauses for table and column constraints.
	Physical organization clauses.
	Deferred options for columns and tables.

Table 1–4 (Cont.) Oracle Command Parameters Not Supported by Oracle Database Lite

Command	Element Unsupported by Oracle Lite
CREATE TRIGGER	On Views
	OR REPLACE
	INSTEAD OF
	REFERENCING OLD
	REFERENCING NEW
	WHEN
	OR
ALTER TABLE	RENAME
ALTER INDEX	Rename index option.
	Rebuild index option.
SET TRANSACTION	READ ONLY
	READ WRITE
UPDATE	Set clause containing subqueries that select more than one column.
	Returning clause where row IDs for updated rows are returned.
TO_CHAR	When used to extract timestamp from date value.

Note: There may be differences in subqueries for Oracle and Oracle Database Lite.

Oracle Database Lite does not support the following commands and clauses.

- Commands related to the following database objects.
  - Clusters
  - Database links
  - Stored functions and procedures other than Java stored procedures
  - Packages
  - Profiles
  - Rollback segments
  - Snapshot logs
  - Table spaces
- Physical data storage clauses such as PCTFREE.

1.2.5 Miscellaneous Data Definition Language (DDL)

Oracle Database Lite does not support space management, table spaces, and INITRANS.

Oracle Database Lite DDL does not commit when executed as Oracle does, but commits as part of the current transaction.

1.2.6 Datatypes

Oracle Database Lite supports more datatypes than Oracle. For results similar to those of Oracle in Oracle Database Lite, use `NUMBER` and specify precision and scale.

Oracle anticipates datatypes to return and their display. It may produce results automatically, where Oracle Database Lite may need a specific `CAST` (*one_datatype* AS *another_datatype*) in the statement. You should avoid `INT`, `FLOAT`, and `DOUBLE` if you want portability between machine types. Oracle Database Lite uses the native implementations of these datatypes while Oracle maps these to specific `NUMBER` datatypes.

1.2.7 Indicator Variables

Oracle Database Lite uses 32-bit `LONG` indicator variables integers. Oracle uses, 16-bit `SHORT` indicator variables integers.

1.2.8 Data Precision During Arithmetic Operations

Oracle databases look at the datatype on the left side of an assignment when deciding how many decimal places of a result to store into a column. Oracle Database Lite follows SQL-92 convention, and only provides the maximum number of digits of precision from the right side of the assignment.

1.2.9 Data Dictionaries

The Oracle Database Lite data dictionary is different from the Oracle data dictionary. Oracle Database Lite provides many commonly used system views including `ALL_TABLES` and `ALL_INDEXES`.

1.2.10 Tables Not Installed with Oracle Database Lite

The table `system.product_privs`, which contains product user profiles in an Oracle database, does not exist in the Oracle Database Lite.

1.2.11 Messages

Oracle Database Lite may not generate the same messages that Oracle databases generate in response to SQL commands. The error codes may also be different. Applications should not depend on a specific error code or message text to recognize that an error has occurred.

1.2.12 Sequences

Oracle Database Lite does not support `CYCLE` and `CACHE` clauses in sequence statements. Sequence numbers are also subject to `ROLLBACK` under some circumstances.

1.2.13 PL/SQL

Oracle Database Lite does not support PL/SQL. However, Oracle Database Lite does support stored procedures and triggers written in Java.

1.2.14 SQL Functions

Oracle Database Lite does not support trigonometric functions, SOUNDEX, or bit operations.

1.2.15 Locking and Transactions

Oracle Database Lite begins a transaction with the first use of SELECT. In some isolation levels, the use of a SELECT on one connection can lock out an UPDATE of the same table on another connection. You may need to COMMIT after a SELECT to free the lock, so the UPDATE may proceed.

1.3 Oracle Database Lite SQL Conventions

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. Oracle Database Lite SQL 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;
```

Reserved words, keywords, identifiers and parameters are not case-sensitive. However, text literals and quoted names are case-sensitive. See the syntax descriptions in Chapter 3, "SQL Functions" and Chapter 4, "SQL Commands".

1.3.1 SQL Statement Syntax

SQL syntax definitions use the following conventions. SQL syntax definitions are always shown in monospace text.

1.3.1.1 Capital Letters

```
SELECT
```

Indicates literal text that must be entered as shown.

1.3.1.2 Lowercase

```
table_name
```

Indicates a place holder that should be replaced by an appropriate value or expression. Any additional delimiter that the replacement value or expression requires such as single quotes is shown.

1.3.1.3 Bracket Delimited

```
[PUBLIC] OR [MAXVALUE | NOMAXVALUE]
```

Indicates an optional item or clause. Multiple items or clauses are separated by vertical bars. Do not enter brackets or vertical bars.

1.3.1.4 Braces

```
{ENABLE | DISABLE | COMPILE}
```

Braces enclose two or more required alternative choices, separated by vertical bars. Do not enter braces or vertical bars.

1.3.1.5 Vertical Bars

```
{IDENTITY | NULL} OR  [MAXVALUE integer | NOMAXVALUE]
```

Vertical bars separate two or more choices, either required arguments enclosed in braces { } or optional arguments enclosed in brackets [ ]. Do not enter vertical bars, braces, or brackets.

1.3.1.6 Ellipsis

```
[, column] ...
```

Indicates that further repetitions of the argument expressed in the same format are permissible. Do not enter ellipses.

1.3.1.7 Underline

```
[ASC | DESC]
```

Indicates the default value used if you do not specify any of the options separated by vertical bars.

1.3.1.8 Block Letters

```
PCTFREE
```

Indicates a keyword that should be entered exactly as shown.

1.3.1.9 Initial Colon

```
: integer_value
```

Indicates a place holder that should be replaced by an appropriate reference to a host variable. You include the initial colon with the host variable reference.

1.3.2 SQL Tables

A database can be made up of one or more database files or *catalogs* in ODBC and SQL-92. The fundamental unit of storage in SQL is a table consisting of rows of data organized in columns. All database objects, including tables, views, and indexes, are owned by a user name or a schema. By default in Oracle Database Lite, tables are created as part of the user schema, the schema with the same name as the login ID.

1.3.3 SQL Object Names

Object names in SQL must begin with a letter and may contain numbers and the special characters "_" and "$". Names are generally not case-sensitive. Mixed case names are permitted when enclosed in double quotes (" ").

Object names may be qualified by the catalog and schema to which they belong by separating the qualifiers with a period ".". For example,

```
production.payroll.emp.salary
```

This example refers to the `salary` column of the `emp` table owned by the `payroll` schema in the `production` catalog.

1.3.4 SQL Operator Precedence

The following list describes the relative precedence of SQL operators. The operators at the top of the list have the highest precedence (they are evaluated first); the operators at the bottom of the list have the lowest precedence (they are evaluated last). Operators of equal precedence are evaluated from left to right.

1. `+ (unary), -(unary), PRIOR`
2. `*,/`
3. `+, -, ||`
4. All comparison operators
5. `NOT`
6. `AND`
7. `OR`

You can use parentheses in an expression to override operator precedence. Expressions inside parentheses are evaluated before those outside parentheses.

1.3.5 SQL Sessions

The execution of SQL statements requires the existence of a SQL session. An application can establish a SQL session by performing the following.

- Issuing a SQL statement that requires a SQL session (a default session is implicitly established).
- Issuing `SQLConnect` or `SQLDriverConnect` ODBC calls.

A SQL session is closed when one of the following occurs.

- The `SQLDisconnect` API in ODBC is called.
- An ODBC program terminates.

1.3.6 SQL Transactions

SQL databases handle requests in logical units of work called transactions. A transaction is a group of related operations that must be performed successfully before any changes to the database are finalized.

A SQL transaction starts when any DDL or DML statement is executed in a session. When you are satisfied that no errors occurred during the transaction, you can end the transaction with a `COMMIT` command. The database then changes to reflect the operation. If an error occurs, you can abandon the changes with the `ROLLBACK` command.

Oracle Database Lite does not commit a DDL statement until you issue the `COMMIT` command. Oracle immediately commits all DDL statements.

1.3.7 Issuing SQL Statements From a Program

Oracle Database Lite datatypes and object classes are interoperable with other programming languages. You can issue SQL statements to Oracle Database Lite in a host language if you connect to the database from within the application, using the appropriate ODBC or JDBC driver.

1.3.8 SQL and ODBC

The Open Database Connectivity (ODBC) interface from Microsoft defines a call level interface to provide interoperability across different databases. ODBC specifies a set of interface functions to allow the following features.

- Connections to databases by different vendors.
- Preparation and execution of SQL statements in a common language.
- Retrieval of query results into local program variables.

Oracle Database Lite supports the ODBC 2.0 call level interface (CLI). Oracle Database Lite SQL supports implicit type conversion from the character string type to another datatype when necessary. For example, if the datatype of a column `AGE` is `INTEGER`, and you execute the following statement.

```
UPDATE EMPLOYEE SET AGE = '30' WHERE NAME = 'John'
```

'30' is automatically converted to an `INTEGER` type.

1.4 ODBC SQL Syntax Conventions

There are two principal reasons to use ODBC SQL syntax rather than the SQL syntax that is specific to your database.

First, SQL statements written in ODBC syntax are easily transferred among ODBC-compliant databases. Even though ODBC SQL syntax does not include many of the keywords and arguments that invoke important functionality for a specific database, SQL statements written in ODBC syntax are fully portable among all ODBC-compliant databases.

Second, you can use ODBC SQL syntax to execute SQL statements against databases that you are not familiar with. While ODBC SQL syntax cannot invoke your database's full functionality like your database's own SQL syntax, you can use it to perform many of the most common, and important, database functions.

You can always use database-specific SQL syntax, even when connected to a database through ODBC, since ODBC passes SQL statements through to a connected database without modification.

1.5 Oracle Database Lite Database Object Naming Conventions

This section lists rules for naming Oracle Database Lite database objects and their parts.

1. User names must be from 1 to 30 characters long. Columns, indexes, tables, and schemas can be up to 128 characters long. Oracle Database Lite has no limit on name length, but it is recommended that you limit your name length to 30 characters.
2. Names cannot contain quotation marks.
3. Names are not case sensitive.
4. A name must begin with an alphabetic character.
5. Names can contain only alphanumeric characters and the characters `_`, `$`, and `#`. The use of `$` and `#` is not recommended.
6. A name cannot be an Oracle Database Lite reserved word.
7. The word `DUAL` should not be used as a name for an object or part.

8. The Oracle Database Lite 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 to read. See Appendix A, "Oracle Database Lite Keywords and Reserved Words" for a list of Oracle Lite keywords.
9. A name must be unique across its name space.
10. A name can be enclosed in double quotes. Such names can contain any combination of characters, ignoring rules 3 through 7 in this list.
11. Names cannot contain a dot (".") character.

1.6 Formats

The sections Number Format Elements and Date Format Elements list the elements you can use to create a valid number or date format. Formats can be used as arguments to the SQL functions: `TO_DATE`, `TO_NUMBER`, `TO_CHAR`, and `TRUNC`.

1.6.1 Number Format Elements

Oracle Database Lite number formats are listed in Table 1–5.

Table 1–5 Oracle Database Lite Number Formats

Element	Example	Description
9	9999	The number of nines specifies the number of significant digits returned. Blanks are returned for leading zeros and for a value of zero.
0	0000.00	Returns a leading zero or a value of zero as a 0, rather than as a blank.
$	$9999	Prefixes value with a dollar sign.
B	B9999	Returns zero value as blank, regardless of zeros in the format model.
MI	9999MI	Returns "-" after negative values. For positive values, a trailing space is returned.
S	S9999	Returns "+" for positive values and "-" for negative values.
PR	9999PR	Returns negative values in <angle brackets>. For positive values, a leading and trailing space are returned.
D	99D99	Returns the decimal character, separating the integral and fractional parts of a number.
G	9G999	Returns the group separator.
C	C999	Returns the ISO currency symbol.
L	L999	Returns the local currency symbol.
, (comma)	9,999	Returns a comma.
. (period)	99.99	Returns a period, separating the integral and fractional parts of a number.
EEEE	9.999EEEE	Returns a value in scientific notation.

1.6.2 Date Format Elements

Oracle Database Lite date formats are listed in Table 1–6.

Table 1–6 Oracle Database Lite Date Formats

Element	Description
SCC or CC	Century; "S" prefixes BC dates with "-".
YYYY or SYYYY	4-digit year; "S" prefixes BC dates with "-".
IYYY	4-digit year based on the ISO standard.
YYY or YY or Y	Last 3, 2, or 1 digit(s) of year.
IYY or IY or I	Last 3, 2, or 1 digit(s) of the ISO year.
Y,YYY	Year with comma.
Q	Quarter of year (1, 2, 3, 4; JAN-MAR = 1)
MM	Month (01-12; JAN = 01)
MONTH	Name of month; padded with blanks to length of 9 characters.
MON	Abbreviated name of the month.
WW	Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.
IW	Week of year (1-52 or 1-53) based on the ISO standard.
W	Week of month (1-5) where week 1 starts on the first day of the year and continues to the seventh day of the year.
DDD	Day of year (1-366).
DD	Day of month (1-31).
D	Day of week (1-7).
DAY	Name of day, padded with blanks to length of 9 characters.
DY	Abbreviated name of day.
AM or PM	Meridian indicator.
A.M. or P.M.	Meridian indicator with periods.
HH or HH12	Hour of day (1-12).
HH24	Hour of day (0-23).
MI	Minute (0-59).
RR	Last 2 digits of year; for years in other countries.
SS	Second (0-59).
SSSSS	Seconds past midnight (0-86399).
- / . ; : "text"	Punctuation and quoted text is reproduced in the result.

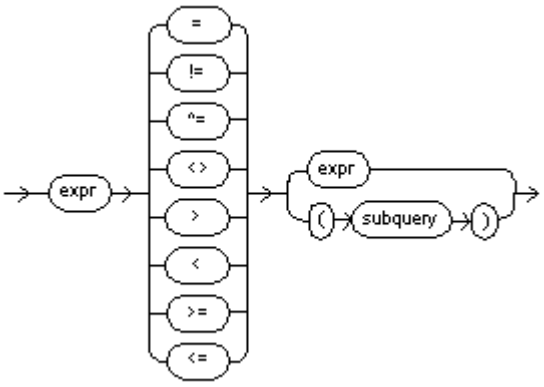
1.7 Specifying SQL Conditions

Use one of the following syntax forms to specify a SQL condition. The syntax diagrams in this document use a variation of Backus-Nauer Form (BNF) notation. For a description of the convention used in this document, please see Section 4.2.6, "BNF Notation Conventions".

1.7.1 Simple Comparison Conditions

A simple comparison condition specifies a comparison with expressions or subquery results using the syntax displayed in Figure 1–1.

Figure 1–1 A SIMPLE COMPARISON Condition



BNF Notation

```
{ expr { = | != | ^= | <> | > | < | >= | <= } { expr |"(" subquery")"}
```

For example,

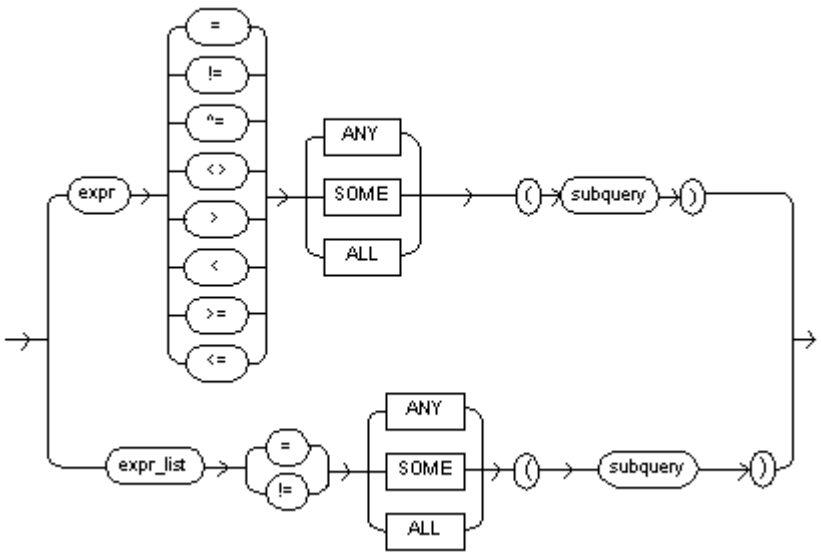
```
SELECT * FROM EMP WHERE SAL > 2000;
```

For information on comparison operators, see Comparison Operators.

1.7.2 Group Comparison Conditions

A group comparison condition specifies a comparison with any or all members in a list or subquery using the syntax displayed in Figure 1–2.

Figure 1–2 A GROUP COMPARISON Condition



BNF Notation

```
{ expr
   { = | != | ^= | <> | > | < | >= | <= }
```

```
   { ANY | SOME | ALL }
   {"(" subquery")"}
| expr_list
   { = | != }
   { ANY | SOME | ALL }
   { "(" subquery ")"}
}
```

For example:

```
SELECT * FROM EMP WHERE ENAME = any ('SMITH', 'WARD', 'KING');
```

1.7.2.1 A Row_Value_Constructor in a Subquery Comparison

This allows the comparison of columns or expressions using a subquery that returns a multi-column result. This feature allows users to supply a row value constructor, such as a list of comma-separated expressions enclosed within parenthesis.

1.7.2.2 Subquery in Place of a Column

You may insert a subquery anywhere. An arithmetic expression or a column can appear. The subquery needs to be enclosed in parenthesis and is restricted to return a maximum of one row with one column.

For example,

1. Subquery in a select list. The following query is supported (assuming c1 and c2 are columns in table t1 and c1 is a primary key).

   ```
   SELECT (select c1 from t1 b where a.c1 = b.c1),
       c2 from t1 a where <condition>
   ```

   The select list of the subquery in a select list can itself contain a subquery. There is no limit to the number of nested subqueries.

2. Subquery in an expression: The following query is supported (with the same assumption as example 1).

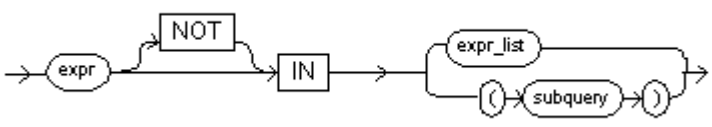
   ```
   SELECT *  from t1 a  where
      (select c1 from t1 where c1 = 10) =
      (select c1 from t1 b where a.c1 = b.c1) - 20;
   ```

3. A subquery can contain Group By, Union, Minus, and Intersect, but not an Order By clause.

1.7.3 Membership Conditions

A membership condition tests for membership in a list or subquery using the syntax displayed in Figure 1–3.

Figure 1–3 A MEMBERSHIP Condition



BNF Notation

```
expr [NOT] IN { expr_list | "("subquery ")"}
```

For example,

```
SELECT * FROM EMP WHERE ENAME not in ('SMITH', 'WARD', 'KING');
```

1.7.4 Range Conditions

A range condition tests for inclusion in a range using the syntax displayed in Figure 1–4.

Figure 1–4 A RANGE Condition

BNF Notation

```
expr [ NOT ] BETWEEN expr AND expr ;
```

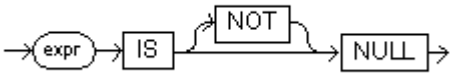
For example,

```
SELECT * FROM EMP WHERE SAL between 2000 and 50000;
```

1.7.5 NULL Conditions

A NULL condition tests for nulls using the syntax displayed in Figure 1–5.

Figure 1–5 A NULL Condition



BNF Notation

```
expr IS [NOT]  NULL
```

For example:

```
SELECT * FROM EMP WHERE MGR IS NOT NULL;
```

1.7.6 EXISTS Conditions

An EXISTS condition tests for the existence of rows in a subquery using the syntax displayed in Figure 1–6.

Figure 1–6 An EXISTS Condition

BNF Notation

```
EXISTS "("subquery")"
```

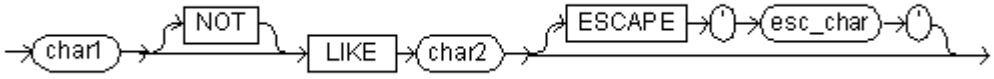
For example,

```
SELECT * FROM EMP WHERE EXISTS (SELECT ENAME FROM EMP WHERE MGR IS NULL);
```

1.7.7 LIKE Conditions

A LIKE condition specifies a test involving pattern matching using the syntax displayed in Figure 1–7.

Figure 1–7 Like Conditions Syntax



BNF Notation

```
char1 [NOT] LIKE  char2 [ESCAPE "'"esc_char"'" ]
```

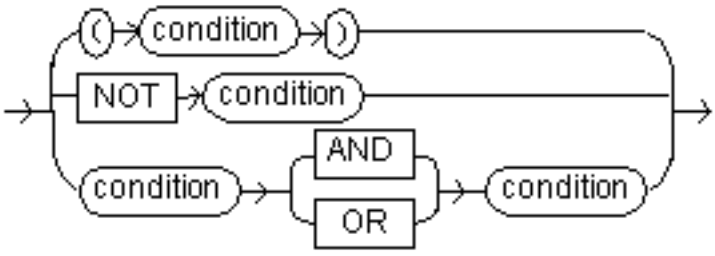
For example,

```
SELECT * FROM EMP WHERE NAME like 'SM%"
```

1.7.8 Compound Conditions

A COMPOUND condition specifies a combination of other conditions using the syntax displayed in Figure 1–8.

Figure 1–8 A COMPOUND Condition



BNF Notation

```
{ "(" condition ")"
  | NOT  condition
  | condition {AND | OR} condition
```

```
}
;
```

For example,

```
SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500;
```

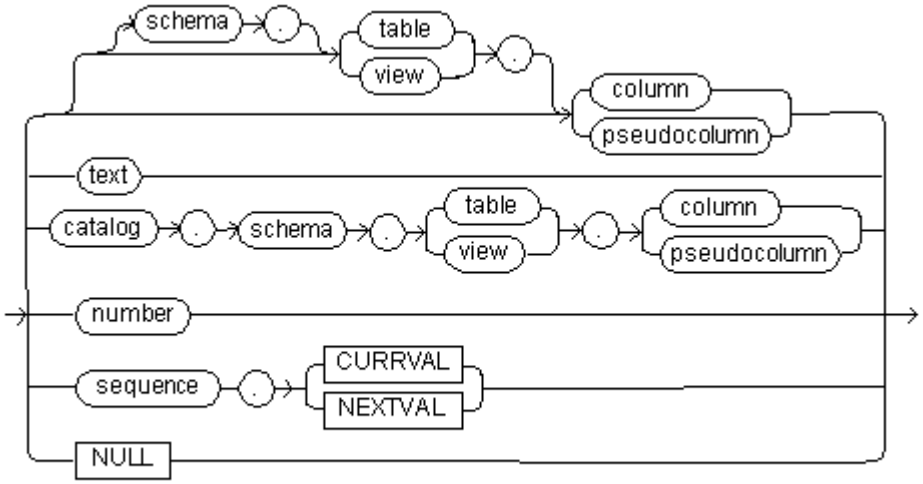
1.8 Specifying Expressions

Use one of the following syntax forms to specify a SQL expression.

1.8.1 Form I, Simple Expression

A simple expression specifies column, pseudocolumn, constant, sequence number, or null using the syntax displayed in Figure 1–9.

Figure 1–9 A SIMPLE Expression



BNF Notation

```
{ [schema .] { table | view } "." { column | pseudocolumn }
| text
| catalog "." schema "." { table| view } "." { column | pseudocolumn }
| number
| sequence "." { CURRVAL | NEXTVAL }
| NULL
}
```

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 materialized view. Qualifying a public synonym with `PUBLIC` is supported only in Data Manipulation Language (DML) statements, not Data Definition Language (DDL) statements.

The pseudocolumn can be either `LEVEL`, `ROWID`, or `ROWNUM`. You can use a pseudocolumn only with a table, not with a view or materialized view.

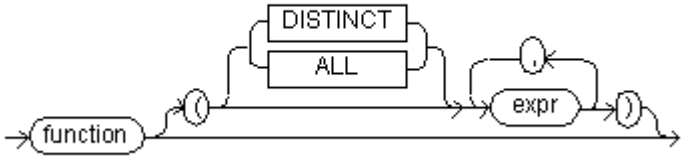
Examples

```
emp-ename
'this is a text string'
10
```

1.8.2 Form II, Function Expression

A built-in function expression specifies a call to a single-row SQL function using the syntax displayed in Figure 1–10.

Figure 1–10 A FUNCTION Expression



BNF Notation

```
function ["(" [DISTINCT | ALL] expr [, expr]...")"] ;
```

Some valid built-in function expressions are:

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

1.8.3 Form III, Java Function Expression

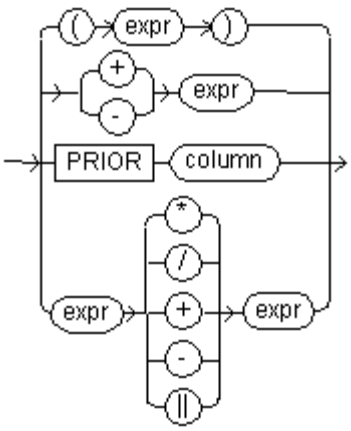
```
java_function_name (expr , expr...)
schema.table.java_function_name (expr , expr...)
```

For information on how to use Java functions, see the *Oracle Database Lite Developer's Guide for Java*.

1.8.4 Form IV, Compound Expression

A compound expression specifies a combination of other expressions using the syntax displayed in Figure 1–11.

Figure 1–11 A COMPOUND Expression



BNF Notation

```
 { "(" expr ")"
  | { + | - } expr
  | PRIOR column
  | expr( * | / | + | - | ||) expr
 }
;
```

Some combinations of functions are inappropriate and are rejected. For example, the `LENGTH` function is inappropriate within an aggregate function.

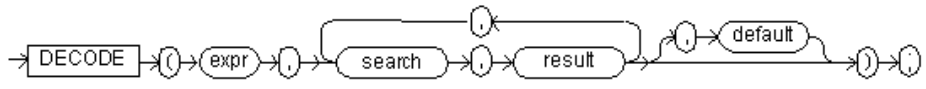
Examples

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

1.8.5 Form V, DECODE Expression

A `DECODE` expression uses the special `DECODE` syntax displayed in Figure 1–12.

Figure 1–12 The DECODE Expression



BNF Notation

```
DECODE "(" expr "," search "," result [, search "," result]... [, default] ")" ;
```

To evaluate this expression, Oracle Database Lite compares *expr* to each *search* value one by one. If *expr* is equal to a *searc*h, Oracle Database Lite returns the corresponding *result*. If no match is found, Oracle Database Lite returns *default*, or, if *default* is omitted, returns null. If *expr* and *search* contain character data, Oracle Database Lite compares them using non-padded comparison semantics.

The *search*, *resul*t, and *default* values can be derived from expressions. Oracle Database Lite evaluates each *search* value only before comparing it to *expr*, rather than evaluating

all *search* values before comparing any of them with *expr*. Consequently, Oracle Database Lite never evaluates a *search* if a previous *search* is equal to *expr*.

Oracle Database Lite automatically converts *expr* and each *search* value to the datatype of the first *search* value before comparing. Oracle Database Lite automatically converts the return value to the same datatype as the first *result*. If the first *result* has the datatype CHAR or if the first *result* is null, then Oracle Database Lite converts the return value to the datatype VARCHAR2.

In a DECODE expression, Oracle Database Lite considers two nulls to be equivalent. If *expr* is null, Oracle Database Lite returns the *result* of the first *search* that is also null. The maximum number of components in the DECODE expression, including *expr*, searches, results, and default is 255.

Example

This expression decodes the value DEPTNO. In this example, if DEPTNO is 10, the expression evaluates to 'ACCOUNTING'. If DEPTNO is not 10, 20, 30, or 40, the expression returns 'NONE'.

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

1.8.6 Form VI, Expression List

An EXPRESSION LIST is a series of expressions, each separated by a comma as displayed in Figure 1–13. The entire series is enclosed in parenthesis.

Figure 1–13 The EXPRESSION List

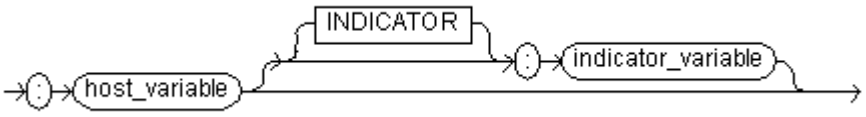
BNF Notation

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

1.8.7 Form VII, Variable Expression

A VARIABLE EXPRESSION specifies a host variable with an optional indicator variable as displayed in Figure 1–14. This form of expression can appear in a programmatic programming interface.

Figure 1–14 The VARIABLE Expression



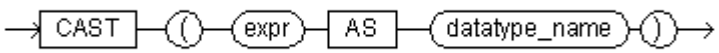
BNF Notation

```
":" host_variable [[INDICATOR] ":" indicator_variable]
```

1.8.8 Form VIII, CAST Expression

A `CAST` expression converts one built-in datatype or collection-typed value into another built-in datatype or collection-typed value as displayed in Figure 1–15.

Figure 1–15 The CAST Expression



BNF Notation

```
CAST "(" expr AS datatype_name ")"
```

For the operand, *expr* is a built-in datatype. Table 1–7 shows which built-in datatypes accept `CAST` conversion to another datatype. (`CAST` does not support `LONG`, `LONG RAW`, or any of the `LOB` datatypes.)

Table 1–7 Built-In Datatypes that Accept the CAST Conversion

From/ To	Char, Varchar2	Numeric	Date	Time	Timestamp	Raw
Char, Varchar2	X	X	X	X	X	X
Numeric	X	X				
Date	X		X		X	
Time	X			X	X	
Timestamp	X		X	X	X	
Raw	X					X

The `Date` datatype is affected by the `SQLCompatibility` setting defined in the **POLITE.INI** file.

- `Date` and `Timestamp` are equivalent if you have set: `SQLCompatibility=Oracle`
- `Date` and `Timestamp` are not equivalent if you have set: `SQLCompatibility=SQL92`

See the *Oracle Database Lite Administration and Deployment Guide* for more information about the **POLITE.INI** file.

The numeric category includes the following datatypes: `BIGINT`, `BINARY`, `BIT`, `DECIMAL`, `DOUBLE PRECISION`, `FLOAT`, `INTEGER`, `NUMBER`, `NUMERIC`, `REAL`, `SMALLINT`, and `TINYINT`.

Built-In Datatype Examples

```
SELECT CAST ('1997-10-22' AS DATE) FROM DUAL;
SELECT * FROM t1 WHERE CAST (ROWID AS CHAR(5)) = '01234';
```

1.9 Oracle Database Lite SQL Datatypes and Literals

For a complete list of Oracle Database Lite SQL datatypes, see Appendix C, "Oracle Database Lite Datatypes". For information about literals, see Appendix D, "Oracle Database Lite Literals".

1.9.1 Character String Comparison Rules

Oracle Database Lite compares character string values 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 vary. Table 1–8 lists the results of comparing five pairs of character values using each comparison semantic. Generally, the results of blank-padded and non-padded comparisons are the same. The last comparison in the table illustrates the differences between the blank-padded and non-padded comparison semantics.

Table 1–8 Comparison of Blank-Padded and Non-Padded Comparison Semantics

Blank-Padded	Non-Padded
`'ab' > 'aa'`	`'ab' > 'aa'`
`'ab' > 'a '`	`'ab' > 'a  '`
`'ab' > 'a'`	`'ab' > 'a'`
`'ab' = 'ab'`	`'ab' = 'ab'`
`'a  ' = 'a'`	`'a  ' > 'a'`

1.9.1.1 Blank-Padded Comparison Semantics

If the two values have different lengths, Oracle Database Lite first adds blanks to the end of the shorter one so that their lengths are equal. Oracle Database Lite then compares the values character by character up to the first character that differs. The value with the greater than character (>) in the first differing position is considered greater. If two values have no differing characters, then they are considered equal. This rule means that two values are equal if they differ only in the number of trailing blanks. Oracle Database Lite uses blank-padded comparison semantics only when both values in the comparison are either expressions of the datatype `CHAR`, text literals, or values returned by the `USER` and `DATABASE` functions.

1.9.1.2 Non-Padded Comparison Semantics

Oracle Database Lite compares two values character by character up to the first character that differs. The value with the greater than character (>) in that position is considered greater. If two values of different length are identical up to the end of the shorter one, the longer value is considered greater. If two values of equal length have no differing characters, then the values are considered equal. Oracle Database Lite uses non-padded comparison semantics whenever one or both values in the comparison have the datatype `VARCHAR2`. As a result, when comparing a `CHAR` value with a `VARCHAR2` value, Oracle Database Lite considers the character value 'a ' unequal to 'a'.

1.10 Comments Within SQL Statements

You can associate comments with SQL statements and schema objects. Comments within SQL statements do not affect the statement execution, but they can make your application easier to read and maintain.

A comment can appear between any keywords, parameters, or punctuation marks in a statement. You can include a comment in a statement using one of the following options.

- Begin the comment with a slash and an asterisk (/*). Proceed with the text of the comment. This text can span multiple lines. End the comment with an asterisk and a slash (*/). 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.

Example 1

```
SELECT * FROM EMP WHERE EMP.DEPTNO = /* The subquery matches values in EMP.DEPTNO
with values in DEPT.DEPTNO */ (SELECT DEPTNO FROM DEPT WHERE LOC='DALLAS');
```

This statement returns the following output.

```
    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
     7566 JONES      MANAGER        7839 1981-04-0      2975                    20
     7902 FORD       ANALYST        7566 1981-12-0      3000                    20
     7369 SMITH      CLERK          7902 1980-12-1       800                    20
     7788 SCOTT      ANALYST        7566 1982-12-0      3000                    20
     7876 ADAMS      CLERK          7788 1983-01-1      1100                    20
```

Example 2

```
SELECT ENAME, -- select the employee name
 SAL          -- and the salary
 FROM EMP     -- from the EMP table
 WHERE SAL    -- where the salary
 >=           -- is greater than or equal to
 3000         -- 3000
 ;
```

This statement returns the following output:

```
ENAME            SAL
---------- ---------
KING            5000
FORD            3000
SCOTT           3000
```

1.11 Tuning SQL Statement Execution Performance With the EXPLAIN PLAN

To execute a SQL statement, Oracle might need to perform several operations. The combination of the operations Oracle uses to execute a statement is called an execution plan, which includes an access path for each table that the statement accesses and an ordering of the tables (the join order) with the appropriate join method. The execution plan shows you exactly how Oracle Database Lite executes your SQL statement.

The components of an execution plan include the following:

- An ordering of the tables referenced by the statement.
- An access method for each table mentioned in the statement.

- A join method for tables affected by join operations in the statement.
- Data operations, such as FILTER, SORT, UNION, and so on.

The EXPLAIN PLAN command stores the execution plan chosen by the Oracle Database Lite optimizer for SELECT, UPDATE, INSERT, and DELETE statements into the table—PLAN_TABLE. Before using the EXPLAIN PLAN statement, a user creates the plan table using an interactive query tool, such as `msql`.

You can examine the execution plan chosen by the optimizer for a SQL statement by using the EXPLAIN PLAN statement. When the statement is issued, the optimizer chooses an execution plan and then inserts data describing the plan into a database table. Simply issue the EXPLAIN PLAN statement and then query the output table.

The EXPLAIN PLAN output shows how Oracle executes SQL statements, which helps a developer or DBA understand how a query is being executed. Thus, you can identify additional indexes needed, or how best to modify the query. The query modification may involve a re-write or use of optimizer hints to change the join order.

1. Use the SQL script—`utlxplan.sql`—to create the sample output table called `PLAN_TABLE` in your schema. Alternatively, you can use `msql` to create the plan table. See Section 1.11.1, "The PLAN Table".
2. Include the EXPLAIN PLAN `FOR` clause prior to the SQL statement. The syntax is as follows:

   ```
   Explain_plan_statement ::= EXPLAIN PLAN [SET STATEMENT_ID = 'text' ] [INTO
   [schema.] plan_table] [FOR] statement;
   ```

   Where

   - `statement` is any SELECT, UPDATE, INSERT, DELETE statement
   - '`text`' is a literal provided by the user to identify all rows for the given query.
   - `schema.plan_table` is the table where you want the result to be stored. The table must conform to the layout given in the `utlxplan.sql` script. The default value for is `PLAN_TABLE` in your own schema.
3. After issuing the EXPLAIN PLAN statement, query the `PLAN_TABLE` for the output.

The EXPLAIN PLAN command is not unique to Oracle Database Lite. It is a feature of the Oracle database. However, not all SQL operations supported in the Oracle database are supported by Oracle Database Lite. This section shows the operation subset that you can use in Oracle Database Lite.

In addition, this section does not go into full details on how the EXPLAIN PLAN works. For a full description of the EXPLAIN PLAN, see the "Using Explain Plan" chapter in the *Oracle Database Performance Tuning Guide*.

- Section 1.11.1, "The PLAN Table"
- Section 1.11.2, "EXPLAIN PLAN Examples"

1.11.1 The PLAN Table

The `PLAN_TABLE` is the default sample output table into which the `EXPLAIN PLAN` statement inserts rows describing execution plans. See Table 1–9 for a description of the columns in the table.

Use the SQL script `utlxplan.sql` to manually create a local `PLAN_TABLE` in your schema.

Table 1–9 Plan Table

Column	Data Type	Description
statement_id	Varchar2(30)	User specified ID
Timestamp	Date	Date and time of creation
Remarks	Varchar2(80)	A user-specified remarks
Operation	Varchar2(30)	The name of operation, such as SELECT, INSERT, UPDATE, DELETE, TABLE ACCESS and so on. See the Operations table for more information.
Options	Varchar2(30)	Qualification for the operation
object_owner	Varchar2(30)	Owner of a table or index
object_name	Integer	Name of the table or index
Id	Integer	Step identification number
parent_id	Integer	Parent step number
Position	Integer	Order of processing among the steps that have the same parent step id
Cost	Integer	Estimated cost in number of I/Os.
Cardinality	Integer	The estimated number of rows produced
Text	Varchar2(4096)	First 4096 bytes of the statement text stored with the first step of execution. For example, id=0

Table 1–10 lists each combination of Operation and Option produced by the EXPLAIN PLAN statement and its meaning within an execution plan.

Table 1–10 Operation and Option Values Produced by the EXPLAIN PLAN

Operation	Options	Comments
CONNECT BY		Retrieves rows in hierarchical order for a query containing a CONNECT BY clause.
FILTER	None	Operation accepting a set of rows, eliminates some of them, and returns the rest.
FOR UPDATE	None	Operation retrieving and locking the rows selected by a query containing a FOR UPDATE clause.
INDEX		Retrieval of one or more rowids from an index.
NESTED LOOP		Operation accepting two sets of rows, an outer set and an inner set. Oracle compares each row of the outer set with each row of the inner set, returning rows that satisfy a condition.
SORT	AGGREGATE, UNIQUE, GROUP BY, ORDER BY	A sort is being performed for aggregation, duplicate removal, group by or order by operations respectively.
TABLE ACCESS	FULL	All data pages of the table will be scanned.
TABLE ACCESS	BY INDEX ROWID, BY ROWID	The table rows are accessed using rowids from an index, or provided by some other means.

Table 1–10 (Cont.) Operation and Option Values Produced by the EXPLAIN PLAN

Operation	Options	Comments
UNION ALL		A UNION ALL operation is being performed.
VIEW		A logical or physical view is being materialized.
CREATE TEMP TABLE	ORDER BY, READ COMMITTED, GROUP BY, UNION, CONNECT BY, MINUS, AGGREGATE	Option indicates the reason for creating the temporary table.
INSERT		An INSERT operation is being performed.
UPDATE		An UPDATE operation is being performed.
DELETE		A DELETE operation is being performed.
SELECT		A SELECT operation is being performed.
MINUS		A MINUS operation is being performed.

1.11.2 EXPLAIN PLAN Examples

The following examples demonstrate the EXPLAIN PLAN statement. The output for each example should only be used as a guideline. The actual output is subject to change based on the analysis of internal data structures. The examples are based on a sample schema, as follows:

Sample Schema

```
drop table s;
drop table sp;
drop table p;
drop table j;
drop table spj;
create table S ( S# char(3), SNAME Char(10), Status Int, City char(10));
create table P (P# char(3), PNAME Char(10), Color Char(10), Weight Int, City
Char(10));
create table SP (S# char(3), P# Char(3), Qty Int);create table J ( J# char(3),
JNAME Char(10), City char(10));create table SPJ (S# char(3), P# Char(3), J#
Char(3), Qty Int);
insert into S values ('S1', 'Smith', 20, 'London');insert into S values ('S2',
'Jones', 10, 'Paris');
insert into S values ('S3', 'Blake', 30, 'Paris');insert into S values ('S4',
'Clark', 20, 'London');
insert into S values ('S5', 'Adams', 30, 'Athens');insert into P values ('P1',
'Nut', 'Red', 12, 'London');
insert into P values ('P2', 'Bolt', 'Green', 17, 'Paris');
insert into P values ('P3', 'Screw', 'Blue', 17, 'Rome');
insert into P values ('P4', 'Screw', 'Red', 14, 'London');
insert into P values ('P5', 'Cam', 'Blue', 12, 'Paris');
insert into P values ('P6', 'Cog', 'Red', 19, 'London');
insert into J values ('J1', 'Sorter', 'Paris');insert into J values ('J2',
'Punch', 'Rome');insert into J values ('J3', 'Reader', 'Athens');
insert into J values ('J4', 'Console', 'Athens');
insert into J values ('J5', 'Collator', 'London');
insert into J values ('J6', 'Terminal', 'Oslo');
insert into J values ('J7', 'Tape', 'London');
insert into SP values ('S1', 'P1', 300);
insert into SP values ('S1', 'P2', 200);
```

```
insert into SP values ('S1', 'P3', 400);
insert into SP values ('S1', 'P4', 200);
insert into SP values ('S1', 'P5', 100);
insert into SP values ('S1', 'P6', 100);
insert into SP values ('S2', 'P1', 300);
insert into SP values ('S2', 'P2', 400);
insert into SP values ('S3', 'P2', 200);
insert into SP values ('S4', 'P2', 200);
insert into SP values ('S4', 'P4', 300);
insert into SP values ('S4', 'P5', 400);
insert into SPJ values ('S1', 'P1', 'J1', 200);
insert into SPJ values ('S1', 'P1', 'J4', 700);
insert into SPJ values ('S2', 'P3', 'J1', 400);
insert into SPJ values ('S2', 'P3', 'J2', 200);
insert into SPJ values ('S2', 'P3', 'J3', 200);
insert into SPJ values ('S2', 'P3', 'J4', 500);
insert into SPJ values ('S2', 'P3', 'J5', 600);
insert into SPJ values ('S2', 'P3', 'J6', 400);
insert into SPJ values ('S2', 'P3', 'J7', 800);
insert into SPJ values ('S2', 'P5', 'J5', 100);
insert into SPJ values ('S3', 'P3', 'J1', 200);
insert into SPJ values ('S3', 'P4', 'J2', 500);
insert into SPJ values ('S4', 'P6', 'J3', 300);
insert into SPJ values ('S5', 'P2', 'J2', 200);
commit;
create unique index SIX1 on S ( S# );
create unique index PIX1 on P ( P# );
create unique index SPIX1 on SP ( S#, P# );
create unique index SPJIX1 on SPJ ( S#, P#, J# );
create index PCOLOR on P(Color);
```

The following examples demonstrate three examples of the output for the EXPLAIN PLAN for specific select statements:

- Section 1.11.2.1, "Example for Select Distinct and Group By"
- Section 1.11.2.2, "Example for Select Statement with Union"
- Section 1.11.2.3, "Example for Select Statement With Multiple Qualifiers"

1.11.2.1 Example for Select Distinct and Group By

The following is an example query and corresponding output from the EXPLAIN PLAN for a select statement where `select distinct P# from SPJ group by p#,j# having avg(qty) > 320;`

Table 1–11 Sample Output

ID	POSITION	PARENT_ID	OPERATION	OPTIONS	OBJNAME
0			SORT	ORDER BY	
1	1	0	CREATE TEMP TABLE	ORDER BY	
2	1	1	SELECT		
3	1	2	FILTER		
4	1	3	CREATE TEMP TABLE	GROUP BY	
5	1	4	TABLE ACCESS	FULL	SPJ

1.11.2.2 Example for Select Statement with Union

The following is an example query and corresponding output from the EXPLAIN PLAN for a select statement where `select s.* from S, SP where s.s#=Sp.P# and status > 20 and qty > 40 union Select s.* from s,sp, p where s.s#=sp.s# and sp.p#=p.p# and p.color='Red';`

Table 1–12

ID	POSITION	PARENT_ID	OPERATION	OPTIONS	OBJNAME
0			SORT	ORDER BY	
1	1	0	CREATE TEMP TABLE	ORDER BY	
2	1	1	UNION ALL		
3	1	2	SELECT		
4	1	3	FILTER		
5	1	4	NESTED LOOP		
6	1	5	TABLE ACCESS	FULL	S
7	2	5	TABLE ACCESS	FULL	SP
8	2	2	SELECT		
9	1	8	FILTER		
10	1	9	NESTED LOOP		
11	1	10	NESTED LOOP		
12	1	11	TABLE ACCESS	BY INDEX ROWID	P
13	1	12	INDEX		PCOLOR
14	2	11	TABLE ACCESS	FULL	SP
15	2	10	TABLE ACCESS	BY INDEX ROWID	S
16	1	15	INDEX		SIX1

1.11.2.3 Example for Select Statement With Multiple Qualifiers

The following is an example query and corresponding output from the EXPLAIN PLAN for a select statement where `select s.* from s, sp where s.s# = sp.s# and status > 20 and city in (select city from j where j# = 'J1' or j# = 'J2');`

Table 1–13

ID	POSITION	PARENT_ID	OPERATION	OPTIONS	OBJECT_NAME
0			SELECT		
1	1	0	FILTER		
2	1	1	NESTED LOOP		
3	1	2	TABLE ACCESS	FULL	S
4	2	2	TABLE ACCESS	BY INDEX ROWID	SP
5	1	4	INDEX		SPIX1
6	2	1	SELECT		
7	1	6	FILTER		

Table 1–13 (Cont.)

ID	POSITION	PARENT_ID	OPERATION	OPTIONS	OBJECT_NAME
8	1	7	TABLE ACCESS	FULL	J

2

SQL Operators

This document discusses SQL operators used with Oracle Database Lite. Topics include:

2.1 SQL Operators Overview

An operator manipulates individual data items and returns a result. The data 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`. There are two general classes of operators: unary and binary. Oracle Database Lite SQL also supports set operators.

2.1.1 Unary Operators

A unary operator uses only one operand. A unary operator typically appears with its operand in the following format.

```
operator operand
```

2.1.2 Binary Operators

A binary operator uses two operands. A binary operator appears with its operands in the following format.

```
operand1 operator operand2
```

2.1.3 Set Operators

Set operators combine sets of rows returned by queries, instead of individual data items. All set operators have equal precedence. Oracle Database Lite supports the following set operators.

- `UNION`

- UNION ALL
- INTERSECT
- MINUS

The levels of precedence among the Oracle Database Lite SQL operators from high to low are listed in Table 2–1. Operators listed on the same line have the same level of precedence.

Table 2–1 Levels of Precedence of the Oracle Database Lite SQL Operators

Precedence Level	SQL Operator
1	Unary + - arithmetic operators, PRIOR operator
2	* / arithmetic operators
3	Binary + - arithmetic operators, || character operators
4	All comparison operators
5	NOT logical operator
6	AND logical operator
7	OR logical operator

2.1.4 Other Operators

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

2.2 Arithmetic Operators

Arithmetic operators manipulate numeric operands. The '-' operator is also used in date arithmetic. Supported arithmetic operators are listed in Table 2–2.

Table 2–2 Arithmetic Operators

Operator	Description	Example
+ (unary)	Makes operand positive	`SELECT +3 FROM DUAL;`
- (unary)	Negates operand	`SELECT -4 FROM DUAL;`
/	Division (numbers and dates)	`SELECT SAL / 10 FROM EMP;`
*	Multiplication	`SELECT SAL * 5 FROM EMP;`
+	Addition (numbers and dates)	`SELECT SAL + 200 FROM EMP;`
-	Subtraction (numbers and dates)	`SELECT SAL - 100 FROM EMP;`

2.3 Character Operators

Character operators used in expressions to manipulate character strings are listed in Table 2–3.

Table 2–3 Character Operators

Operator	Description	Example
||	Concatenates character strings	`SELECT 'The Name of the employee is: ' || ENAME FROM EMP;`

2.3.1 Concatenating Character Strings

With Oracle Database Lite, you can concatenate character strings with the following results.

- Concatenating two character strings results in another character string.
- Oracle Database Lite preserves trailing blanks in character strings by concatenation, regardless of the strings' datatypes.
- Oracle Database Lite provides the `CONCAT` character function as an alternative to the vertical bar operator. For example,

```
SELECT CONCAT (CONCAT (ENAME, ' is a '),job) FROM EMP WHERE SAL > 2000;
```

This returns the following output.

```
CONCAT(CONCAT(ENAME
-------------------------
KING       is a PRESIDENT
BLAKE      is a MANAGER
CLARK      is a MANAGER
JONES      is a MANAGER
FORD       is a ANALYST
SCOTT      is a ANALYST

6 rows selected.
```

- Oracle Database Lite treats zero-length character strings as nulls. When you concatenate a zero-length character string with another operand the result is always the other operand. A null value can only result from the concatenation of two null strings.

2.4 Comparison Operators

Comparison operators used in conditions that compare one expression with another are listed in Table 2–4. The result of a comparison can be `TRUE`, `FALSE`, or `UNKNOWN`.

Table 2–4 Comparison Operators

Operator	Description	Example
=	Equality test.	`SELECT ENAME "Employee" FROM EMP WHERE SAL = 1500;`
!=, ^=, <>	Inequality test.	`SELECT ENAME FROM EMP WHERE SAL ^= 5000;`
>	Greater than test.	`SELECT ENAME "Employee", JOB "Title" FROM EMP WHERE SAL > 3000;`
<	Less than test.	`SELECT * FROM PRICE WHERE MINPRICE < 30;`

Table 2–4 (Cont.) Comparison Operators

Operator	Description	Example
>=	Greater than or equal to test.	`SELECT * FROM PRICE WHERE MINPRICE >= 20;`
<=	Less than or equal to test.	`SELECT ENAME FROM EMP WHERE SAL <= 1500;`
IN	"Equivalent to any member of" test. Equivalent to "`=ANY`".	`SELECT * FROM EMP WHERE ENAME IN ('SMITH', 'WARD');`
ANY/ SOME	Compares a value to each value in a list or returned by a query. Must be preceded by `=, !=, >, <, <= or >=`. Evaluates to `FASLE` if the query returns no rows.	`SELECT * FROM DEPT WHERE LOC = SOME ('NEW YORK','DALLAS');`
NOT IN	Equivalent to "`!=ANY`". Evaluates to `FALSE` if any member of the set is `NULL`.	`SELECT * FROM DEPT WHERE LOC NOT IN ('NEW YORK', 'DALLAS');`
ALL	Compares a value with every value in a list or returned by a query. Must be preceded by `=, !=, >, <, <= or >=`. Evaluates to `TRUE` if the query returns no rows.	`SELECT * FROM emp WHERE sal >= ALL (1400, 3000);`
[NOT] BETWEEN *x* and *y*	[Not] greater than or equal to *x* and less than or equal to *y*.	`SELECT ENAME, JOB FROM EMP WHERE SAL BETWEEN 3000 AND 5000;`
EXISTS	`TRUE` if a sub-query returns at least one row.	`SELECT * FROM EMP WHERE EXISTS (SELECT ENAME FROM EMP WHERE MGR IS NULL);`
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 following `ESCAPE` is interpreted literally, useful when *y* contains a percent (%) or underscore (_).	`SELECT * FROM EMP WHERE ENAME LIKE '%E%';`
IS [NOT] NULL	Tests for nulls. This is the only operator that should be used to test for nulls.	`SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500;`

2.5 Logical Operators

Logical operators which manipulate the results of conditions are listed in Table 2–5.

Table 2–5 Logical Operators

Operator	Description	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

2.6 Set Operators

Set operators which combine the results of two queries into a single result are listed in Table 2–6.

Table 2–6 Set Operators

Operator	Description	Example
UNION	Returns all distinct rows selected by either query.	SELECT * FROM (SELECT ENAME FROM EMP WHERE JOB = 'CLERK' UNION SELECT ENAME FROM EMP WHERE JOB = 'ANALYST');
UNION ALL	Returns all rows selected by either query, including all duplicates.	SELECT * FROM (SELECT SAL FROM EMP WHERE JOB = 'CLERK' UNION SELECT SAL FROM EMP WHERE JOB = 'ANALYST');
INTERSECT and INTERSECT ALL	Returns all distinct rows selected by both queries.	SELECT * FROM orders_list1 INTERSECT SELECT * FROM orders_list2

Table 2–6 (Cont.) Set Operators

Operator	Description	Example
MINUS	Returns all distinct rows selected by the first query but not the second.	SELECT * FROM (SELECT SAL FROM EMP WHERE JOB = 'PRESIDENT' MINUS SELECT SAL FROM EMP WHERE JOB = 'MANAGER');

Note: : The syntax for INTERSECT ALL is supported, but it returns the same results as INTERSECT.

2.7 Other Operators

Other operators used by Oracle Database Lite are listed in Table 2–7.

Table 2–7 Other Operators

Operator	Description	Example
(+)	Indicates that the preceding column is the outer join column in a join.	SELECT ENAME, DNAME FROM EMP, DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO (+);
PRIOR	Evaluates the following expression for the parent row of the current row in a hierarchical, or tree-structured query. In such a query, you must use this operator in the CONNECT BY clause to define the relationship between the parent and child rows.	SELECT EMPNO, ENAME, MGR FROM EMP CONNECT BY PRIOR EMPNO = MGR;

3

SQL Functions

This document discusses SQL functions used with Oracle Database Lite. Topics include:

3.1 SQL Function Types

This section lists the different types of SQL functions.The "SQL Functions Overview" provides an explanation of each function.

SQL Function Types

Number Function	SQL Types
CEIL	ROUND - Number Function
FLOOR	TRUNC
MOD	

Character Function	SQL Types
CHR	ROUND - Date Function
CONCAT	SUBSTR
INITCAP	SUBSTRB
LCASE See LOWER	TRANSLATE
LOWER	TRIM
LPAD	UCASE See UPPER
LTRIM	UPPER
REPLACE	USER
RPAD	

Character Functions	Returning Number Values
ASCII	LENGTH
BIT_LENGTH (See LENGTH)	LENGTHB
CHAR_LENGTH (See LENGTH)	OCTET_LENGTH (See LENGTH)
INSTR	POSITION
INSTRB	

Date Functions	SQL Types		
ADD_MONTHS	DAYOFMONTH	MONTHNAME	TIMESTAMPADD
CURDATE	DAYOFWEEK	MONTHS_BETWEEN	TIMESTAMPDIFF
CURRENT_DATE	DAYOFYEAR	NEXT_DAY	TRUNC
CURRENT_TIME	HOUR	NOW	WEEK
CURRENT_ TIMESTAMP	LAST_DAY	ROUND - Date Function	YEAR
CURTIME	MINUTE	SECOND	
DAYNAME	MONTH	SYSDATE	

Conversion Functions	Other Functions	Grouping Functions
CAST	CASE	AVG
CONVERT	DATABASE	COUNT
TO_CHAR	DECODE	MAX
TO_NUMBER	EXTRACT	MIN
TO_DATE	GREATEST	STDDEV
	IFNULL (See CASE and NVL)	SUM
	INTERVAL	VARIANCE
	LEAST	
	LOCATE	
	NVL	
	SUBSTR	
	USER	

3.2 SQL Functions Overview

SQL functions are similar to SQL operators in that both manipulate data items and both return a result. SQL functions differ from SQL operators in the format in which they appear with their arguments. The SQL function format enables functions to operate with zero, one, or more arguments.

```
function(argument1, argument2, ...) alias
```

If passed an argument whose datatype differs from an expected datatype, most functions perform an implicit datatype conversion on the argument before execution. If passed a null value, most functions return a null value.

SQL functions are used exclusively with SQL commands within SQL statements. There are two general types of SQL functions: single row (or scalar) functions and aggregate functions. These two types differ in the number of database rows on which they act. A single row function returns a value based on a single row in a query, whereas an aggregate function returns a value based on all the rows in a query.

Single row SQL functions can appear in select lists (except in `SELECT` statements that contain a `GROUP BY` clause) and `WHERE` clauses.

Aggregate functions are the `set` functions: `AVG`, `MIN`, `MAX`, `SUM`, and `COUNT`. You must provide them with an alias that can be used by the `GROUP BY` function.

Most functions have an SQL form and an ODBC form that can differ slightly in functionality.

3.2.1 Number Functions

Number functions accept numeric input and return numeric values.

3.2.2 Character Functions

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

3.2.3 Character Functions Returning Number Values

Some character functions return only number values.

3.2.4 Date Functions

Date functions operate on values of the `DATE` datatype. All date functions return a value of the `DATE` datatype, except the `MONTHS_BETWEEN` function which returns a number.

3.2.5 Conversion Functions

Conversion functions convert a value from one datatype to another. Generally, the form of the function name follows the convention *datatype* TO *datatype*. The first datatype is the input datatype; the last datatype is the output datatype.

3.3 SQL Functions Alphabetical Listing

This section lists Oracle Database Lite SQL functions in alphabetical order and defines each function. The discussion includes:

- Syntax
- Purpose
- Argument and Description
- Examples
- Usage Notes
- ODBC Functionality (where relevant)

3.3.1 ADD_MONTHS

Syntax

```
ADD_MONTHS(d, n)
```

d, a value of the Date datatype.

n, an integer that represents a number of months.

Purpose

Adds a specified date *d* to a specified number of months *n* and returns the resulting date. If the day component of argument *d* is the last day of the month, or if the resulting month has fewer days than the day component of *d*, then ADD_MONTHS returns the last day of the resulting month. Otherwise, ADD_MONTHS returns a value that has the same day component as *d*.

Example

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

Returns the following result.

```
TO_CHAR(ADD_MONTHS(HIREDATE              Next month
---------------------------------------- ----------
1981-01-17                               DD-MM-YYYY
```

3.3.2 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.

Example

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

Returns the following result.

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

3.3.3 AVG

Syntax

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

Purpose

Returns the average value of a column *n*.

Example 1

```
SELECT AVG(SAL) FROM EMP;
```

Returns the following result.

```
 AVG(SAL)
---------
  2073.21
```

Example 2

```
SELECT {FN AVG (SAL)} FROM EMP;
```

Returns the following result.

```
{FNAVG(SAL)}
------------
     2073.21
```

Example 3

```
SELECT AVG (DISTINCT DEPTNO) FROM EMP;
```

Returns the following result.

```
AVG(DISTINCTDEPTNO)
-------------------
                 20
```

Example 4

```
SELECT AVG (ALL DEPTNO) FROM EMP;
```

Returns the following result.

```
AVG(ALLDEPTNO)
--------------
        22.142
```

ODBC Function

```
{FN AVG ([DISTINCT | ALL] n)}
```

where *n* is the name of a numeric column.

3.3.4 CASE

Syntax

```
CASE
WHEN condition 1
 THEN result 1
WHEN condition 2
 THEN result 2
...
WHEN condition n
 THEN result n
ELSE result x
END,
```

Purpose

Specifies a conditional value using arguments listed in Table 3–1.

Table 3–1 Arguments Used with the CASE Function

Argument	Description
WHEN	Begins a condition clause.
condition	Specifies the condition.
THEN	Begins a result clause.
result	Specifies the result of the associated condition.
ELSE	An optional clause specifying the result of any value not described in a condition clause.
END	Terminates the case statement.

Usage Notes

The CASE function specifies conditions and results for a select or update statement. You can use the CASE function to search for data based on specific conditions or to update values based on a condition.

Example

```
SELECT CASE JOB
WHEN 'PRESIDENT' THEN 'The Honorable'
WHEN 'MANAGER' THEN 'The Esteemed'
ELSE 'The good'
END,
ENAME
FROM EMP;
```

Returns the following result.

```
CASEJOBWHEN'PRESI ENAME
----------------- ----------
The Honorable     KING
The Esteemed      BLAKE
The Esteemed      CLARK
The Esteemed      JONES
The good          MARTIN
The good          ALLEN
The good          TURNER
The good          JAMES
The good          WARD
The good          FORD
The good          SMITH
The good          SCOTT
The good          ADAMS
The good          MILLER

14 rows selected.
```

3.3.5 CAST

Syntax

```
SELECT CAST ( <source_operand > AS <data_type > ) FROM DUAL;
```

Purpose

Converts data from one type to another type using arguments listed in Table 3–2.

Table 3–2 Arguments Used with the CAST Function

Argument	Description
<*source_operand*>	a value expression or NULL.
<*data_type*>	the type of target.

Usage Notes

The table in Figure 3–1 displays the conversion results of source operands to datatypes.

Figure 3–1 Conversion Results of Source Operands and Datatypes

Conversion Results

<*source_operand*> \ <*data_type*>	EN	AN	VC	FC	D	T	TS	YM	DT
EN	V	V	V	V	X	X	X	R	R
AN	V	V	V	X	X	X	X	X	X
C	V	R	R	V	V	V	V	V	X
D	X	X	V	V	V	X	V	X	X
T	X	X	V	V	X	V	V	X	X
TS	X	X	V	V	V	V	V	X	X
YM	R	X	V	V	X	X	X	V	X
DT	R	X	V	V	X	X	X	X	V

The conversion results of source operands to datatypes are defined in Table 3–3.

Table 3–3 Definitions of Conversion Results and Source Operands

Result Definitions	Source Operands
EN = exact number	D = date
C = fixed or variable length character	TS = timestamp
VC = variable length character	DT = date-time
T = time	V = valid
YM = year-month interval	R = valid with restrictions
AN = approximate numeric	X = invalid
FC = fixed length character	

If <*source_operand*> is an exact numeric and <*data_type*> is an interval, then the interval contains a single date-time field.

If <*source_operand*> is an interval and <*data_type*> is an exact numeric, then the interval contains a single date-time field.

If <*source_operand*> is a character string and <*data_type*> specifies a character string, then their character repertoire is the same.

If <*data_type*> is numeric and the result cannot be represented without losing leading significant digits, then the following exception is raised: data-exception, numeric value out of range.

Example 1

```
SELECT CAST('0' AS INTEGER) FROM DUAL;
```

Returns the following result.

```
CAST('0'ASINTEGER)
------------------
                 0
```

Example 2

```
SELECT CAST(0 AS REAL) FROM DUAL;
```

Returns the following result.

```
CAST(0ASREAL)
-------------
            0
```

Example 3

```
SELECT CAST(1E0 AS NUMERIC(12, 2)) FROM DUAL;
```

Returns the following result.

```
CAST(1E0ASNUMERIC(12
--------------------
                   1
```

Example 4

```
SELECT CAST(CURRENT_TIMESTAMP AS VARCHAR(30)) FROM DUAL;
```

Returns the following result.

```
CAST(CURRENT_TIMESTAMPASVARCH
-----------------------------
1999-04-12 14:53:53
```

3.3.6 CEIL

Syntax

```
CEIL (n)
```

Purpose

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

Example

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

Returns the following result.

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

3.3.7 CHR

Syntax

```
CHR (n)
```

Purpose

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

Example

```
SELECT CHR(68)||CHR(79)||CHR(71) "Dog" FROM DUAL;
```

Returns the following result.

```
Dog
---
DOG
```

3.3.8 CONCAT

Syntax

```
CONCAT(char1, char2)
```

or

```
CHAR1 || CHAR2
```

Purpose

Returns *char1* concatenated with *char2*, where *char1* and *char2* are string arguments. This function is equivalent to the concatenation operator (||).

Example

This example uses nesting to concatenate three character strings.

```
SELECT CONCAT( CONCAT(ename, ' is a '), job) "Job"
FROM emp
WHERE empno = 7900;
```

Returns the following result.

```
Job
-------------------------
JAMES      is a CLERK
```

ODBC Function

```
{FN CONCAT (char1, char2)}
```

3.3.9 CONVERT

Syntax

```
{ fn CONVERT(value_exp, data_type) }
```

Purpose

Converts a character string from one character set to another.

The *value_exp* argument is the value to be converted.

The *data_type* argument is the name of the character set to which *char* is converted.

Usage Notes

The common character sets are listed in Table 3–4.

Table 3–4 Common Character Sets Used with the CONVERT Function

Common Character Sets	
US7ASCII	WE8ISO8859P1
WE8DEC	HP West European Laserjet 8-bit character set
WE8HP	DEC French 7-bit character set
F7DEC	IBM West European EBCDIC Code Page 500
WE8EBCDIC500	IBM PC Code Page 850 ISO 8859-1 West European 8-bit character set
WE8PC850	ISO 8859-1 West European 8-bit character set

Example

```
SELECT {fn CONVERT('Groß', 'US7ASCII') }
"Conversion" FROM DUAL;
```

Returns the following result.

```
conversi
--------
Groß
```

3.3.10 COUNT

Syntax

```
COUNT({* | [DISTINCT | ALL] expr})
```

Purpose

Returns the number of rows in the query.

Example 1

```
SELECT COUNT(*) "Total" FROM emp;
```

Returns the following result.

```
Total
----------
14
```

Example 2

```
SELECT COUNT(job) "Count" FROM emp;
```

Returns the following result.

```
Count
----------
```

```
14
```

Example 3

```
SELECT COUNT(DISTINCT job) "Jobs" FROM emp;
```

Returns the following result.

```
Jobs
----------
5
```

Example 4

```
SELECT COUNT (ALL JOB) FROM EMP;
```

Returns the following result.

```
COUNT(ALLJOB)
-------------
```

3.3.11 CURDATE

Syntax

```
{ fn CURDATE ( <value_expression > ) }
```

Purpose

Returns the current date.

Usage Notes

If you specify *expr* (expression), 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.

Example 1

```
SELECT {fn CURDATE()} FROM DUAL;
```

Returns the following result.

```
{FNCURDATE
-----------
1999-04-12
```

Example 2

```
SELECT {fn WEEK({fn CURDATE()})} FROM DUAL;
```

Returns the following result.

```
{FNWEEK({FNCURDATE()})}
-----------------------
                     16
```

3.3.12 CURRENT_DATE

Syntax

```
CURRENT_DATE
```

Purpose

Returns the current date.

Example

```
SELECT CURRENT_DATE FROM DUAL;
```

Returns the following result.

```
CURRENT_DATE
------------
1999-04-12
```

ODBC Function

```
{fn CURDATE()}
```

3.3.13 CURRENT_TIME

Syntax

```
CURRENT_TIME
```

Purpose

Returns the current time.

Example

```
SELECT CURRENT_TIME FROM DUAL;
```

Returns the following result.

```
CURRENT_T
---------
15:54:18
```

ODBC Function

```
{fn CURTIME()}
```

3.3.14 CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP
```

Purpose

Returns the current local date and local time as a timestamp value but only displays the current local date by default. You can view current local time information by using `CURRENT_TIMESTAMP` as a value of the `TO_CHAR` function and by including a time format. For more information, see Example 2.

Example 1

```
SELECT CURRENT_TIMESTAMP FROM DUAL;
```

Returns the following result.

```
CURRENT_TI
---------
1999-04-12
```

Example 2

```
SELECT TO_CHAR (CURRENT_TIMESTAMP, 'HH24:MM:SS, Day, Month, DD, YYYY')FROM DUAL;
```

Returns the following result.

```
TO_CHAR(CURRENT_TIMESTAMP
----------------------------------------
18:04:05, Tuesday  , April    , 06, 1999
```

ODBC Function

```
{fn CURTIME()}
```

3.3.15 CURTIME

Syntax

```
{ fn CURTIME ( <value_expression > ) }
```

Purpose

Returns the current time.

Example 1

```
SELECT {fn CURTIME()} FROM DUAL;
```

Returns the following result.

```
{FNCURTIM
---------
11:09:59
```

Example 2

```
SELECT {fn HOUR({fn CURTIME()})} FROM DUAL;
```

Returns the following result.

```
{FNHOUR({FNCURTIME()})}
-----------------------
                     11
```

3.3.16 DATABASE

Syntax

```
{ fn DATABASE () }
```

Purpose

Specifies the name of the database. If you are using ODBC, the DATABASE function returns the name of the current default database file without the **.ODB** extension.

Usage Notes

A database name function returns the same value as that of `SQLGetConnectOption()` with the option `SQL_CURRENT_QUALIFIER`.

Example

The following example returns a result for users connected to the default database.

```
SELECT {fn DATABASE () } FROM DUAL;
```

Returns the following result.

```
{FNDATABASE()}
--------------
POLITE
```

3.3.17 DAYNAME

Syntax

```
{ fn DAYNAME (date_expr) }
```

Purpose

Returns the day of the week as a string.

Example

```
SELECT {fn dayname({fn curdate()})} from dual;
```

Returns the current day of the week as a string.

3.3.18 DAYOFMONTH

Syntax

```
{ fn DAYOFMONTH ( <value_expression > ) }
```

Purpose

Returns the day of the month as an integer using arguments listed in Table 3–5.

Table 3–5 Argument Used with the DAYOFMONTH Function

Argument	Description
<*value_expression*>	Date on which the day of the month is computed. The result is between 1 and 31, where 1 represents the first day of the month.

Example 1

```
SELECT {fn DAYOFMONTH ({fn CURDATE()})} FROM DUAL;
```

Returns the following result:

```
{FNDAYOFMONTH({FNCURDATE()})
----------------------------
                          12
```

Example 2

```
SELECT {fn DAYOFMONTH('1997-07-16')} "DayOfMonth" FROM DUAL;
```

Returns the following result.

```
DayOfMonth
----------
        16
```

3.3.19 DAYOFWEEK

Syntax

```
{ fn DAYOFWEEK ( <value_expression > ) }
```

Purpose

Returns the day of the week as an integer using arguments listed in Table 3–6.

Table 3–6 Argument Used with the DAYOFWEEK Function

Argument	Description
<*value_expression*>	Date on which the day of the week is computed. The result is between 1 and 7, where 1 represents Sunday.

Example 1

```
SELECT {fn DAYOFWEEK ({fn CURDATE()})} FROM DUAL;
```

Returns the following result.

```
{FNDAYOFWEEK({FNCURDATE()})}
----------------------------
                           2
```

Example 2

```
SELECT {fn DAYOFWEEK('1997-07-16')} "DayOfWeek" FROM DUAL;
```

Returns the following result.

```
DayOfWeek
------------------
4
```

3.3.20 DAYOFYEAR

Syntax

```
{ fn DAYOFYEAR ( <value_expression > ) }
```

Purpose

Returns the day of the year as an integer using arguments listed in Table 3–7.

Table 3–7 Argument Used with the DAYOFYEAR Function

Argument	Description
<*value_expression*>	A date on which the day of the year is computed. The result is between 1 and 366.

Example 1

```
SELECT {fn DAYOFYEAR ({fn CURDATE()})} FROM DUAL;
```

Returns the following result.

```
{FNDAYOFYEAR({FNCURDATE()})}
----------------------------
                         102
```

Example 2

```
SELECT {fn DAYOFYEAR('1997-07-16')} "DAYOFYEAR" FROM DUAL;
```

Returns the following result.

```
DayOfYear
---------
197
```

3.3.21 DECODE

Syntax

```
DECODE (expr, search, result [, search, result...] [,default])
```

Purpose

Search for an expression's values and then evaluate them in terms of a specified result.

Usage Notes

To evaluate an expression, Oracle Database Lite compares the expression to each search value one by one. If the expression is equal to a search, Oracle Database Lite returns the corresponding result. If no match is found, Oracle Database Lite returns default, or, if default is omitted, returns null. If the expression and search contain character data, Oracle Database Lite compares them using non-padded comparison semantics.

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

Oracle Database Lite automatically converts the expression and each search value to the datatype of the first search value before making comparisons. Oracle Database Lite automatically converts the return value to the same datatype as the first result. If the first result has the datatype `CHAR` or if the first result is null, then Oracle Database Lite converts the return value to the datatype `VARCHAR2`.

In a `DECODE` expression, Oracle Database Lite considers two nulls to be equivalent. If the expression is null, Oracle Database Lite returns the result of the first search that is also null.

The maximum number of components in the `DECODE` expression, including the expression, searches, results, and default is 255.

Example 1

The following expression decodes the `DEPTNO` column in the `DEPT` table. If `DEPTNO` is 10, the expression evaluates to `'ACCOUNTING'`; if `DEPTNO` is 20, it evaluates to

'RESEARCH'; and so on. If DEPTNO is not 10, 20, 30, or 40, the expression returns 'NONE'.

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

Example 2

The following example uses the DECODE clause in a SELECT statement.

```
SELECT DECODE (deptno, 10, 'ACCOUNTING',
20, 'RESEARCH',
30, 'SALES',
40, 'OPERATIONS',
'NONE')
FROM DEPT;
```

Returns the following result.

```
DECODE(DEP
----------
ACCOUNTING
RESEARCH
SALES
OPERATIONS
```

3.3.22 EXTRACT

Syntax

EXTRACT (*extract-field FROM extract source*)

Purpose

Returns information from the *i* portion of the *extract-source*. The *extract-source* argument contains date-time or interval expressions. The *extract-field* argument contains one of the following keywords: YEAR, MONTH, DAY, HOUR, MINUTE, or SECOND.

The precision of the returned value is defined in implementation. The scale is 0 unless SECOND is specified. When SECOND is specified, the scale is not less than the fractional seconds precision of the *extract-source* field.

Example 1

```
SELECT EXTRACT (DAY FROM '06-15-1966') FROM DUAL;
```

Returns the following result.

```
EXTRACT(DAY
-----------
         15
```

Example 2

```
SELECT  EXTRACT (YEAR FROM {FN CURDATE()}) FROM DUAL;
```

Returns the following result.

```
EXTRACT(YEAR
```

```
------------
        1999
```

3.3.23 FLOOR

Syntax

```
FLOOR (n)
```

Purpose

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

Example

```
SELECT FLOOR(15.7) "Floor" FROM DUAL;
```

Returns the following result.

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

3.3.24 GREATEST

Syntax

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

Purpose

Returns the greatest of the list of *exprs* (expressions). All *exprs* after the first are implicitly converted to the datatype of the first *exprs* before the comparison. Oracle Database Lite compares the *exprs* using non padded comparison semantics. Character comparison is based on the value of the character in the database character set. One character is greater than another if it has a higher value. If the value returned by this function is character data, its datatype is always `VARCHAR2`.

Example

```
SELECT GREATEST('HARRY','HARRIOT','HAROLD') "GREATEST" FROM DUAL;
```

Returns the following result.

```
GREATEST
--------
HARRY
```

3.3.25 HOUR

Syntax

```
HOUR (time_exp)
```

Purpose

Returns the hour as an integer value in the range of 0-23.

Example 1

```
SELECT {FN HOUR ('14:03:01')} FROM DUAL;
```

Returns the following result.

```
{FNHOUR('14:03:01')}
--------------------
                  14
```

Example 2

```
SELECT {fn HOUR({fn CURTIME()})} FROM DUAL;
```

Returns the following result.

```
{FNHOUR({FNCURTIME()})}
-----------------------
                     11
```

3.3.26 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;
```

Returns the following result.

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

3.3.27 INSTR

Syntax

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

Purpose

Searches the string argument *char1*, beginning with its *nth* character, for the *mth* occurrence of string argument *char2*, where *m* and *n* are numeric arguments. Returns the position in *char1* of the first character of this occurrence.

Usage Notes

If *n* is negative, INSTR 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 that INSTR begins searching at the first character of *char1* for the first occurrence of *char2*. The return value is relative to the beginning of *char1*, regardless of the value of *n*, and is expressed in characters. If the search is unsuccessful (if *char2* does not appear *m* times after the *nth* character of *char1*), the return value is *0*. For additional information, see the syntax for the POSITION function.

Example

```
SELECT INSTR('CORPORATE FLOOR','OR',3,2) "Instring" FROM DUAL;
```

Returns the following result.

```
 Instring
---------
       14
```

3.3.28 INSTRB

Syntax

```
INSTRB(char1, char2, [, n [, m ]])
```

Purpose

Searches the string argument *char1*, beginning with its *nth* byte, for the *mth* occurrence of string argument *char2*, where *m* and *n* are numeric arguments. Returns the position in *char1* of the first byte of this occurrence. The same as INSTR except that *n* and the function's return value are expressed in bytes rather than 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;
```

Returns the following result.

```
Instring in bytes
-----------------
               14
```

3.3.29 INTERVAL

Syntax

```
INTERVAL (datetime values)
```

Purpose

Subtracts one `datetime` from another and generates the result. When you add or subtract one interval from another, the result is always another interval. You can multiply or divide an interval by a numeric constant.

Example 1

```
SELECT CURRENT_DATE - INTERVAL '8' MONTH FROM DUAL;
```

Returns the following result.

```
CURRENT_DATE-INTERVAL
---------------------
1998-08-09
```

Example 2

```
SELECT TO_CHAR (INTERVAL '6' DAY * 3) FROM DUAL;
```

Returns the following result.

```
TO_CHAR(INTERVAL'6'DAY*3)
```

```
------------------------
18
```

3.3.30 LAST_DAY

Syntax

```
LAST_DAY(d)
```

Purpose

Returns a date that represents the last day of the month in which date *d* occurs.

Usage Notes

You can use this function to determine how many days are left in the current month.

Example 1

```
SELECT LAST_DAY (SYSDATE) FROM DUAL;
```

Returns the following result.

```
LAST_DAY
----------
1999-04-30
```

Example 2

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

Returns the following result.

```
{FNNOW()}   Last        Days Left
----------  ----------  ---------
1999-04-12  1999-04-30         18
```

3.3.31 LEAST

Syntax

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

Purpose

Returns the least of the list of *exprs* (expressions). All *exprs* after the first are implicitly converted to the datatype of the first *exprs* before the comparison. Oracle Database Lite compares the *exprs* using non-padded comparison semantics. Character comparison is based on the value of the character in the database character set. One character is less than another if it has a lower value. If the value returned by this function is character data, its datatype is always `VARCHAR2`.

Example

```
SELECT LEAST('HARRY','HARRIOT','HAROLD') "LEAST" FROM DUAL;
```

Returns the following result.

```
LEAST
```

```
-------
HAROLD
```

3.3.32 LENGTH

Syntax

```
LENGTH (char)
{fn LENGTH(char)}
BIT_LENGTH (char)
CHAR_LENGTH (char)
OCTET_LENGTH (char)
```

LENGTH returns the number of characters in *char*. BIT_LENGTH, CHAR_LENGTH, and OCTET_LENGTH return the length of *char* in bits, characters, or octets, respectively.

Purpose

Returns the length in characters of the string argument *char*. If *char* has the datatype CHAR, the length includes all trailing blanks. If *char* is null, it returns null.

Usage Notes

BIT_LENGTH, CHAR_LENGTH, and OCTET_LENGTH are SQL-92 functions. CHAR_LENGTH is the same as LENGTH, and OCTET_LENGTH is the same as LENGTHB.

Example

```
SELECT LENGTH('CANDIDE') "Length in characters" FROM DUAL;
```

Returns the following result.

```
Length in characters
--------------------
                   7
```

3.3.33 LENGTHB

Syntax

```
LENGTHB(char)
{fn LENGTHB(char)}
```

Purpose

Returns the length in bytes of the string argument *char*. If *char* is null, it returns null. For a single-byte database character set, LENGTHB is equivalent to LENGTH.

Example

```
SELECT LENGTHB('CANDIDE') "Length in bytes" FROM DUAL;
```

Returns the following result.

```
Length in bytes
---------------
              7
```

3.3.34 LOCATE

Syntax

```
LOCATE (string_exp1, string_exp2[,start])
```

Purpose

Returns the starting position of the first occurrence of *string_exp1* within the first character position of *string_exp2*. You can use the *start* value to specify a search location other than the first character position of *string_exp2*.

Example 1

The following example selects the starting position of the character 'R' in the string expression 'TURNER' for every row of the EMP table.

```
SELECT {FN LOCATE ('R', 'TURNER')} FROM EMP ENAME;
```

Returns the following result.

```
{FNLOCATE('R'
-------------
            3
            3
            3
            3
            3
            3
            3
            3
            3
            3
            3
            3
            3
            3

14 rows selected.
```

Example 2

The following example selects the starting position of the character 'R' in the string expression 'TURNER' and starts its search at the fourth character in 'TURNER'. The example displays the results found for every occurrence of 'TURNER' in every row of the EMP table.

```
SELECT {FN LOCATE ('R', 'TURNER',4)} FROM EMP ENAME;
```

Returns the following result.

```
{FNLOCATE('R'
-------------
            6
            6
            6
            6
            6
            6
            6
            6
            6
```

```
                6
                6
                6
                6
                6

14 rows selected.
```

3.3.35 LOWER

Syntax

```
LOWER(char)
```

Purpose

Returns a string argument *char*, with all its letters in lowercase. The return value has the same datatype as *char*, either CHAR or VARCHAR2.

Example

```
SELECT LOWER('LOWER') FROM DUAL;
```

Returns the following result.

```
LOWER
-----
lower
```

ODBC Function

```
{fn LCASE (char)}
```

3.3.36 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('Page1',15,'*.') "LPAD example" FROM DUAL;
```

Returns the following result.

```
LPAD example
----------------
*.*.*.*.*.Page1
```

3.3.37 LTRIM

Syntax

```
LTRIM(char [, set])
```

Purpose

Returns the string argument *char*, with its left-most characters removed up to the first character which is not in the string argument *set*, which defaults to (a single space).

Example

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

Returns the following result.

```
LTRIM example
---------------
XxyLAST WORD
```

ODBC Function

```
{fn LTRIM (char) }      (trims leading blanks)
```

3.3.38 MAX

Syntax

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

Purpose

Returns the maximum value of an expression specified by the argument *expr*.

Example

```
SELECT MAX(SAL) FROM EMP;
```

Returns the following result.

```
MAX(SAL)
---------
5000
```

3.3.39 MIN

Syntax

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

Purpose

Returns the minimum value of an expression specified by the argument *expr*.

Example

```
SELECT MIN(SAL), MAX(SAL) FROM EMP;
```

Returns the following result.

```
 MIN(SAL)
---------
```

```
       800
```

3.3.40 MINUTE

Syntax

```
MINUTE (time_exp)
```

Purpose

Returns the minute as an integer value in the range of 0-59.

Example 1

```
SELECT {FN MINUTE ('14:03:01')} FROM DUAL;
```

Returns the following result.

```
{FNMINUTE('14:03:01')}
----------------------
                     3
```

Example 2

```
SELECT {fn MINUTE({fn CURTIME()})} FROM DUAL;
```

Returns the following result.

```
{FNMINUTE({FNCURTIME()})}
-------------------------
                       23
```

3.3.41 MOD

Syntax

```
MOD (m,n)
```

Purpose

Returns the remainder of *m* divided by *n*. Returns *m* if *n* is 0.

Example

```
SELECT MOD (26,11) "ABLOMOV" FROM DUAL;
```

Returns the following result.

```
  ABLOMOV
---------
        4
```

3.3.42 MONTH

Syntax

```
MONTH (date_exp)
```

Purpose

Returns the month as an integer value in the range of 1-12.

Example 1

```
SELECT {FN MONTH ('06-15-1966')} FROM DUAL;
```

Returns the following result.

```
{FNMONTH('06-15-1966')}
-----------------------
                      6
```

Example 2

```
SELECT {fn MONTH({fn CURDATE()})} FROM DUAL;
```

Returns the following result.

```
{FNMONTH({FNCURDATE()})}
------------------------
                       4
```

3.3.43 MONTHNAME

Syntax

```
{ fn MONTHNAME (date_exp) }
```

Purpose

Returns the name of the month as a string.

Example

```
select {fn monthname({fn curdate()})} from dual;
```

Returns the current month of the year as a string.

3.3.44 MONTHS_BETWEEN

Syntax

```
MONTHS_BETWEEN(d1, d2 )
```

Purpose

Returns number of months between dates *d1* and *d2*. If *d1* is later than *d2*, result is positive; if earlier, negative. If *d1* and *d2* are either the same days of the month or both last days of months, the result is always an integer. Otherwise, Oracle Database Lite 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;
```

Returns the following result.

```
   Months
---------
```

```
1.0322581
```

3.3.45 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

```
SELECT NEXT_DAY('15-MAR-92','TUESDAY') "NEXT DAY" FROM DUAL;
```

Returns the following result.

```
NEXT DAY
----------
1992-03-17
```

3.3.46 NOW

Syntax

```
NOW
```

Purpose

Returns the current local date and local time as a timestamp value but only displays the current local date by default. You can view current local time information by using NOW as a value of the TO_CHAR function and by including a time format. For more information, see Example 2.

Example 1

```
SELECT {FN NOW()} FROM DUAL;
```

Returns the following result.

```
{FNNOW()}
----------
1999-04-07
```

Example 2

```
SELECT TO_CHAR ({fn NOW ('YYYY, Month, DD, HH24:MM:SS')}) FROM DUAL;
```

Returns the following result.

```
TO_CHAR({FNNOW('YYYY
----------------------------------------
1999-04-07 12:55:31
```

3.3.47 NVL

Syntax

```
NVL(expr1, expr2)
```

Purpose

If *expr1* is null, returns *expr2*; if *expr1* is not null, returns *expr1*. The arguments *expr1* and *expr2* must be of the same datatype.

Example 1

```
SELECT ename, NVL(TO_CHAR(COMM),'NOT APPLICABLE') "COMMISSION"
FROM emp
WHERE deptno = 30;
```

Returns the following result.

```
ENAME      COMMISSION
---------- ----------------
BLAKE      NOT APPLICABLE
MARTIN     1400.00
ALLEN      300.00
TURNER     .00
JAMES      NOT APPLICABLE
WARD       500.00

6 rows selected.
```

Example 2

```
SELECT {fn IFNULL(Emp.Ename, 'Unknown')},
NVL (Emp.comm, 0) FROM EMP;
```

Returns the following result.

```
{FNIFNULL( 'UNKNOWN')}
---------- -----------
KING                 0
BLAKE                0
CLARK                0
JONES                0
MARTIN            1400
ALLEN              300
TURNER               0
JAMES                0
WARD               500
FORD                 0
SMITH                0
SCOTT                0
ADAMS                0
MILLER               0

14 rows selected.
```

Example 3

```
SELECT sal+NVL(comm, 0) FROM EMP;
```

Returns the following result.

```
SAL+NVL(COMM
```

```
------------
        5000
        2850
        2450
        2975
        2650
        1900
        1500
         950
        1750
        3000
         800
        3000
        1100
        1300

14 rows selected.
```

ODBC Function

```
{fn IFNULL (expr1, expr2)}
```

3.3.48 POSITION

Syntax

```
POSITION ( <substring_value_expression>
                IN <value_expression> )
```

The arguments for the POSITION function are listed in Table 3–8.

Table 3–8 Arguments Used with the POSITION Function

Argument	Description
<*value_expression*>	a source string to search in.
<*substring_value_expression*>	a sub-string to search for.
<*start_len_cnt*>	the starting position for the search

Purpose

Returns the starting position of the first occurrence of a sub-string in a string.

Usage Notes

If the length of <*substring_value_expression*> is 0, the result is null. If <*substring_value_expression*> occurs in <*value_expression*>, the result is the position of the first character of <*substring_value_expression*>. Otherwise, the result is 0. If <*start_len_cnt*> is omitted, the function starts the search from position 1. For additional information, see the INSTR and INSTRB functions.

Example

```
SELECT POSITION ('CAT' IN 'CATCH') FROM DUAL;
```

Returns the following result.

```
POSITION('CAT'IN'CATCH')
------------------------
                       1
```

ODBC Function

```
{fn LOCATE ( <substring_value_expression> ,
  <value_expression>[, <start_len_cnt> ] ) }
```

3.3.49 QUARTER

Syntax

```
{ fn QUARTER ( <value_expression> ) }
```

The arguments for the QUARTER function are listed in Table 3–9.

Table 3–9 Arguments Used with the QUARTER Function

Argument	Description
<*value_expression*>	A date on which the quarter is computed. The result is between 1 and 4, where 1 represents January 1 through March 31.

Purpose

Returns the quarter of a date as an integer.

Example

```
SELECT {fn QUARTER ({fn CURDATE()})} FROM DUAL;
```

Returns the following result.

```
{FNQUARTER({FNCURDATE()})}
--------------------------
                         2
```

3.3.50 REPLACE

Syntax

```
REPLACE(char, search_string [, replacement_string])
```

Purpose

Returns *char* with every occurrence of *search_string* replaced with *replacement_string*, where *char*, *search_string*, and *replacement_string* are string arguments.

Usage Notes

If *replacement_string* is omitted or null, all occurrences of *search_string* are removed. If *search_string* is null, then char is returned. This function provides a super-set of the functionality provided by the TRANSLATE function. TRANSLATE provides single character, one to one, and substitution functions. REPLACE enables 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;
```

Returns the following result.

```
Changes
---------------
BLACK and BLUE
```

3.3.51 ROUND - Date Function

Syntax

```
ROUND(d [,fmt])
```

The format models to be used with the ROUND (and TRUNC) date function, and the units to which it rounds dates are listed in Table 3–10. The default model, DD, returns the date rounded to the day with a time of midnight.

Table 3–10 The Format Models with the ROUND Date Function

Formal Model	Rounding Unit
CC or SCC	Century
YYYY, SYYYY, YEAR, SYEAR, YYY, YY, Y	Year (rounds up on July 1)
IYYY, IYY, IY, I	ISO Year
Q	Quarter (rounds up in 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

Purpose

Returns *d* rounded to the unit specified by the format model *fmt*. If you omit *fmt*, *d* is rounded to the nearest day.

Example

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

Returns the following result.

```
FIRST OF
---------
1993-01-0
```

3.3.52 ROUND - Number Function

Syntax

```
ROUND(n [,m ])
```

Purpose

Returns *n* rounded to *m* places to the 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 1

```
SELECT ROUND (54.339, 2) FROM DUAL;
```

Returns the following result.

```
ROUND(54.339
------------
54.34
```

3.3.53 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';
```

Returns the following result.

```
RPAD example
-------------
enameababababa
```

3.3.54 RTRIM

Syntax

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

Purpose

Returns the string argument *char*, with its right-most characters removed following the last character which is not in the string argument set. This defaults to ' ' (a single space).

Example 1

```
SELECT RTRIM ('TURNERyxXxy', 'xy') "RTRIM example" FROM DUAL;
```

Returns the following result.

```
RTRIM examp
-----------
TURNERyxX
```

Example 2

```
SELECT {fn RTRIM ('TURNERyxXxy', 'xy')} FROM DUAL;
```

Returns the following result.

```
{{FNRTRIM('T
-----------
TURNERyxX
```

ODBC Function

```
{fn RTRIM (char)}    (trims leading blanks)
```

3.3.55 SECOND

Syntax

```
SECOND (time_exp)
```

Purpose

Returns the second as an integer value in the range of 0-59.

Example 1

```
SELECT {FN SECOND ('14:03:01')} FROM DUAL;
```

Returns the following result.

```
{FNSECOND('14:03:01')}
----------------------
                     1
```

Example 2

```
SELECT {fn SECOND({fn CURTIME()})} FROM DUAL;
```

Returns the following result.

```
{FNSECOND({FNCURTIME()})}
-------------------------
                       59
```

3.3.56 STDDEV

Syntax

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

Purpose

Returns the standard deviation of *x*, a number. Oracle Database Lite calculates the standard deviation as the square root of the variance defined for the VARIANCE group function.

Example

```
SELECT STDDEV(sal) "Deviation" FROM emp;
```

Returns the following result.

```
Deviation
---------
1182.5032
```

3.3.57 SUBSTR

Syntax

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

Purpose

Returns a portion of the string argument *char*, beginning with the character at position *m* and *n* characters long.

Usage Notes

If *m* is positive, SUBSTR counts from the beginning of *char* to find the first character. If *m* is negative, SUBSTR counts backwards from the end of *char*. The value *m* cannot be 0. If *n* is omitted, SUBSTR returns all characters to the end of *char*. The value *n* cannot be less than 1.

Example

```
SELECT SUBSTR('ABCDEFG',3,4) "Subs" FROM DUAL;
```

Returns the following result.

```
Subs
----
CDEF
```

3.3.58 SUBSTRB

Syntax

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

Purpose

Returns a portion of the string argument *char*, beginning with the byte at position *m* and *n* bytes long. The same as SUBSTR, except that the arguments *m* and *n* specify bytes rather than characters. For a single-byte database character set, SUBSTRB is equivalent to SUBSTR.

Example

```
SELECT SUBSTRB('ABCDEFG',5,4) "Substring with bytes" FROM DUAL;
```

Returns the following result.

```
Substring with bytes
--------------------
EFG
```

3.3.59 SUM

Syntax

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

Purpose

Returns the sum of values of *n*.

Example

```
SELECT deptno, SUM(sal) TotalSalary FROM emp GROUP BY deptno;
```

Returns the following result.

```
   DEPTNO TOTALSALARY
--------- -----------
       10        8750
       20       10875
       30        9400
```

3.3.60 SYSDATE

Syntax

```
SYSDATE
```

Purpose

Returns the current date and time. Requires no arguments.

Usage Notes

You cannot use this function in the condition of the Oracle Database Lite `DATA` type column. You can only use the time in a `TIME` column, and both date and time in a `TIMESTAMP` column.

Example

```
SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') NOW FROM DUAL;
```

Returns the following result.

```
NOW
-------------------
04-12-1999 19:13:48
```

3.3.61 TIMESTAMPADD

Syntax

```
{fn TIMESTAMPADD (<interval>, <value_exp1 >, <value_exp2 >)}
<value_exp1 > + <value_exp2 >
```

The arguments for the `TIMESTAMPADD` function are listed in Table 3–11.

Table 3–11 Arguments Used with the TIMESTAMPADD Function

Argument	Description
<*interval*>	Specifies the unit of the second operand, <*value_exp1*>. The following keywords are valid values for intervals.
	SQL_TSI_FRAC_SECOND
	SQL_TSI_SECOND
	SQL_TSI_MINUTE
	SQL_TSI_HOUR
	SQL_TSI_DAY
	SQL_TSI_WEEK
	SQL_TSI_MONTH
	SQL_TSI_QUARTER
	SQL_TSI_YEAR
<*value_exp1*>	an integer
<*value_exp2*>	a timestamp
<*value_expression*>	an operand

Purpose

Adds a date and time value to the current timestamp.

Example

The following example adds one day to the current timestamp for 1999-04-13.

```
SELECT {fn TIMESTAMPADD (SQL_TSI_DAY, 1, {fn NOW()})} FROM DUAL;
```

Returns the following result.

```
{FNTIMESTA
----------
1999-04-14
```

3.3.62 TIMESTAMPDIFF

Syntax

```
{fn TIMESTAMPDIFF (<interval>, <value_exp1 >, <value_exp2 >)}
<value_expression > - <value_expression >
```

The arguments for the TIMESTAMPDIFF function are listed in Table 3–12.

Table 3–12 Arguments Used with the TIMESTAMPDIFF Function

Argument	Description
<*interval*>	specifies the unit of the second operand, <*value_exp1*>. The following keywords are valid values for intervals:
	SQL_TSI_FRAC_SECOND
	SQL_TSI_SECOND
	SQL_TSI_MINUTE
	SQL_TSI_HOUR
	SQL_TSI_DAY
	SQL_TSI_WEEK
	SQL_TSI_MONTH
	SQL_TSI_QUARTER
	SQL_TSI_YEAR
<*value_exp1*>	an integer
<*value_exp2*>	a timestamp
<*value_expression*>	an operand

Purpose

Calculates the difference between two timestamp values using a specified interval.

Example 1

```
SELECT {fn TIMESTAMPDIFF (SQL_TSI_DAY, {fn CURDATE()}, '1998-12-09')} FROM DUAL;
```

Returns the following result.

```
{FNTIMESTAMPDIFF(SQL_TSI_DAY
----------------------------
                        -125
```

Example 2

```
SELECT ENAME, {fn TIMESTAMPDIFF (SQL_TSI_YEAR, {fn CURDATE()},HIREDATE)} FROM EMP;
```

Returns the following result.

```
ENAME      {FNTIMESTAMPDIFF(SQL_TSI_YEA
---------- ----------------------------
KING                                -17
BLAKE                               -17
CLARK                               -17
JONES                               -18
MARTIN                              -17
ALLEN                               -18
TURNER                              -17
JAMES                               -17
WARD                                -18
FORD                                -17
SMITH                               -18
SCOTT                               -16
ADAMS                               -16
MILLER                              -17

14 rows selected.
```

3.3.63 TO_CHAR

Syntax for Dates

```
TO_CHAR(d [, fmt])
```

Syntax for Numbers

```
TO_CHAR(n [, fmt])
```

Purpose

Converts a date or number to a value of the VARCHAR2 datatype, using the optional format *fmt* using arguments listed in Table 3–13.

Table 3–13 Arguments Used with the TO_CHAR Function

Argument	Description
d	date column or SYSDATE
fmt	format string
n	number column or literal

Usage Notes

- If you omit *fmt*, the argument *d* or *n* is converted to a VARCHAR2 value. For dates, the argument *d* is returned in the default date format. For numbers, the argument *n* is converted to a value exactly long enough to hold its significant digits.
- Date literals must be preceded by the DATE keyword when used as arguments to TO_CHAR.

You can specify a default date format for all databases on your computer by setting the NLS_DATE_FORMAT parameter in the **POLITE.INI** file. See the *Oracle Database Lite Administration and Deployment Guide* for more information on setting the NLS_DATE_FORMAT parameter in the **POLITE.INI** file.

Example

```
SELECT TO_CHAR (SYSDATE, 'Day, Month, DD, YYYY')"TO_CHAR example" FROM DUAL;
```

Returns the following result.

```
TO_CHAR example
--------------------------------
Saturday , May      , 22, 1999
```

3.3.64 TO_DATE

Syntax

```
TO_DATE(char [, fmt ])
```

Purpose

Converts the character string argument *char* to a value of the DATE datatype. The *fmt* argument is a date format specifying the format of *char*.

Example

```
SELECT TO_DATE('January 26, 1996, 12:38 A.M.', 'Month dd YYYY HH:MI A.M.') FROM
DUAL;
```

Returns the following result.

```
TO_CHAR(TO_DATE('JANUARY26
---------------------------------------
1996-01-26 12:38:00
```

3.3.65 TO_NUMBER

Syntax

```
TO_NUMBER(char [, fmt ])
```

Purpose

Converts the string argument *char* that contains a number in the format specified by the optional format model *fmt*, to a return value of the NUMBER datatype.

Usage Notes

- For information about date and number formats, see Formats.
- 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.
- Dates in the Oracle format (such as 06-JUN-85 and 6-JUN-1985), the SQL-92 format (such as 1989-02-28), or the format specified by the NLS_DATE_FORMAT parameter are converted automatically when inserted into a date column.
- You can specify a default date format for all databases on your computer by setting the NLS_DATE_FORMAT parameter in the **POLITE.INI** file. See the *Oracle Database Lite Administration and Deployment Guide* for more information on setting the NLS_DATE_FORMAT parameter in the **POLITE.INI** file.

Example

The following example updates the salary of an employee named Blake according to the value specified in the TO_NUMBER function. In this example, you first view Blake's salary. Then, update Blake's salary and view it again.

```
SELECT * FROM EMP WHERE ENAME = 'BLAKE';
```

Returns the following result.

```
    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
--------- ---------- --------- --------- --------- --------- ---------- ---------
     7698 BLAKE      MANAGER        7839 1981-05-0      2850                    30

UPDATE EMP SET SAL = SAL + TO_NUMBER('100.52','9,999.99') WHERE ENAME = 'BLAKE';
```

Returns the following result.

```
1 row updated.

SELECT * FROM EMP WHERE ENAME = 'BLAKE';
```

Returns the following result.

```
    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
--------- ---------- --------- --------- --------- --------- ---------- ---------
     7698 BLAKE      MANAGER        7839 1981-05-0   2950.52                    30
```

3.3.66 TRANSLATE

Syntax

```
TRANSLATE(char, from, to)
```

Purpose

Returns *char* with all occurrences of each character in *from* replaced by its corresponding character in *to*, where *char*, *from*, and *to* are string arguments.

Usage Notes

- 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 from the return value all characters in *from*. TRANSLATE interprets the empty string as null, and if this function has a null argument, it returns null.

Example

```
SELECT TRANSLATE('2KRW229', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'9999999999XXXXXXXXXXXXXXXXXXXXXXXXXX') "Licence" FROM DUAL;
```

Returns the following result.

```
Licence
-------
9XXX999
```

3.3.67 TRIM

Syntax

```
TRIM( [[<trim_spec >] char ]
    FROM ] string )
```

If <*trim_spec*> is omitted, then BOTH is implied. If *char* is omitted, then a space character is implied as listed in Table 3–14.

Table 3–14 Arguments Used with the TRIM Function

Argument	Description
<*trim_spec*>	a specification: LEADING, TRAILING, or BOTH
char	a single character
string	the target string to be trimmed

Purpose

Removes leading and/or trailing blanks (or other characters) from a string.

Example

```
SELECT TRIM ('OLD' FROM 'OLDMAN') FROM DUAL;
```

Returns the following result.

```
TRIM('
------
MAN
```

3.3.68 TRUNC

Syntax with Numeric Arguments

```
TRUNC(n [, m])
```

Syntax with Date Arguments

```
TRUNC(d [, fmt])
```

Purpose with Numeric Arguments

Returns *n* truncated to *m* decimal places, where *m* and *n* are numeric arguments. If *m* is omitted, truncates to 0 places. If *m* is negative, truncates (makes zero) *m* digits to the left of the decimal point.

Purpose with Date Arguments

Returns the date *d* with its time portion truncated to the time unit specified by the format model *fmt*. If you omit *fmt*, then *d* is truncated to the nearest day.

Usage Notes

The format models to be used with the TRUNC (and ROUND) date function, and the units to which it rounds dates are listed in Table 3–15. The default model, DD, returns the date rounded to the day with a time of midnight.

Table 3–15 Arguments Used with the TRUNC Function

Format Model	Rounding Unit
CC or SCC	Century
YYYY, SYYYY, YEAR, SYEAR, YYY, YY, Y	Year (rounds up on July 1)
IYYY, IYY, IY, I	ISO Year
Q	Quarter (rounds up in 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

Example 1

```
SELECT TRUNC(TO_DATE('27-OCT-92', 'DD-MON-YY'), 'YEAR') "First Of The Year"
FROM DUAL;
```

Returns the following result.

```
First Of T
----------
1992-01-01
```

Example 2

```
SELECT TRUNC(15.79,1) "Truncate" FROM DUAL;
```

Returns the following result.

```
 Truncate
---------
     15.7
```

Example 3

```
SELECT TRUNC(15.79,-1) "Truncate" FROM DUAL;
```

Returns the following result.

```
 Truncate
---------
       10
```

3.3.69 UPPER

Syntax

```
UPPER(char)
```

Purpose

Returns the string argument *char* with all its letters converted to uppercase. The return value has the same datatype as *char*.

Example

```
SELECT UPPER('Carol') FROM DUAL;
```

Returns the following result.

```
UPPER
-----
CAROL
```

ODBC Function

```
{fn UCASE (char)}
```

3.3.70 USER

Syntax

```
USER
```

Purpose

Returns the current schema name as a character string.

Example 1

```
SELECT USER "User" FROM DUAL;
```

Returns the following result.

```
User
--------
SYSTEM
```

Example 2

```
SELECT {fn USER()} FROM DUAL;
```

Returns the following result.

```
{FNUSER()}
------------------------------
SYSTEM
```

ODBC Function

```
{ fn USER()}
```

3.3.71 VARIANCE

Syntax

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

Purpose

Returns variance of *x*, a number. Oracle Lite calculates the variance of *x* using this formula.

xi is one of the elements of *x*.

n is the number of elements in the set *x*. If *n* is 1, the variance is defined to be 0.

Example

```
SELECT VARIANCE(sal) "Variance" FROM emp;
```

Returns the following result.

```
 Variance
---------
1398313.9
```

3.3.72 WEEK

Syntax

```
{ fn WEEK ( <value_expression> ) }
```

Purpose

Returns the week of the year as an integer using arguments listed in Table 3–16.

Table 3–16 Arguments Used with the WEEK Function

Argument	Description
<value_expression>	A date on which the week is computed. The result is between 1 and 53.

Example 1

```
SELECT {fn WEEK({fn CURDATE()})} FROM DUAL;
```

Returns the following result.

```
{FNWEEK({FNCURDATE()})}
-----------------------
                     16
```

Example2

```
SELECT {fn week('1999-06-15')} FROM DUAL;
```

Returns the following.

```
EK('1999-06-15')}
-----------------
               25
```

3.3.73 YEAR

Syntax

```
YEAR (date_exp)
```

Purpose

Returns the `YEAR` as an integer.

Example 1

```
SELECT {FN YEAR ('06-15-1966')} FROM DUAL;
```

Returns the following result.

```
{FNYEAR('06-15-1966')}
----------------------
                  1966
```

Example 2

```
SELECT {fn YEAR({fn CURDATE()})} FROM DUAL;
```

Returns the following result.

```
{FNYEAR({FNCURDATE()})}
-----------------------
                   1999
```

4

SQL Commands

This document discusses SQL commands used by Oracle Database Lite. Topics include:

4.1 SQL Command Types

The following lists the different types of SQL commands including clauses and pseudocolumns. An explanation of each SQL command, clause, and pseudocolumn is provided in "SQL Commands Overview".

SQL Commands

Table 4–1 Data Definition Language (DDL) Commands

DDL	DDL	DDL
ALTER SEQUENCE	CREATE PROCEDURE	DROP INDEX
ALTER SESSION	CREATE SCHEMA	DROP JAVA
ALTER TABLE	CREATE SEQUENCE	DROP PROCEDURE
ALTER TRIGGER	CREATE SYNONYM	DROP SCHEMA
ALTER USER	GRANT	DROP SEQUENCE
ALTER VIEW	REVOKE	DROP SYNONYM
CREATE DATABASE	CREATE TABLE	DROP TABLE
CREATE FUNCTION	CREATE TRIGGER	DROP TRIGGER
CREATE GLOBAL TEMPORARY TABLE	CREATE USER	DROP USER
CREATE INDEX	CREATE VIEW	DROP VIEW
CREATE JAVA	DROP FUNCTION	TRUNCATE TABLE

Table 4–2 Data Manipulation Language (DML)

DML	DML
DELETE	SELECT
EXPLAIN PLAN	subquery::=

Table 4–2 (Cont.) Data Manipulation Language (DML)

DML	DML
INSERT	UPDATE

Table 4–3 Transaction Control Commands

Command	Command
COMMIT	SAVEPOINT
ROLLBACK	SET TRANSACTION

Table 4–4 Clauses

Clause	Clause
CONSTRAINT clause	DROP clause

Table 4–5 Pseudocolumns

Pseudocolumns	Pseudocolumns
CURRVAL and NEXTVAL pseudocolumns	OL__ROW_STATUS pseudocolumn
ROWNUM pseudocolumn	ROWID pseudocolumn
LEVEL pseudocolumn	

4.2 SQL Commands Overview

Oracle Database Lite uses several different types of SQL commands. This section discusses the different types of SQL commands.

4.2.1 Data Definition Language (DDL) Commands

Data definition language (DDL) commands enable you to perform the following tasks.

- Create, alter, and drop schema objects
- Grant and revoke privileges and roles
- 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.

4.2.2 Data Manipulation Language (DML) Commands

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

4.2.3 Transaction Control Commands

Transaction control commands manage changes made by DML commands.

4.2.4 Clauses

Clauses are subsets of commands that modify the command.

4.2.5 Pseudocolumns

Pseudocolumns are values generated from commands 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 SQL-92.

4.2.6 BNF Notation Conventions

The syntax diagrams in this document use a variation of Backus-Nauer Form (BNF), a convention used to show syntax in many programming languages. Emphasis and symbols have the following meaning in this version of BNF syntax.

- Keywords are shown in `UPPERCASE`.
- Placeholders for which you must substitute an actual value are shown in lowercase. These can include clauses and other expressions.
- Vertical (|) bars separate multiple choices. They indicate "or".
- Parentheses and other punctuation enclosed in quotes must be typed as shown, for example "(".
- Square brackets ([]) are not typed. They indicate that the enclosed syntax is optional.
- Curly braces ({}) usually are not typed. They indicate that you must specify one of the enclosed choices. (The choices are separated by vertical bars.)
- Loops or repetitions are indicated by a second, bracketed appearance of the term, set of terms, or expression, followed by ellipsis points. The brackets indicate that the repetition is optional (all repetitions are optional). The ellipsis points indicate that multiple repetitions are allowed. The bracketed appearance of the term begins with a comma if the repetitions are comma delimited.
- All other punctuation (quotation marks, commas, semicolons, and so on) must be typed as shown.

4.3 SQL Commands Alphabetical Listing

This section lists Oracle Database Lite SQL commands, clauses, and pseudocolumns in alphabetical order and discusses each. This discussion includes the following.

- Syntax
- BNF Notation
- Purpose
- Prerequisites
- Argument and Descriptions
- Usage Notes
- Examples
- Related Topics
- ODBC Functionality (where relevant)

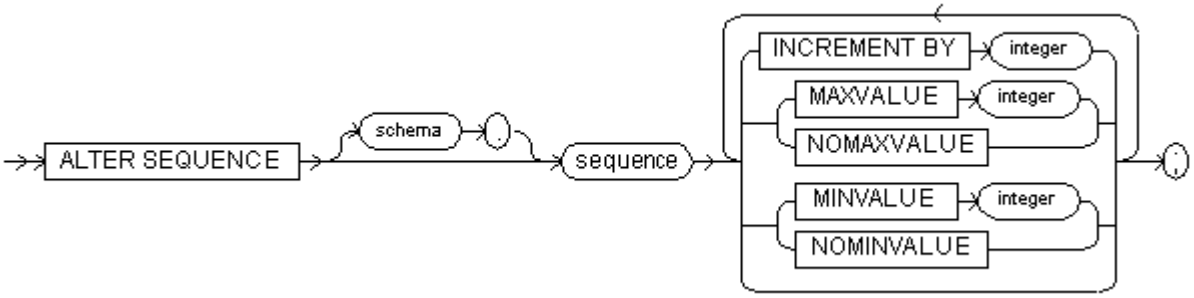
Note: All examples refer to sample database objects supplied with Oracle Database Lite. Some DDL examples may alter the structure and data of the sample database objects. To avoid altering the sample database objects, use the `ROLLBACK` command after each DDL example that you try in the database.

4.3.1 ALTER SEQUENCE

Syntax

The syntax for the `ALTER SEQUENCE` command is displayed in Figure 4–1.

Figure 4–1 The ALTER SEQUENCE Command



BNF Notation

```
ALTER SEQUENCE [schema .] sequence
  [(INCREMENT BY "integer"
    | (MAXVALUE "integer" | NOMAXVALUE)
    | (MINVALUE "integer" | NOMINVALUE)
  ]
;
```

Prerequisite

The sequence must be in your own schema.

Purpose

Changes a sequence in one of the following ways.

- Changes the increment between future sequence values.
- Sets or eliminates the minimum or maximum value.

The arguments for the `ALTER SEQUENCE` command are listed in Table 4–6.

Table 4–6 Arguments Used with the ALTER SEQUENCE Command

Argument	Description
schema	The name of the schema to contain the sequence. If you omit *schema*, Oracle Database Lite alters the sequence in your own schema.
sequence	The name of the sequence to be altered.

Table 4–6 (Cont.) Arguments Used with the ALTER SEQUENCE Command

Argument	Description
INCREMENT BY	Specifies the interval between sequence numbers. Can be any positive or negative integer, but cannot be 0. If negative, then the sequence descends. If positive, the sequence ascends. This value can have 10 or fewer digits. The absolute of this value must be less than the difference of `MAXVALUE` and `MINVALUE`. If you omit the `INCREMENT BY` clause, the default is 1.
MAXVALUE	Specifies the maximum value the sequence can generate. This integer value can have 10 or fewer digits. `MAXVALUE` must be greater than `MINVALUE`.
NOMAXVALUE	Specifies a maximum value of 2147483647 for an ascending sequence or –1 for a descending sequence.
MINVALUE	Specifies the minimum value that the sequence can generate. This integer value can have 10 or fewer digits. `MINVALUE` must be less than `MAXVALUE`.
NOMINVALUE	Specifies a minimum value of 1 for an ascending sequence or –2147483647 for a descending sequence.

Usage Notes

- To restart a sequence at a different number, you must drop and recreate the sequence. Only future sequence numbers are affected by the `ALTER SEQUENCE` command.
- Oracle Database Lite performs some validations. For example, you cannot specify a new `MAX VALUE` that is less than the current sequence number, or a new `MINVALUE` that is greater than the current sequence number.

Example

This statement sets a new maximum value for the ESEQ sequence.

```
ALTER SEQUENCE eseq MAXVALUE 1500
```

ODBC 2.0

Although the `ALTER SEQUENCE` command is not part of ODBC SQL; ODBC passes the command through to your database.

Related Topics

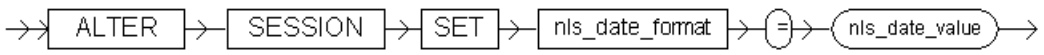
CREATE SEQUENCE, DROP SEQUENCE

4.3.2 ALTER SESSION

Syntax

The syntax for the `ALTER SESSION` command is displayed in Figure 4–2.

Figure 4–2 The ALTER SESSION Command



BNF Notation

```
ALTER SESSION SET nls_date_format = nls_date_value ;
```

Prerequisite

None

Purpose

To specify or modify any of the conditions or parameters that affect your connection to the database. Oracle Database Lite only enables you to use the `SET` clause of this command to specify or modify the NLS date format. The statement stays in effect until you disconnect from the database.

The arguments for the `ALTER SESSION` command are listed in Table 4–7.

Table 4–7 Arguments Used with the ALTER SESSION Command

Argument	Description
parameter_name	With Oracle Lite, the `ALTER SESSION` command has only one parameter name: `NLS_DATE_FORMAT`.
parameter_value	The NLS date format. For example: `YYYY MM DD HH24:MI:SS`.

Example

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'YYYY MM DD HH24:MI:SS';
```

Oracle Lite uses the new default date format.

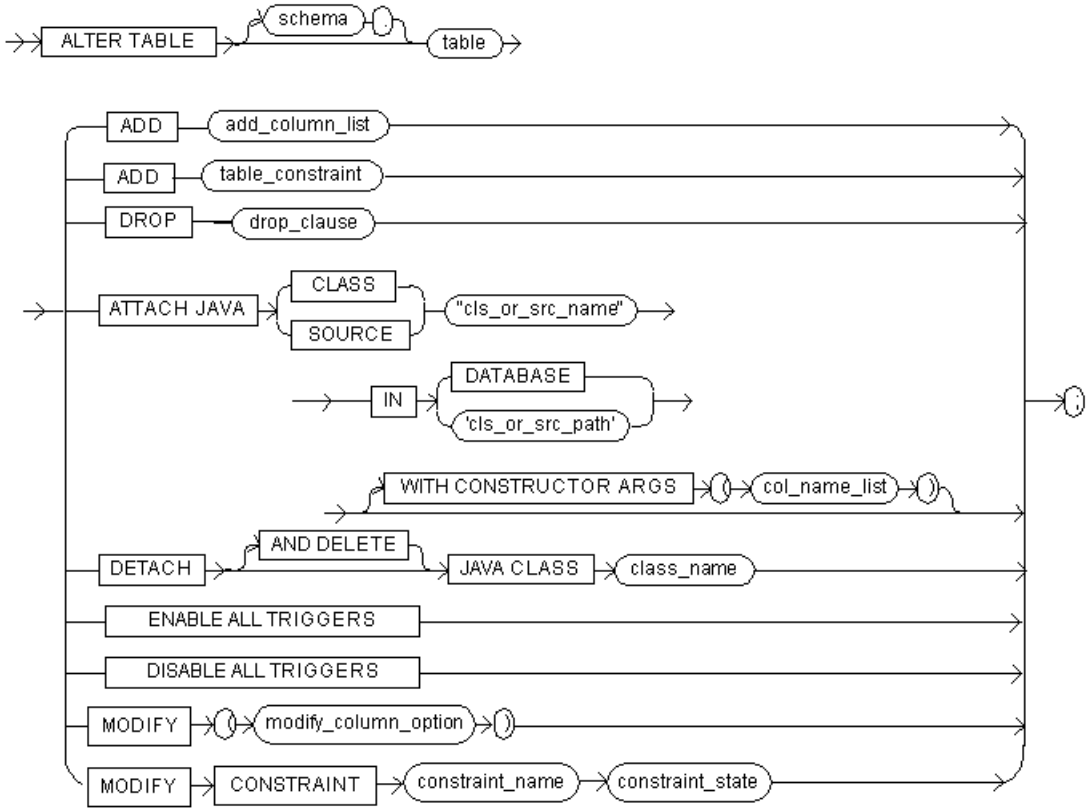
```
SELECT TO_CHAR(SYSDATE) Today FROM DUAL;

TODAY
-------------------
1997 08 12 14:25:56
```

4.3.3 ALTER TABLE

Syntax

The syntax for `ALTER TABLE` is displayed in Figure 4–3.

Figure 4–3 The ALTER TABLE Command



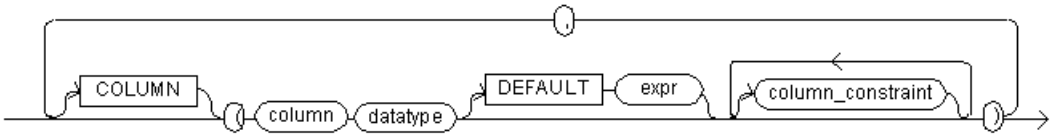
BNF Notation

```
ALTER TABLE [schema .] table
{
  ADD add_column_list
 |ADD table_constraint
 |DROP drop_clause
 |ATTACH JAVA {CLASS | SOURCE} cls_or_src_name
   IN {DATABASE | cls_or_src_path}
    [WITH CONSTRUCTOR ARGUMENTS "(" col_name_list ")"
 |DETACH [AND DELETE] JAVA CLASS class_name
 |ENABLE ALL TRIGGERS
 |DISABLE ALL TRIGGERS
 |MODIFY "(" modify_column_option")"
 |MODIFY CONSTRAINT constraint_name constraint_state
}
;
```

add_column_list::=

The syntax for the `add_column_list` expression is displayed in Figure 4–4.

Figure 4–4 The add_column_list Expression



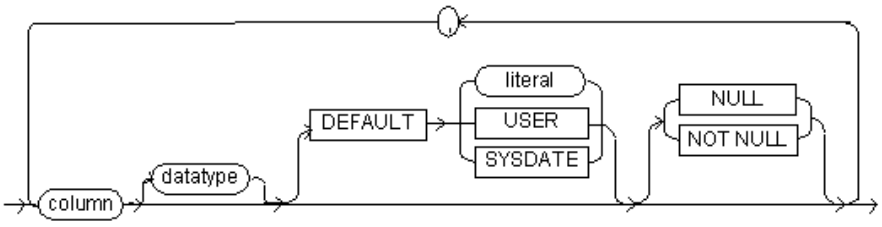
BNF Notation

```
[COLUMN] "("column datatype [DEFAULT expr] [column_constraint]
[, column_constraint]...")" [, [COLUMN] "("column datatype [DEFAULT expr]
[column_constraint] [, column_constraint]...")"]...
```

modify_column_option::=

The syntax for `modify_column_option` expression is displayed in Figure 4–5.

Figure 4–5 The modify_column_option Expression



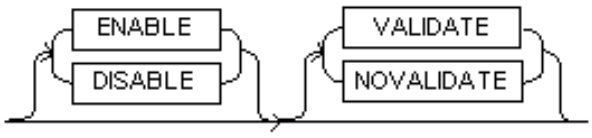
BNF Notation

```
column [datatype] [DEFAULT { literal | USER | SYSDATE }] [ NULL | NOT NULL ]
      [,  column [ [datatype] [DEFAULT { literal | USER | SYSDATE }]
       [ NULL | NOT NULL ] ] ]...
```

constraint_state::=

The syntax for `constraint_state` expression is displayed in Figure 4–6.

Figure 4–6 The constraint_state Expression



BNF Notation

```
([ENABLE | DISABLE] [VALIDATE | NOVALIDATE])
```

Prerequisite

The table must be in your own schema. You must be logged into the database as `SYSTEM` or as a user with DBA/DDL privileges.

Purpose

Changes the definition of a table in one of the following ways:

- Adds a column or integrity constraint
- Drops a column or integrity constraint
- Attaches a Java class
- Detaches a Java class
- Add, or change default value of a column
- Change datatype or size of a column
- Disable or enable a constraint
- Change nullity property of a column

The arguments for the `ALTER TABLE` command are listed in Table 4–8.

Table 4–8 Arguments Used with the ALTER TABLE Command

Argument	Description
schema	The name of the schema, which is a character string of up to 128 characters. The schema name must be different from any user names since each user name comes with a default schema with the same name. If you create a schema with the same name as a user name, Oracle Lite returns an error. See "CREATE USER" for more information.
table	The name of a database table.
ADD	Specifies that a column or integrity constraint is added to the database table.
DROP	Specifies that a column or integrity constraint is dropped from the database table.
column	The name of a database column.
datatype	The datatype of the database column.
DEFAULT	Specifies a default value *expr* (expression) for the new column. It can be one of the following: ■ `DEFAULT NULL`, `DEFAULT USER` (the user name when the table is created), `DEFAULT` literal ■ `ODBC FUNCTIONS` - `TIMESTAMPADD`, `TIMESTAMPDIFF`, `DATABASE`, `USER` ■ `SQL FUNCTIONS` - `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`, `SYSDATE` For more information about expressions, see Section 1.8, "Specifying Expressions".
expr	A valid expression. Expressions are evaluated when `ALTER TABLE` is executed, not when a row is inserted with a default value. For more information, see Section 1.8, "Specifying Expressions".
column_constraint	A column integrity constraint. For more information, see CONSTRAINT clause. You cannot add a column with a not null constraint to a table that already contains data.
table_constraint	A table integrity constraint. For more information, see "CONSTRAINT clause".
drop_clause	An integrity constraint to be dropped. For more information, see "DROP clause".
`ATTACH JAVA`	Attaches a Java class or source file to the database table.

Table 4–8 (Cont.) Arguments Used with the ALTER TABLE Command

Argument	Description
IN	Indicates that the Java class or source file must be attached in either a database, Java class, or source path.
DATABASE	The database in which you attach the Java class or source path.
DETACH	Detaches a Java class from the database table.
CLASS	Specifies a Java class.
SOURCE	Specifies a Java source file.
cls_or_src_name	A fully qualified Java class or source file name.
cls_or_src_path	The directory containing the specified Java class or source file.
WITH CONSTRUCTOR ARGS	Specifies attributes of the class to be used as arguments to the Java constructor.
col_name_list	List of columns (attributes) in the database table.
AND DELETE	Deletes the Java class from the database.
class_name	The name of a fully qualified Java class.
ENABLE ALL TRIGGERS	Enables all triggers associated with the table. The triggers are fired whenever their triggering condition is satisfied. To enable a single trigger, use the ENABLE clause of ALTER TRIGGER. See ALTER TRIGGER.
DISABLE ALL TRIGGERS	Disables all triggers associated with the table. A disabled trigger is not fired even if the triggering condition is satisfied. To disable a single trigger, use the DISABLE clause of ALTER TRIGGER. See ALTER TRIGGER.
MODIFY	This specifies a new default for an existing column. Oracle Database Lite 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.
modify_column_option	This modifies the definition of an existing column. Any of the optional parts of the column definition, *datatype*, *default value* (*literal*, USER, or SYSDATE) or *column constraint* state (NULL, NOT NULL) which are omitted remain unchanged. Existing datatypes can be changed to a new datatype as long as the existing data is such that the data conversion does not produce any conversion errors. Increasing the size of a varchar column whose existing size is greater than 15 characters does not require any data conversion. All other changes require a data conversion step. Each column is converted individually. Each datatype change involves a rewrite of all objects and creation of all dependent indexes. A column undergoing datatype alteration which is part of an index created using the KEY COLUMNS clause, may cause the ALTER TABLE MODIFY command to fail because the index recreation is unable to reestablish the KEY COLUMNS option. An index created using KEY COLUMNS, should be dropped before modifying the column.
CONSTRAINT	Modifies the state of an existing *constraint*. ENABLE specifies that the constraint is applied to all new data in the table. Before a referential integrity constraint can be enabled, its referenced constraint must be enabled.

Table 4–8 (Cont.) Arguments Used with the ALTER TABLE Command

Argument	Description
ENABLE VALIDATE	This setting specifies that all existing data complies with the constraint. An enabled validated constraint guarantees that all data is and continues to be valid. If a user places a primary key constraint in ENABLE VALIDATE mode, validation ensures that primary key columns contain no nulls. If VALIDATE or NOVALIDATE are omitted, the default is VALIDATE.
ENABLE NOVALIDATE	This setting ensures that all new DML operations on the constrained data comply with the constraint, but does not ensure that existing data in the table complies with the constraint. Enabling a primary key constraint automatically creates a primary index to enforce the constraint. This index is converted to an ordinary index if the primary key constraint is subsequently disabled. If the constraint is subsequently re-enabled, the index is checked for any primary key constraints and if no violations are detected, is restored to primary key status.
DISABLE VALIDATE	This setting disables the constraint and converts the index on the primary key constraint to an ordinary index, but keeps the constraint valid. No DML statements are allowed on the table through the SQLRT engine but you may be able to perform a DML statement through Oracle Database Lite Java Access Classes (JAC). If VALIDATE or NOVALIDATE are omitted, the default is NOVALIDATE.
DISABLE NOVALIDATE	This setting signifies that Oracle Database Lite makes no effort to maintain the constraint (because it is disabled) and cannot guarantee that the constraint is true (because it is not validated). A primary key constraint index is downgraded to an ordinary index. You cannot drop a table with a primary key that is referenced by a foreign key even if the foreign key constraint is in the DISABLE NOVALIDATE state.

Usage Notes

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 when a default value is also specified, regardless of whether or not the table is empty.

If VALIDATE or NOVALIDATE are omitted from the ENABLE argument, the default is NOVALIDATE.

If VALIDATE or NOVALIDATE are omitted from the DISABLE argument, the default is NOVALIDATE.

The nullity constraint is the only integrity constraint that can be added to an existing column using the MODIFY clause with the column constraint syntax. NOT NULL can be added only if the column contains no nulls. A NULL can be added provided the column is not a component of a primary key constraint.

Example

The following statement adds the columns THRIFTPLAN and LOANCODE to the EMP table. THRIFTPLAN has a datatype, NUMBER, with a maximum of seven digits and two

decimal places. LOANCODE has a datatype, CHAR, with a size of one and a NOT NULL integrity constraint:

```
ALTER TABLE emp
ADD (thriftplan NUMBER(7,2),
loancode CHAR(1));
```

Related Topics

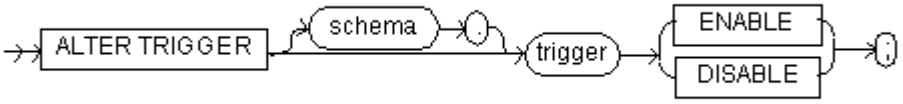
CONSTRAINT clause, CREATE TABLE, CREATE VIEW

4.3.4 ALTER TRIGGER

Syntax

The syntax for the ALTER TRIGGER command is displayed in Figure 4–7.

Figure 4–7 The ALTER TRIGGER Command



BNF Notation

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

Prerequisites

To alter a trigger you must have the DBA/DDL privilege.

Purpose

To enable or disable a database trigger. For information on creating a trigger, see CREATE TRIGGER. For information on dropping a trigger, see DROP TRIGGER.

> **Note:** This statement does not change the declaration or definition of an existing trigger. To redeclare or redefine a trigger, use the CREATE TRIGGER statement with OR REPLACE.

The arguments for the ALTER TRIGGER command are listed in Table 4–9.

Table 4–9 Parameters of the ALTER TRIGGER Command

Parameter	Description
schema	The schema containing the trigger. If you omit schema, Oracle Database Lite assumes the trigger is in your own schema.
trigger	The name of the trigger to be altered.
ENABLE	Enables the trigger. You can also use the ENABLE ALL TRIGGERS clause of ALTER TABLE to enable all triggers associated with a table. See ALTER TABLE.
DISABLE	Disables the trigger. You can also use the DISABLE ALL TRIGGERS clause of ALTER TABLE to disable all triggers associated with a table. See ALTER TABLE.

Examples

Consider a trigger named REORDER created on the INVENTORY table. The trigger 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, Oracle Database Lite enables it automatically. You can subsequently disable the trigger with the following statement.

```
ALTER TRIGGER reorder DISABLE;
```

When the trigger is disabled, Oracle Database Lite 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 re-enable the trigger, Oracle Database Lite fires the trigger whenever a part's inventory falls below its reorder point as a result of an UPDATE statement. It is possible that a part's inventory falls below its reorder point while the trigger was disabled. In that case, when you reenable the trigger, Oracle Database Lite does not automatically fire the trigger for this part until another transaction further reduces the inventory.

Related Topics

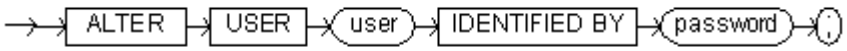
CREATE TRIGGER

4.3.5 ALTER USER

Syntax

The syntax for ALTER USER is displayed in Figure 4–8.

Figure 4–8 The ALTER USER Command



BNF Notation

```
ALTER USER user IDENTIFIED BY password ;
```

Prerequisite

You can change your user password in the database if you meet one of the following conditions.

- You are connected to the database as that user.
- You are connected to the database as SYSTEM or as a user with DBA/DDL or ADMIN privileges.
- You are granted the UNRESOLVED XREF TO ADMIN or UNRESOLVED XREF TO DBA/DDL role.

Purpose

Changes a database user password.

The arguments for the ALTER USER command are listed in Table 4–10.

Table 4–10 Arguments Used with the ALTER USER Command

Argument	Description
user	The user to be altered. Here, user is a unique user name with no more than 30 characters, beginning with one character. The first character in user cannot be a blank space.
`IDENTIFIED BY`	Indicates how Oracle Database Lite permits user access.
password	Specifies a new password for the user which is a name of up to 128 characters. The password does not appear in quotes and is not case-sensitive.

Example

The following example creates a user named `todd` identified by the password, `tiger`. It then changes the user's password to `lion`.

```
CREATE USER todd IDENTIFIED BY tiger;

ALTER USER todd IDENTIFIED BY lion;
```

Related Topics

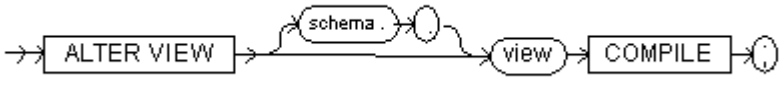
CREATE USER, DROP USER

4.3.6 ALTER VIEW

Syntax

The syntax for the `ALTER VIEW` command is displayed in Figure 4–9.

Figure 4–9 The ALTER VIEW Command



BNF Notation

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

Prerequisite

The view must be in your own schema. You must be logged into the database as `SYSTEM` or as a user with DBA/DDL privileges.

Purpose

Recompiles a view.

The arguments for the `ALTER VIEW` command are listed in Table 4–11.

Table 4–11 Arguments Used with the ALTER VIEW Command

Argument	Description
schema	The schema to contain the view. If you omit schema, Oracle Database Lite alters the view in your own schema.
view	The name of the view to be recompiled.

Table 4–11 (Cont.) Arguments Used with the ALTER VIEW Command

Argument	Description
COMPILE	Causes Oracle Lite to recompile the view. The `COMPILE` keyword is required.

Usage Notes

You can use `ALTER VIEW` to explicitly recompile a view that is invalid. Explicit recompilation enables you to locate recompilation errors before run-time. 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, Oracle Database Lite recompiles the view regardless of whether it is valid or invalid. Oracle Database Lite also invalidates any local objects that depend on the view.

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

The following code demonstrates the `ALTER VIEW` SQL command. The `COMPILE` keyword is required.

```
ALTER VIEW customer_view COMPILE;
```

Related Topics

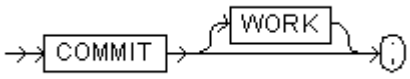
CREATE VIEW, DROP VIEW

4.3.7 COMMIT

Syntax

The syntax for `COMMIT` is displayed in Figure 4–10.

Figure 4–10 The COMMIT Command



BNF Notation

```
COMMIT [WORK] ;
```

Prerequisite

None

Purpose

Ends your current transaction, making permanent to the database all its changes.

The arguments for the `COMMIT` command are listed in Table 4–12.

Table 4–12 Arguments Used with the Commit Command

Argument	Description
WORK	An optional argument with no effect. WORK is supported only for compliance with standard SQL. The statements COMMIT and COMMIT WORK are equivalent.

Usage Notes

Oracle Database Lite does not autocommit any DDL statements except for CREATE DATABASE. You must commit your current transaction to make permanent all of its changes to the database.

Example

The following code demonstrates the COMMIT command. This example inserts a row into the DEPT table and commits the change. The WORK argument is optional.

```
INSERT INTO dept VALUES (50, 'Marketing', 'TAMPA');

COMMIT;
```

ODBC 2.0

Although the COMMIT command is not part of the ODBC SQL syntax, ODBC passes the command through to your database.

An ODBC program typically uses the API call SQLTransact() with the SQL_COMMIT flag.

Related Topics

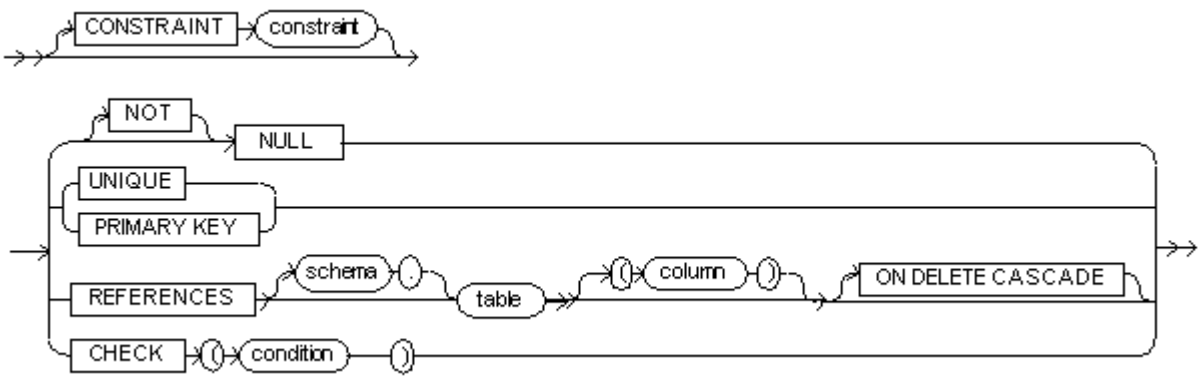
ROLLBACK

4.3.8 CONSTRAINT clause

Syntax

The syntax for the COLUMN CONSTRAINT clause is displayed in Figure 4–11.

Figure 4–11 The COLUMN_CONSTRAINT Clause



BNF Notation

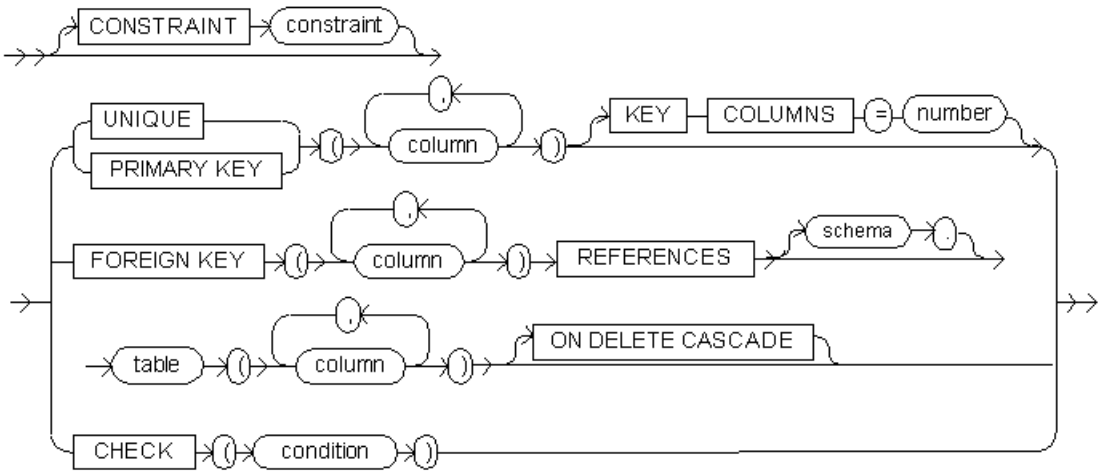
```
[CONSTRAINT constraint]
{ [NOT] NULL
```

```
    | {UNIQUE | PRIMARY KEY}
    | REFERENCES [schema .] table ["("column")"] [ON DELETE CASCADE]
    | CHECK "(" condition ")"
 }
```

Syntax

The syntax for the `TABLE CONSTRAINT` clause is displayed in Figure 4–12.

Figure 4–12 The TABLE CONSTRAINT Clause



BNF Notation

```
[CONSTRAINT constraint]
{

 { UNIQUE | PRIMARY KEY } "("column [, column] ..")" [ KEY COLUMNS = number ]
 | FOREIGN KEY "("column [, column] ...")" REFERENCES [ schema .] table
"("column [, column] ...")" [ON DELETE CASCADE]
 | CHECK "("condition")"
}
```

Prerequisite

`CONSTRAINT` clauses can appear in both the CREATE TABLE and ALTER TABLE commands. To define an integrity constraint, you must be logged into the database as `SYSTEM` or as a user with DBA/DDL privileges. Oracle Database Lite only has integrity constraints.

Purpose

Defines an integrity constraint.

The arguments for the `CONSTRAINT` clause are listed in Table 4–13.

Table 4–13 Arguments Used with the Constraint Clause

Argument	Description
CONSTRAINT	Identifies the integrity constraint named by the constraint argument. Oracle Database Lite stores the constraint's name and definition in the data dictionary. If you omit the CONSTRAINT keyword, Oracle Database Lite generates a name with this form: POL_SYS_CONS*n*, where *n* is an integer that makes the name unique within the database.
constraint	The name of the constraint being added.
NULL	Specifies that a column can contain null values.
NOT NULL	Specifies that a column cannot contain null values. By default, a column can contain nulls.
UNIQUE	Designates a column, or a combination of columns, as a unique key.
PRIMARY KEY	Designates a column, or a combination of columns, as the table's primary key.
KEY COLUMNS =	This specifies how many columns should be used to create the index. This clause is useful when an index is needed on a large number of columns, since it reduces the size of the index. Query performance may suffer when multiple rows qualify as prefix columns of an index key as given by the KEY COLUMNS value, since the database looks up all qualifying rows to find the matching row(s).
number	An integer which specifies the number of KEY COLUMNS.
FOREIGN KEY	Designates a column, or a combination of columns in the child table, as the foreign key in a referential integrity constraint.
schema	The name of the schema, which is a character string up to 128 characters. The schema name must be different from any user names since each user name comes with a default schema with the same name. If you create a schema with the same name as a user name, Oracle Database Lite returns an error. See CREATE USER for more information.
REFERENCES	Identifies the primary key or unique key of the parent table that is referenced by a foreign key in a referential integrity constraint.
table	Specifies the table on which the constraint is placed. If you specify only *table* and omit the *column* argument, the foreign key automatically references the primary key of the table.
column	Specifies the column of the table on which the constraint is placed.
ON DELETE CASCADE	Specifies that Oracle Database Lite maintains referential integrity by automatically removing dependent foreign key values when you remove a referenced primary key or unique key value.
CHECK	Specifies that a condition be checked for each row in the table. Oracle Database Lite only supports the following operators and functions in CHECK conditions. + - / * = ! = < > < = > = IS NULL, LIKE, BETWEEN, TO_CHAR TO_NUMBER, TO_DATE, TRANSLATE
condition	Specifies the condition that each row in the table must satisfy. For more information about creating a valid condition, see Section 1.7, "Specifying SQL Conditions".

Example

The following example creates a table T, with columns A and B. The example uses the `PRIMARY KEY` constraint clause to make column A the table's primary key.

```
CREATE TABLE T (A CHAR(20) PRIMARY KEY, B CHAR(20));
```

Related Topics

ALTER TABLE, CREATE TABLE

4.3.9 CREATE DATABASE

Syntax

The syntax for `CREATE DATABASE` is displayed in Figure 4–13.

Figure 4–13 The CREATE DATABASE Command

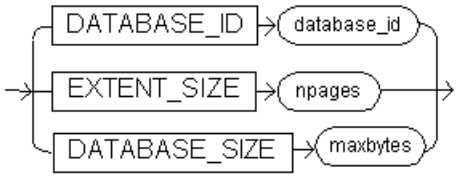
BNF Notation

```
CREATE DATABASE database database_parameter [, database_parameter]...;
```

database_parameters::=

The syntax for the `database_parameters` expression is displayed in Figure 4–14.

Figure 4–14 The database_parameters Expression



BNF Notation

```
{|DATABASE_ID database_id
 |DATABASE_SIZE max_bytes
 |EXTENT_SIZE npages
}
;
```

Prerequisite

None

Purpose

Creates a database.

The arguments for the `CREATE DATABASE` command are listed in Table 4–14.

Table 4–14 Arguments Used with the CREATE DATABASE Command

Argument	Description
database	A data file name or full path name. Full path names must be enclosed in double quotation marks. If no path name is specified, the data file is created in the directory specified by the data source name (DSN) if connected through ODBC. If neither the full path name nor DSN are valid, the database is created under the current working directory. The length of *database* is limited by the operating system or file system. If a duplicate database name is used, an error occurs.
DATABASE_ID	An optional numeric identifier for the database.
database_id	A unique identifier for the database. Must be a unique number from 16 to 32765. If omitted, the default initial value is 64. The *database_id* parameter in the **POLITE.INI** file indicates the next available database ID. It is possible to create two databases with the same database ID; however, you cannot connect to both databases at the same time.
DATABASE_SIZE	The database size.
maxbytes	The maximum file size to which the database can grow. If omitted, the default value is 256M. The abbreviations K, M, and G may be used for kilobytes, megabytes, and gigabytes, respectively. If an abbreviation is not specified, the default is K. If specifying an abbreviation, you must use an integer value between 250 kilobytes and 4 gigabytes, for example, 256M, 1000K, or 2G.
EXTENT_SIZE	An incremental amount of pages in a database file. When a database runs out of pages in the current file, it extends the file by this number of pages.
npages	The number of 4K (kilobyte) pages which make up an extent (the minimum unit of allocation for a table). A number that is a multiple of 2 is required for *npages*. The default value is 4. If set to 0, Oracle Database Lite sets *npages* to the default value.

Usage Notes

The number of pages should be less than or equal to 64.

Keywords may be listed in any order.

Before you can run a newly created database, you must first configure its ODBC data source name (DSN) using the ODBC Administrator. See the Oracle Lite User's Guide for more information about creating a DSN or using the ODBC Administrator.

Unlike other DDL statements, Oracle Lite autocommits the `CREATE DATABASE` command. You cannot undo the `CREATE DATABASE` command with a `ROLLBACK` statement.

If the **POLITE.INI** parameter `NLS_SORT` has been set to enable one of the collation sequences, such as `FRENCH`, all databases are created with that collation sequence. The default is `BINARY`. For more information see the *Oracle Database Lite Developer's Guide*.

Example

To create the data file **LIN.ODB** in the directory C:\TMP with the **.ODB** file extension, use.

```
CREATE DATABASE "C:\TMP\LIN"
```

Related Topics

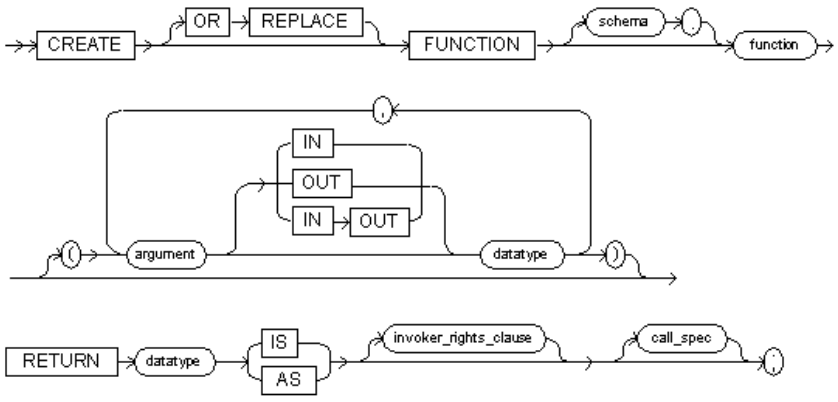
ROLLBACK

4.3.10 CREATE FUNCTION

Syntax

The syntax for `CREATE FUNCTION` is displayed in Figure 4–15.

Figure 4–15 The CREATE FUNCTION Command



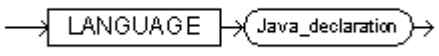
BNF Notation

```
CREATE [OR REPLACE] FUNCTION [schema .] function
["(" argument [ IN | OUT | IN OUT ] datatype
   [, argument [ IN | OUT | IN OUT ] datatype]...
 ")"
]

RETURN datatype { IS | AS } [ invoker_rights_clause] [call_spec]
;
```

call_spec::=

The syntax for the `call_spec` expression is displayed in Figure 4–16.

Figure 4–16 The call_spec Expression



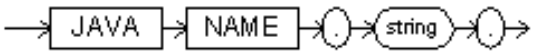
BNF Notation

```
LANGUAGE  Java_declaration
```

Java_declaration::=

The syntax for the `Java_declaration` expression is displayed in Figure 4–17.

Figure 4–17 The Java_declaration Expression



BNF Notation

```
JAVA NAME . string .
```

Prerequisite

To create a function in your own schema, you must be connected to the database as `SYSTEM` or you must have DBA/DDL privileges.

To invoke a call specification, you must have DBA/DDL privileges.

Purpose

To create a call specification for a stored function.

A *stored function* (also called a *user function*) is a Java stored procedure that returns a value. Stored functions are very similar to procedures, except that a procedure does not return a value to the environment in which it is called. For a general discussion of procedures and functions, see CREATE PROCEDURE. For examples of creating functions, see the CREATE FUNCTION examples.

A *call specification* declares a Java method so that it can be called from SQL. The call specification tells Oracle Database Lite which Java method to invoke when a call is made. It also tells Oracle Database Lite what type conversions to make for the arguments and return value.

The `CREATE FUNCTION` statement creates a function as a standalone schema object. For information on dropping a stand alone function, see DROP FUNCTION.

The arguments for the `CREATE FUNCTION` command are listed in Table 4–15.

Table 4–15 Arguments Used with the CREATE FUNCTION Command

Argument	Description
`OR REPLACE`	Recreates the function if it already exists. Use this clause to change the definition of an existing function without dropping, re-creating, and regranting object privileges previously granted on the function. Users who had previously been granted privileges on a redefined function can still access the function without being regranted the privileges. If any function-based indexes depend on the function, Oracle Database Lite marks the indexes `DISABLED`.
schema	The *schema* to contain the function. If you omit schema, Oracle Database Lite creates the function in your current schema.
function	The name of the function to create. See "Usage Notes".
argument	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 that the function sets the value of the argument.

Table 4–15 (Cont.) Arguments Used with the CREATE FUNCTION Command

Argument	Description
IN OUT	Specifies that a value for the argument can be supplied by you and may be set by the function. ■ Changes made either to this parameter or to another parameter may be visible immediately through both names if the same variable is passed to both. ■ If the function is exited with an unhandled exception, any assignment made to this parameter may be visible in the caller's variable. These effects may or may not occur on any particular call. You should use NOCOPY only when these effects do not matter.
datatype	The datatype of an argument. An argument can have any datatype supported by SQL. The datatype cannot specify a length, precision, or scale. Oracle Database Lite 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 SQL. The datatype cannot specify a length, precision, or scale. Oracle Database Lite derives the length, precision, or scale of the return value from the environment from which the function is called.
IS	Associates the SQL identifier with the Java method.
AS	Associates the SQL identifier with the Java method.
invoker_rights_clause	For compatibility with Oracle, Oracle Database Lite recognizes but does not enforce the *invoker_rights_clause*.
call_spec	Maps the Java method name, parameter types, and return type to their SQL counterparts.
LANGUAGE	Specifies the *call_spec* language. In Oracle database this can be C or Java. In Oracle Database Lite, this can only be Java.
java_declaration	Specifies the *call_spec* language. In Oracle database this can be C or Java. In Oracle Database Lite, this can only be Java.
JAVA NAME	The Java method name
string	Identifies the Java implementation of the method. For more information, see the *Oracle Database Lite Developer's Guide for Java*.

Usage Notes

User-defined functions cannot be used in situations that require an unchanging definition. You cannot use user-defined functions.

- In a CHECK constraint clause of a CREATE TABLE or ALTER TABLE statement.
- In a DEFAULT clause of a CREATE TABLE or ALTER TABLE statement.

In addition, when a function is called from within a query or DML statement, the function cannot.

- Have OUT or IN OUT parameters.
- Commit or roll back the current transaction, create or roll back to a savepoint, or alter the session or the system. DDL statements implicitly commit the current transaction, so a user-defined function cannot execute any DDL statements.

- Write to the database, if the function is being called from a SELECT statement. However, a function called from a subquery in a DML statement can write to the database.
- Write to the same table that is being modified by the statement from which the function is called, if the function is called from a DML statement.

Except for the restriction on OUT and IN OUT parameters, Oracle Database Lite enforces these restrictions not only for the function called directly from the SQL statement, but also for any functions that the function calls. Oracle Database Lite also enforces these restrictions on any functions called from the SQL statements executed by that function or any function it calls.

Example

The following example provides complete instructions for creating and testing a function.

1. Create and compile the following Java program and name it Employee.java.

```
public class Employee {
  public static String paySalary (float sal, float fica, float sttax,
     float ss_pct, float espp_pct) {
   float deduct_pct;
   float net_sal;

   /* compute take-home salary */
   deduct_pct = fica + sttax + ss_pct + espp_pct;
   net_sal = sal * deduct_pct;

   String returnstmt = "Net salary is " + net_sal;
   return returnstmt;
 } /*paySalary */
}
```

2. Load the Employee class into Oracle Database Lite. Once loaded, the Employee class methods become stored procedures in Oracle Database Lite.

```
CREATE JAVA CLASS USING BFILE ('C:\', 'Employee.class');
```

3. Since the employeeSalary method returns a value, publish it by using the CREATE FUNCTION statement.

```
CREATE FUNCTION
PAY_SALARY(
    sal float, fica float, sttax float, ss_pct float, espp_pct float)
    return varchar2
as language java name
'Employee.paySalary(float,float,float,float,float)return java.lang.String';
.
/
```

4. Select the PAY_SALARY stored procedure from dual:

```
SELECT PAY_SALARY(6000.00, 0.2, 0.0565, 0.0606, 0.1) from dual;
```

Returns the following result.

```
PAY_SALARY
----------------------------------------
Net Salary is 2502.6
```

Related Topics

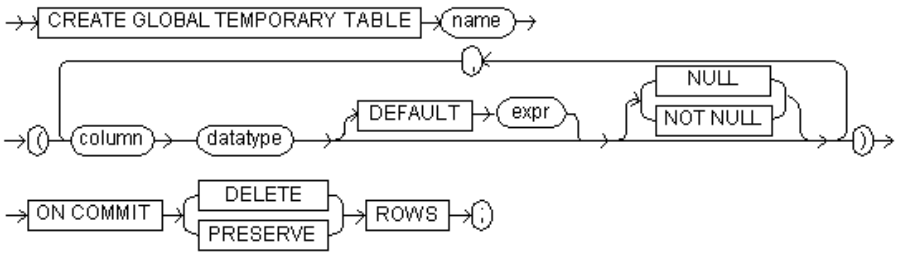
DROP FUNCTION

4.3.11 CREATE GLOBAL TEMPORARY TABLE

Syntax

The syntax for the `CREATE GLOBAL TEMPORARY TABLE` command is displayed in Figure 4–18.

Figure 4–18 The CREATE GLOBAL TEMPORARY TABLE Command



BNF Notation

```
CREATE GLOBAL TEMPORARY TABLE table
"("  column datatype [DEFAULT expr] [{ NULL | NOT NULL}]
  [,  column datatype [DEFAULT expr] [ {NULL | NOT NULL} ]... ")"
ON COMMIT {DELETE | PRESERVE } ROWS ;
```

Purpose

The `CREATE GLOBAL TEMPORARY TABLE` command creates a temporary table which can be transaction specific or session specific. For transaction-specific temporary tables, data exists for the duration of the transaction. For session-specific temporary table, data exists for the duration of the session. Data in a temporary table is private to the session. Each session can only view and modify its own data. On rollback of a transaction, all modifications made to the global temporary table are lost.

The arguments for the `CREATE GLOBAL TEMPORARY TABLE` command are listed in Table 4–16.

Table 4–16 Arguments Used with CREATE GLOBAL TEMPORARY TABLE

Argument	Description
name	An optionally qualified table name.
schema	A schema, which has the same name as the user who owns it. If omitted, the default schema name is used.
column	The name of a table column.
datatype	The datatype of the column. Cannot be used in subquery.

Table 4–16 (Cont.) Arguments Used with CREATE GLOBAL TEMPORARY TABLE

Argument	Description
DEFAULT	Specifies a default value *expr* (expression) for the new column. It can be one of the following: ▪ DEFAULT NULL, DEFAULT USER (the user name when the table is created), DEFAULT literal ▪ ODBC FUNCTIONS - TIMESTAMPADD, TIMESTAMPDIFF, DATABASE, USER ▪ SQL FUNCTIONS - CURRENT_DATE, CURRENT_TIME, CURRRENT_TIMESTAMP, SYSDATE For more information about expressions, see Section 1.7, "Specifying SQL Conditions".

Usage Notes

Temporary tables cannot be partitioned, organized into an index, or clustered.

You cannot specify any referential integrity (foreign key) constraints on temporary tables.

Examples

The following statement creates a temporary table FLIGHT_SCHEDULE for use in an automated airline reservation scheduling system. Each client has its own session and can store temporary schedules. The temporary schedules are deleted at the end of the session.

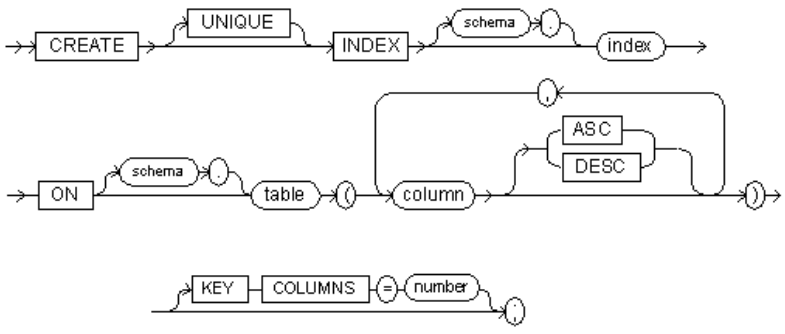
```
CREATE GLOBAL TEMPORARY TABLE flight_schedule (
startdate DATE,
enddate DATE,
cost NUMBER)
ON COMMIT PRESERVE ROWS;
```

4.3.12 CREATE INDEX

Syntax

The syntax for the CREATE INDEX command is displayed in Figure 4–19.

Figure 4–19 The CREATE INDEX Command



BNF Notation

```
CREATE [ UNIQUE ] INDEX [schema .] index ON
[schema .] table
```

```
"(" column [ ASC | DESC]
 [, column [ ASC | DESC]]...
 ")"
[ KEY COLUMNS=number]
;
```

Prerequisite

The table to be indexed must be in your own schema. You must be logged into the database as SYSTEM or as a user with DBA/DDL privileges.

Purpose

Creates an index on one or more columns of a table.

The arguments for the CREATE INDEX command are listed in Table 4–17.

Table 4–17 Arguments Used with the CREATE INDEX Command

Argument	Description
UNIQUE	Designates the specified column or combination of columns as a unique key.
schema	When it follows CREATE INDEX, this is the schema that contains the index. If you omit *schema*, Oracle Database Lite creates the index in your own schema. When used in the ON clause, the schema that contains the table for which the index is created.
index	The name of the index to create. You can create any number of indexes for a table, provided you do not use the same columns and column order for more than one index.
table	The name of the table for which the index is created. If you do not qualify table with a schema, Oracle Database Lite assumes that the table is contained in your own schema.
column	The name of a column in the table. A column of an index cannot be of the datatype LONG or LONG RAW.
ASC | DESC	Provided for DB2 compatibility only. Indexes are always created in ascending order.
KEY COLUMNS =	This specifies how many columns should be used to create the index. This clause is useful when an index is needed on a large number of columns, since it reduces the size of the index. Query performance may suffer when multiple rows qualify as prefix columns of an index key as given by the KEY COLUMNS value. The database looks up all qualifying rows to find the matching row(s).
number	An integer which specifies the number of KEY COLUMNS.

Usage Notes

You can use additional index creation options for tuning purposes. However, only use these options when necessary as they may degrade your database performance. See Appendix E, "Index Creation Options" for more information.

CREATE ANY INDEX can be used to create a index in another schema, but this requires the DBA/DDL role.

Example

The following example creates an index on the SAL column of the EMP table.

```
CREATE INDEX SAL_INDEX ON EMP(SAL);
```

Related Topics

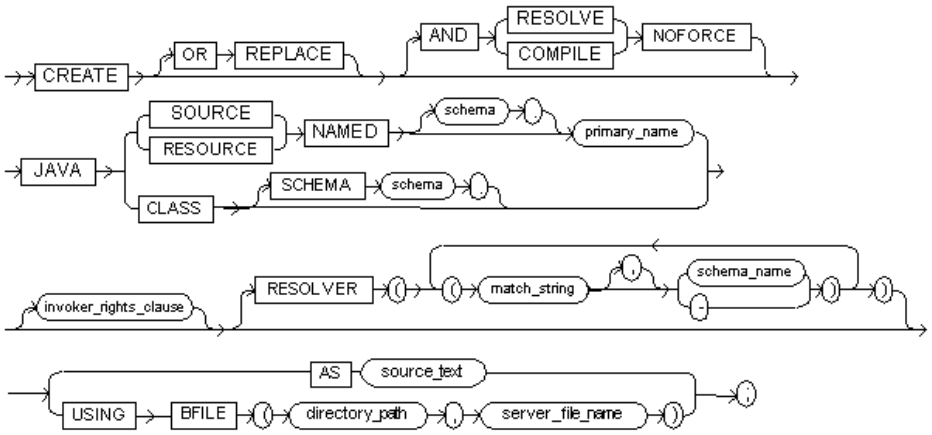
CONSTRAINT clause, CREATE TABLE, DROP INDEX

4.3.13 CREATE JAVA

Syntax

The syntax for CREATE JAVA is displayed in Figure 4–20.

Figure 4–20 The CREATE JAVA Command



BNF Notation

```
CREATE [OR REPLACE] [AND { RESOLVE | COMPILE } NOFORCE ]  JAVA
{ { SOURCE | RESOURCE } NAMED [schema .] primary_name
    | CLASS [SCHEMA schema .]
}

[invoker_rights_clause]
[RESOLVER
"(" "(" match_string [,] { schema_name | - }")"
   ["(" match_string [,] { schema_name | - }")"]...
 ")"
]

{ USING  BFILE "(" directory_path , server_file_name ")"
  | AS source_text
 }
;
```

Prerequisite

To create or replace a schema object containing a Java source, class, or resource in your own schema, you must be connected to the database as SYSTEM or you must have DBA/DDL privileges.

Purpose

To create a schema object containing a Java source, class, or resource.

Note: For information on Java concepts, including Java stored procedures and JDBC, see the *Oracle Database Lite Developer's Guide for Java*.

The arguments for the CREATE JAVA command are listed in Table 4–18.

Table 4–18 Arguments Used with the CREATE JAVA Command

Argument	Description
OR REPLACE	Recreates the schema object containing the Java class, source, or resource if it already exists. Use this clause to change the definition of an existing object without dropping, re-creating, and regranting object privileges previously granted. If you redefine a Java schema object and specify RESOLVE or COMPILE, Oracle Database Lite recognizes but ignores those parameters. Users, previously granted privileges on a redefined function, can still access the function. You do need to re-grant privileges to the users.
RESOLVE | COMPILE	Oracle Database Lite recognizes but ignores this parameter. In Oracle, you specify that the database should attempt to resolve the Java schema object that is created if this statement succeeds. ■ When applied to a class, resolution of referenced names to other class schema objects occurs. ■ When applied to a source, source compilation occurs. *Restriction*: You cannot specify this clause for a Java resource.
NOFORCE	Oracle Database Lite recognizes but ignores this parameter. In Oracle NO FORCE rolls back the results of this CREATE command if you have specified either RESOLVE OR COMPILE, and the resolution or compilation fails. If you do not specify this option, Oracle takes no action if the resolution or compilation fails (that is, the created schema object remains).
CLASS	Loads a Java class file.
RESOURCE	Loads a Java resource file.
SOURCE	Loads a Java source file. Requires the use of the AS *source_text* clause.
NAMED	Oracle Database Lite recognizes but ignores this parameter. In Oracle, it is *required* for a Java source or resource. ■ For a Java source, this clause specifies the name of the schema object in which the source code is held. A successful CREATE JAVA SOURCE statement also creates additional schema objects to hold each of the Java classes defined by the source. ■ For a Java resource, this clause specifies the name of the schema object to hold the Java resource. If you do not specify schema, Oracle creates the object in your own schema. Restrictions: ■ You cannot specify NAMED for a Java class. ■ The *primary_name* cannot contain a database link.

Table 4–18 (Cont.) Arguments Used with the CREATE JAVA Command

Argument	Description
SCHEMA *schema*	Oracle Database Lite recognizes but ignores this parameter. In Oracle, it applies only to a Java class. This optional clause specifies the schema in which the object containing the Java file resides. If you do not specify SCHEMA and you do not specify NAMED (above), Oracle creates the object in your own schema.
invoker_rights_clause	For compatibility with Oracle, Oracle Database Lite recognizes but does not enforce the *invoker_rights_clause*.
RESOLVER	Oracle Database Lite recognizes but ignores this parameter. In Oracle, it specifies a mapping of the fully qualified Java name to a Java schema object, where: ■ *match_string* is either a fully qualified Java name, a wildcard that can match such a Java name, or a wildcard that can match any name. ■ *schema_name* designates a schema to be searched for the corresponding Java schema object. ■ A dash (-) as an alternative to *schema_name* indicates that if *match_string* matches a valid Java name, Oracle can leave the schema unresolved. The resolution succeeds, but the name cannot be used at run time by the class. This mapping is stored with the definition of the schema objects created in this command for use in later resolutions (either implicit or in explicit ALTER...RESOLVE statements).
AS *source_text*	A text of a Java source program.
USING BFILE	Identifies the format of the class file. BFILE is interpreted as a binary file by the CREATE JAVA CLASS or CREATE JAVA RESOURCE.

Usage Notes

When Oracle Database Lite loads a Java class into the database, it does not load dependent classes. Generally, you should use the `loadjava` utility to load Java classes into the database. See the *Oracle Database Lite Developer's Guide for Java* for more information about the `loadjava` utility.

Java Class Example

The following statement creates a schema object and loads the specified Java class into the newly created schema object.

```
CREATE JAVA CLASS USING BFILE (bfile_dir, 'Agent.class');
```

This example assumes the directory path `bfile_dir`, which points to the operating system directory containing the Java class `Agent.class`, already exists. In this example, the name of the class determines the name of the Java class schema object.

Java Source Example

The following statement creates a Java source schema object:

```
CREATE OR REPLACE JAVA SOURCE AS
/* This is a class Test */
import java.math.*; /* */
public class Test {
public static BigDecimal myfunc(BigDecimal a, BigDecimal b)
{ return a.add(b); }
public static Strin myfunc2(String a, String b)
```

```
{ return (a+b); }
};
```

> **Note:** The keyword `public class` should not be used in a comment before the first public class statement.

Java Resource Example

The following statement creates a Java resource schema object named `APPTEXT` from a binary file.

```
CREATE JAVA RESOURCE NAMED "appText"
   USING BFILE ('C:\TEMP', 'textBundle.dat');
```

> **Note:** when embedding any Java statements, the semi-colon character, `" ; "` cannot be the last character in an SQL*Plus statement. If the semi-colon must be the last character in a line, a blank comment line must be added using the following characters: "**/* */**" . The regular comment symbols, "//" do not work in this context. Placing /* */ at the end of the line prevents SQL*Plus from interpreting the semi-colon as the end of the SQL statement.

Related Topics

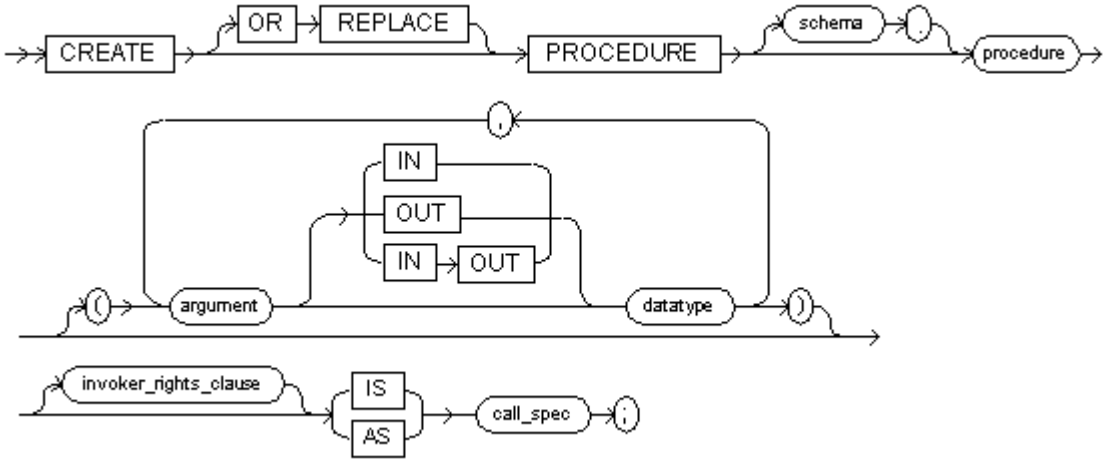
DROP JAVA

4.3.14 CREATE PROCEDURE

Syntax

The syntax for `CREATE PROCEDURE` is displayed in Figure 4–21.

Figure 4–21 The CREATE PROCEDURE Command



BNF Notation

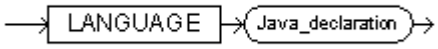
```
CREATE [OR REPLACE] PROCEDURE [schema .] procedure
["(" argument [ IN | OUT | IN OUT ] datatype
   [, argument [ IN | OUT | IN OUT ] datatype]...
```

```
 ")"
]
[invoker_rights_clause] { IS | AS } call_spec
;
```

call_spec::=

The syntax for the `call_spec` expression is displayed in Figure 4–22.

Figure 4–22 The call_spec Expression used with CREATE PROCEDURE



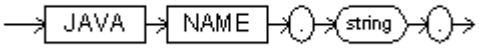
BNF Notation

```
LANGUAGE Java_declaration
```

Java_declaration::=

The syntax for the `Java_declaration` expression is displayed in Figure 4–23.

Figure 4–23 The Java_declaration Expression used with CREATE PROCEDURE



BNF Notation

```
JAVA NAME . string .
```

Prerequisite

To create a procedure in your own schema, you must be connected to the database as `SYSTEM` or you must have DBA/DDL privileges.

Purpose

To create a call specification for a stand alone stored procedure.

A call specification ("call spec") declares a Java method so that it can be called from SQL. The call spec tells Oracle which Java method to invoke when a call is made. It also tells Oracle Database Lite what type conversions to make for the arguments and return value.

Stored procedures offer advantages in the areas of development, integrity, security, and memory allocation. For more information on stored procedures, including how to call stored procedures, see the *Oracle Database Lite Developer's Guide for Java.*

Stored procedures and stored functions are similar. While a stored function returns a value to the environment in which it is called, a stored procedure does not. For information specific to functions, see CREATE FUNCTION.

The `CREATE PROCEDURE` statement creates a procedure as a stand alone schema object. For information on dropping a stand alone procedure, see DROP PROCEDURE.

The arguments for the `Create Procedure` command are listed in Table 4–19.

Table 4–19 Arguments Used with the Create Procedure Command

Argument	Description
`OR REPLACE`	Recreates the procedure if it already exists. Use this clause to change the definition of an existing procedure without dropping, re-creating, and regranting object privileges previously granted on it. If any function-based indexes depend on the package, Oracle Database Lite marks the indexes `DISABLED`.
schema	The schema to contain the procedure. If you omit schema, Oracle Database Lite creates the procedure in your current schema.
procedure	The name of the procedure to create.
argument	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`	Indicates that you must specify a value for the argument when calling the procedure.
`OUT`	Indicates that the procedure passes a value for this argument back to its calling environment after execution.
`IN OUT`	Indicates 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`. Changes made either to this parameter or to another parameter may be visible immediately through both names if the same variable is passed to both. If the procedure is exited with an unhandled exception, any assignment made to this parameter may be visible in the caller's variable. These effects may or may not occur on any particular call. You should use `NOCOPY` only when these effects would not matter.
datatype	The datatype of the argument. An argument can have any datatype supported by Oracle Database Lite SQL. Datatypes cannot specify length, precision, or scale. For example, `VARCHAR2(10)` is not valid, but `VARACHAR2` is valid. Oracle Database Lite derives the length, precision, and scale of an argument from the environment from which the procedure is called.
invoker_rights_clause	For compatibility with Oracle, Oracle Database Lite recognizes but does not enforce the *invoker_rights_clause*.
`IS`	Associates the SQL identifier with the Java method.
`AS`	Associates the SQL identifier with the Java method.
call_spec	Maps the Java method name, parameter types, and return type to SQL counterparts.
`LANGUAGE`	Specifies the *call_spec* language. In Oracle this can be C or Java. In Oracle Database Lite, this can only be Java.
Java_declaration	Identifies the method name in the Java class.
`JAVA NAME`	The Java method name.
string	Identifies the Java implementation of the method. For more information, see the *Oracle Database Lite Developer's Guide for Java*.

Usage Notes

Oracle Database Lite recognizes but does not enforce the *<invoker_rights_clause>*. Oracle Database Lite always uses `current_user` for `AUTHID`.

Example

The following example creates and compiles a Java procedure and tests it against Oracle Database Lite.

1. Create and compile the following Java program and name it **EMPTrigg.java**:

```
import java.sql.*;

public class EMPTrigg {
   public static final String goodGuy = "Oleg";

   public static void NameUpdate(String oldName, String[] newName) {
      if (oldName.equals(goodGuy))
         newName[0] = oldName;
   }

   public static void SalaryUpdate(String name, int oldSalary,
                            int newSalary[])
   {
      if (name.equals(goodGuy))
         newSalary[0] = Math.max(oldSalary, newSalary[0])*10;
   }

   public static void AfterDelete(Connection conn, String name,
               int salary) {
      if (name.equals(goodGuy))
         try {
            Statement stmt = conn.createStatement();
            stmt.executeUpdate(
               "insert into employee values('" + name + "', " +
                                    salary + ")");
            stmt.close();
         } catch(SQLException e) {}
   }
}
```

2. Create the `EMPLOYEE` table with the `NAME` and `SALARY` columns.

```
CREATE TABLE EMPLOYEE (NAME VARCHAR(32), SALARY INT);
```

3. Insert values into the `EMPLOYEE` table by typing the following statements.

```
INSERT INTO EMPLOYEE VALUES ('Alice', 100);

INSERT INTO EMPLOYEE VALUES ('Bob', 100);

INSERT INTO EMPLOYEE VALUES ('Oleg', 100);
```

4. Load the `EMPTrigg` class into Oracle Database Lite. Once loaded, the `EMPTrigg` class methods become stored procedures in Oracle Database Lite.

```
CREATE JAVA CLASS USING BFILE ('c:\', 'EMPTrigg.class');
```

5. Use the `CREATE PROCEDURE` statement to enable SQL to call the methods in the `EMPTrigg` class.

```
CREATE PROCEDURE name_update(
```

```
old_name in varchar2, new_name in out varchar2)
is language java name
'EMPTrigg.NameUpdate (java.lang.String, java.lang.String[])';
/

 CREATE PROCEDURE salary_update(
 ename varchar2, old_salary int, new_salary in out int)
 as language java name
 'EMPTrigg.SalaryUpdate (java.lang.String, int, int[])';
 /

 CREATE PROCEDURE after_delete(
 ename varchar2, salary int)
 as language java name
 'EMPTrigg.AfterDelete (java.sql.Connection, java.lang.String, int)';
 /
```

6. Create a trigger for each of the stored procedures.

```
CREATE TRIGGER NU BEFORE UPDATE OF NAME ON EMPLOYEE FOR EACH ROW
name_update (old.name, new.name);
/

CREATE TRIGGER SU BEFORE UPDATE OF SALARY ON EMPLOYEE FOR EACH ROW
salary_update (name, old.salary, new.salary);
/

CREATE TRIGGER AD AFTER DELETE ON EMPLOYEE FOR EACH ROW
after_delete (name, salary);
/
```

7. Select all rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE;
```

Returns the following result:

```
NAME                             SALARY
-------------------------------- ---------
Alice                               100
Bob                                 100
Oleg                                100
```

Related Topics

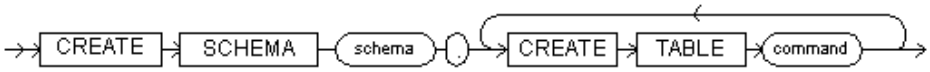
DROP PROCEDURE

4.3.15 CREATE SCHEMA

Syntax

The syntax for the CREATE SCHEMA command is displayed in Figure 4–24.

Figure 4–24 The CREATE SCHEMA Command



BNF Notation

```
CREATE SCHEMA schema . CREATE TABLE command [ CREATE TABLE command]... ;
```

Prerequisite

The CREATE SCHEMA statement can include the CREATE TABLE, CREATE VIEW, and GRANT statements. To issue a CREATE SCHEMA statement, you must be logged into the database as SYSTEM or as a user with DBA/DDL or ADMIN privileges.

Purpose

Creates a schema or an owner of tables, indexes, and views. CREATE SCHEMA can also be used to create multiple tables and views in a single transaction.

The arguments for the CREATE SCHEMA command are listed in Table 4–20.

Table 4–20 Arguments Used with the CREATE SCHEMA Command

Argument	Description
schema	The name of the schema, which is a character string of up to 128 characters. The schema name must be different from any user names since each user name has a default schema with the same name. If you create a schema with the same name as a user name, Oracle Database Lite returns an error. See CREATE USER for more information.
CREATE TABLE	A CREATE TABLE statement to be issued as part of the CREATE SCHEMA statement.
command	Contains all the arguments and keywords for a CREATE TABLE or CREATE VIEW command.

Usage Notes

- Oracle Database Lite treats the schema as the user's private database. Informally, a schema defines a separate name space and a scope of ownership. In other words, two tables may have the same name if they reside in different schemas. All tables and views in the same schema are owned by the owner of that schema. To use a schema different from the one currently in use, you must first disconnect from the current schema, then connect to the new schema.
- CREATE SCHEMA treats a group of separate statements as a single statement; if one of its constituent statements fails, all of its statements are reversed.
- The name of the new schema appears in the POL_SCHEMATA view.

Example 1

To create a sample schema called HOTEL_OPERATION use.

```
CREATE SCHEMA HOTEL_OPERATION;
```

Example 2

To create the schema HOTEL_OPERATION together with the table HOTEL_DIR and the view LARGE_HOTEL use.

```
CREATE SCHEMA HOTEL_OPERATION
CREATE TABLE HOTEL_DIR(
HOTELNAME CHAR(40) NOT NULL,
RATING INTEGER,
ROOMRATE FLOAT,
LOCATION CHAR(20) NOT NULL,
CAPACITY INTEGER);
```

ODBC 2.0

Although the CREATE SCHEMA command is not part of the ODBC SQL syntax, ODBC passes the command through to your database.

Related Topics

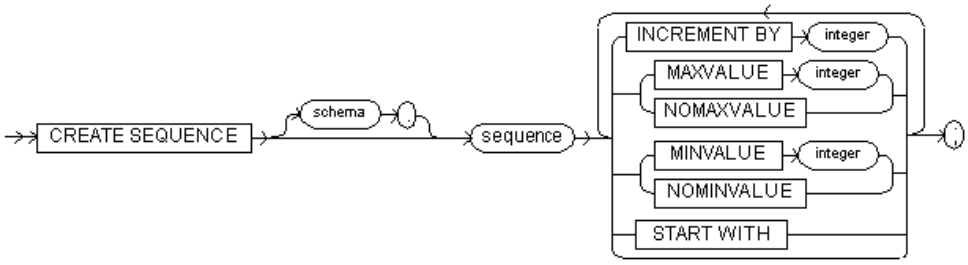
GRANT, CREATE SEQUENCE, CREATE VIEW

4.3.16 CREATE SEQUENCE

Syntax

The syntax for CREATE SEQUENCE is displayed in Figure 4–25.

Figure 4–25 The CREATE SEQUENCE Command



BNF Notation

```
CREATE SEQUENCE [schema .] sequence
{ { INCREMENT BY } integer
 | { MAXVALUE integer | NOMAXVALUE }
 | { MINVALUE integer | NOMINVALUE }
 | { START WITH } integer
 }
[{ { INCREMENT BY } integer
 | { MAXVALUE integer | NOMAXVALUE }
 | { MINVALUE integer | NOMINVALUE }
 | { START WITH } integer
 }]...
;
```

Prerequisite

None

Purpose

Creates a sequence.

The arguments for the CREATE SEQUENCE command are listed in Table 4–21.

Table 4–21 Arguments Used with the CREATE SEQUENCE Command

Argument	Description
schema	The name of the schema to contain the sequence. If you omit schema, Oracle Database Lite creates the sequence in your own schema.
sequence	The name of the sequence to be created.
INCREMENT BY	Specifies the interval between sequence numbers. Can be any positive or negative integer, but cannot be 0. If negative, then the sequence descends. If positive, the sequence ascends. If you omit the INCREMENT BY clause, the default is 1.
START WITH	Specifies the first sequence number to be generated. Use this option to start an ascending sequence at a value greater than its minimum (which is the default), or to start a descending sequence at a value less than its maximum (which is the default).
MAXVALUE	Specifies the maximum value the sequence can generate. This integer value can have 9 or fewer digits. MAXVALUE must be greater than MINVALUE.
NOMAXVALUE	Specifies a maximum value of 2147483647 for an ascending sequence or –1 for a descending sequence.
MINVALUE	Specifies the minimum value that the sequence can generate. This integer value can have 9 or fewer digits. MINVALUE must be less than MAXVALUE.
NOMINVALUE	Specifies a minimum value of 1 for an ascending sequence or –2147483647 for a descending sequence.

Usage Notes

Oracle Database Lite commits sequence numbers when you access the NEXTVAL function. However, unlike Oracle, Oracle Database Lite does not automatically commit sequences. As a result, you can roll back sequences in Oracle Database Lite. To maintain a sequence when using the ROLLBACK command, you must commit the sequence after you create it.

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 returns a value 10 greater than the previous one.

ODBC 2.0

Although the CREATE SEQUENCE command is not part of the ODBC SQL syntax, ODBC passes the command through to your database.

Related Topics

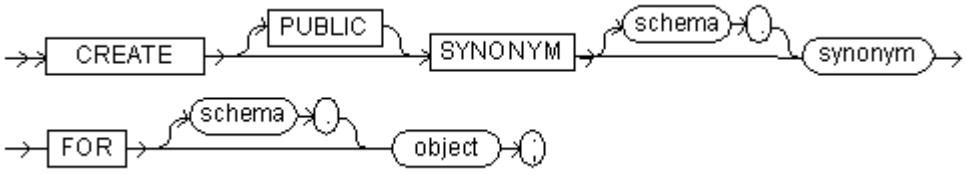
ALTER SEQUENCE, DROP SEQUENCE

4.3.17 CREATE SYNONYM

Syntax

The syntax for CREATE SYNONYM is displayed in Figure 4–26.

Figure 4–26 The CREATE SYNONYM Command



BNF Notation

```
CREATE [PUBLIC] SYNONYM [schema .] synonym
FOR [schema .] object ;
```

Prerequisite

None

Purpose

Creates a public or private SQL synonym.

The arguments for the `CREATE SYNONYM` command are listed in Table 4–22.

Table 4–22 Arguments Used with the CREATE SYNONYM Command

Argument	Description
`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	The schema to contain the synonym. If you omit *schema*, Oracle Database Lite creates the synonym in your own *schema*. You cannot specify schema if you have specified `PUBLIC`.
synonym	The name of the synonym to be created.
`FOR` *object*	Identifies the object for which the synonym is created. If you do not qualify the object with a schema, Oracle Database Lite assumes that the object is in your own schema. The object can be a table, view, sequence, or another synonym. Note that the object need not currently exist and you must have privileges to access the object.

Usage Notes

A private synonym name must be distinct from all other objects in its schema.

You can only use synonyms with the `INSERT`, `SELECT`, `UPDATE`, and `DELETE` statements. You cannot use synonyms with the `DROP` statement.

Example

To define the synonym `PROD` for the table `PRODUCT` in the schema `SCOTT`, issue the following statement.

```
CREATE SYNONYM PROD FOR SCOTT.PRODUCT;
```

Related Topics

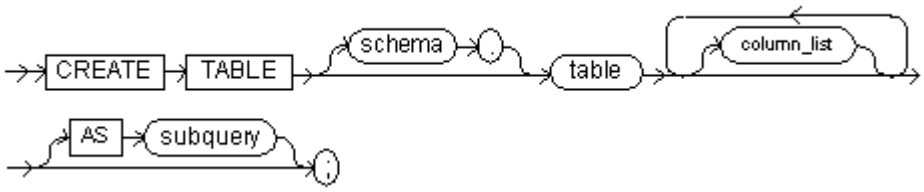
CREATE TABLE, CREATE VIEW, CREATE SEQUENCE, DROP SYNONYM

4.3.18 CREATE TABLE

Syntax

The syntax for the CREATE TABLE command is displayed in Figure 4–27.

Figure 4–27 The CREATE TABLE Command



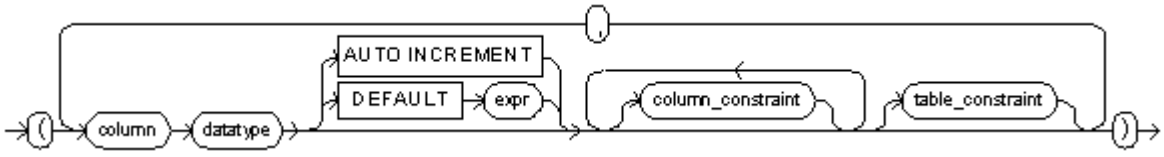
BNF Notation

```
CREATE TABLE [schema .] table
column_list [column_list ]...
[AS subquery] ;
```

column_list::=

The syntax for the column_list expression is displayed in Figure 4–28.

Figure 4–28 The column_list Expression



BNF Notation

```
"("
column datatype [DEFAULT expr|AUTO INCREMENT][column_constraint] [column_
constraint]...
[table_constraint]
[, column datatype [DEFAULT expr|AUTO INCREMENT][column_constraint] [column_
constraint]...
[table_constraint]]...
")"
```

Prerequisite

To create a table in your schema or another schema, you must be logged into the database as SYSTEM or as a user with DBA/DDL privileges.

Purpose

Creates a database table.

The CREATE TABLE command creates and populates a database table based on the result of a specified sub-query. The datatypes for the column are derived from the subquery's result set. See Usage Notes for more information.

The arguments for the CREATE TABLE command are listed in Table 4–23.

Table 4–23 Arguments Used with the CREATE TABLE Command

Argument	Description
schema	A schema, which has the same name as the user who owns it. If omitted, the default schema name is used.
table	The name of a database table. Table names may not contain the period "." character, nor begin with an underscore "_" character.
column	The name of a table column.
datatype	The datatype of the column. Cannot be used in subquery.
DEFAULT	The DEFAULT clause enables you to assign a value 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. To contain this expression, the column size must be increased. The DEFAULT expression can include any SQL function provided the function does not return a column reference or a nested function invocation. **Restrictions on Default Common Values** A DEFAULT expression cannot contain references to Java stored procedures, other columns or the psuedo columns named LEVEL, PRIOR, and ROWNUM. A DEFAULT expression cannot contain a sub query. For more information about expressions, see Chapter 1, "Using SQL", Section 1.8, "Specifying Expressions".
auto increment	Set the column to auto increment column. The data type for any auto increment column has to be of the type INTEGER. The value of an auto increment column is auto incremented and inserted, so that the user does not have to provide the value. The value is unique in the table and contains no null value, and thus can be used as a primary key column, when required. The value of the column is determined by the database system and the user does not have means to control the amount incremented, the start value, or the maximum value. The value of the auto increment column starts with 0 and the maximum positive value is the maximum value of a 4-byte integer (2147483647). Once the auto-incremented value reaches the maximum value, the next auto-incremented value starts from the minimum value of the 4-byte integer (-2147483648).
column_constraint	Adds a column integrity constraint. For more information, see "CONSTRAINT clause".
table_constraint	Adds a table integrity constraint. For more information, see "CONSTRAINT clause".
AS *subquery*	A SELECT statement.

Usage Notes

CREATE ANY TABLE can be used to create a table in another schema, but this requires the DBA/DDL role. Each table can have upto 1000 columns and no more than one primary key constraint.

If the column_list is omitted.

- If table columns are not defined when specifying a sub query, column names are derived from the expressions selected from the sub query.

- If an expression in the select list contains an alias, then the alias is used as the column name.
- If an expression is a column with no alias name, then its name is used as the column name. An expression is illegal if it is not a column and has no alias. The datatypes for the table's columns are the same as the datatypes for the corresponding expressions in the select list of the sub query.
- If the subquery contains UNION or MINUS, the first select statement is chosen for this purpose.

If the column_list is omitted.

- The number of columns in the column_list must equal the number of expressions in the sub query.
- The column definitions can specify only column names, default values, and integrity constraints, but not datatypes or auto incremented columns.
- A referential integrity constraint cannot be defined using the CREATE TABLE statement form. Instead, an ALTER TABLE statement can be used to create the referential integrity constraint at a later point.

If an ORDER BY clause is used in the sub query, the data is inserted in the specified order into the table. This normaly results in clustering of the data according to the order by columns, but is not guaranteed.

To insert into tables with auto-incremented column(s), since the value of an auto-incremented column is generated automatically by the database system, there is no insert operation allowed on this column. To insert a row into a table that has auto increment column(s), the user has to specify the column list that contains no auto increment column(s) for the insert operation to be successful. For example, assuming that we have the following table defined.

```
CREATE TABLE t1 (c1 INT AUTO INCREMENT, c2 INT, c3 INT);
```

To insert into table t1, use the following command.

```
INSERT INTO T1(c2,c3) values (123, 456);
```

If the user does not specify the column list, an error message is returned.

To avoid the column list in the insert statement, the auto-incremented column can be hidden before issuing the INSERT command. For example, if we have the following ALTER COMMAND issued.

```
ALTER TABLE T1 HIDE C1;
```

Then, to insert into table t1, the insert statement can omit the column list as given below.

```
INSERT INTO T1 VALUES (123,456);
```

Example 1

The following statement creates a table named HOTEL_DIR with two columns. They are: HOTEL_NAME which is the primary key, and CAPACITY, which is not nullable and has the default value 0.

```
CREATE TABLE HOTEL_DIR (HOTEL NAME CHAR(40) PRIMARY KEY,
CAPACITY INTEGER DEFAULT 0 NOT NULL)
```

Example 2

The following statement creates a table named HOTEL_RESTAURANT.

```
CREATE TABLE HOTEL_RESTAURANT(REST_NAME CHAR(50) UNIQUE, HOTEL_
NAME CHAR(40) REFERENCES HOTEL_DIR, RATING FLOAT DEFAULT NULL)
```

The columns include.

- REST_NAME - Restaurant name.
- HOTEL_NAME - Name of the hotel that the restaurant is in.
- RATING - Restaurant rating. The default value is null.

The table has the following integrity constraints.

- Two hotels or restaurants cannot have the same name.
- HOTEL_NAME must refer to a hotel in the HOTEL_DIR table.

Related Topics

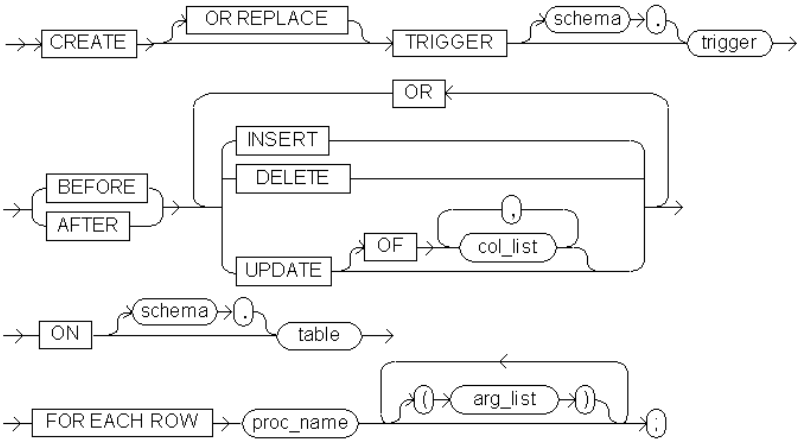
CONSTRAINT clause, DROP TABLE, Transaction Control Commands, SELECT

4.3.19 CREATE TRIGGER

Syntax

The syntax for CREATE TRIGGER is displayed in Figure 4–29.

Figure 4–29 The CREATE TRIGGER Command



BNF Notation

```
CREATE [OR REPLACE] TRIGGER [schema .] trigger
{ BEFORE | AFTER }
{ DELETE | INSERT | UPDATE [OF column [, column]...] }
[OR { DELETE | INSERT | UPDATE [OF col_list [, col_list]...] }]...
ON { [schema .] table
FOR EACH ROW proc_name ["("arg_list")"] ["("arg_list")"]...
;
```

Prerequisite

None

Purpose

Creates and enables a database trigger.

The arguments for the CREATE TRIGGER command are listed in Table 4–24.

Table 4–24 Arguments Used with the CREATE TRIGGER Command

Argument	Description
OR REPLACE	Recreates the trigger if it already exists. Creates the trigger if it does not already exist. Used to change the definition of an existing trigger without dropping, recreating, or regranting object privileges previously granted on it.
schema	The schema to contain the trigger. If omitted, Oracle Database Lite creates the trigger in your own schema.
table	The name of a table in the database.
trigger	The name of the trigger to be created.
BEFORE	Specifies that the trigger should be fired before executing the triggering statement. For row triggers, this is a separate firing before each affected row is changed.
AFTER	Specifies that the trigger should be fired after executing the triggering statement. For row triggers, this is a separate firing after each affected row is changed.
DELETE	Specifies that the trigger should be fired whenever a DELETE statement removes a row from the table.
INSERT	Specifies that the trigger should be fired whenever an INSERT statement adds a row to the table.
UPDATE OF	Specifies that the trigger should be fired whenever an UPDATE statement changes a value in one of the columns specified in the OF clause. If you omit the OF clause, Oracle Database Lite fires the trigger whenever an UPDATE statement changes a value in any column of the table.
col_list	The column(s) that, when updated, cause the trigger to be fired.
ON	Specifies the schema and name of the table on which the trigger is to be created. If omitted, Oracle Database Lite assumes the table is in your own schema.
FOR EACH ROW	Designates the trigger to be a row trigger. Oracle Database Lite fires a row trigger once for each row that is affected by the triggering statement. If you omit this clause, the trigger is a statement trigger. Oracle Database Lite fires a statement trigger only once when the triggering statement is issued if the optional trigger constraint is met.
proc_name	Name of the Java method Oracle Database Lite executes to fire the trigger.
arg_list	Arguments passed to the Java method.

Example

The following example provides you with instructions for creating and testing a trigger.

1. Create the following Java program and name it TriggerExample.java.

```
import java.lang.*;
import java.sql.*;
class TriggerExample {
        public void EMP_SAL(Connection conn, int new_sal)
        {
            System.out.println("new salary is :"+new_sal);
```

```
        }
    }
```

2. Attach `TriggerExample.java` to the EMP table.

```
ALTER TABLE EMP ATTACH JAVA SOURCE "TriggerExample" in '.';
```

3. Create the Java trigger.

```
CREATE TRIGGER SAL_CHECK BEFORE UPDATE OF SAL ON EMP FOR EACH ROW
EMP_SAL(NEW.SAL);
 .
 /
```

4. Update the EMP table using the Java trigger.

```
update emp set sal=sal+5000 where sal=70000;
```

Returns the following result.

```
new salary is:75000

1 row updated
```

Related Topics

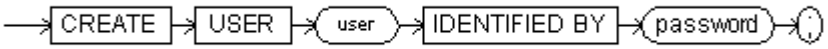
ALTER TRIGGER, ALTER VIEW, CREATE VIEW, DROP TRIGGER

4.3.20 CREATE USER

Syntax

The syntax for `CREATE USER` is displayed in Figure 4–30.

Figure 4–30 The CREATE USER Command



BNF Notation

```
CREATE USER user IDENTIFIED BY password ;
```

Prerequisite

To create users in your schema or other schemas, you must be logged into the database as `SYSTEM` or as a user with DBA/DDL privileges.

Purpose

Creates a database user with no privileges.

The arguments for the `CREATE USER` command are listed in Table 4–25.

Table 4–25 Arguments Used with the CREATE USER Command

Argument	Description
user	The user to be created. Here, *user* is a unique string, beginning with a letter, with a minimum of one byte and a maximum length of 30 bytes.

Table 4–25 (Cont.) Arguments Used with the CREATE USER Command

Argument	Description
`IDENTIFIED BY`	Indicates how Oracle Database Lite permits user access.
password	Specifies a new password for the user which is a name of up to 128 characters. The password does not appear in quotes and is not case-sensitive.

Usage Notes

You can create multiple users in Oracle Database Lite by using the CREATE USER command. A user is not a schema. When you create a user, Oracle Database Lite creates a schema with the same name and automatically assigns it to the new user as the default schema. The name of the new user appears in the `ALL_USERS` view. The new user's default schema appears in the `POL_SCHEMATA` view.

When you connect to an Oracle Lite database as a user, the user name becomes the default schema for that session. If there is no schema to match the user name, Oracle Lite refuses the connection. You can access database objects in the default schema without prefixing them with the schema name.

Users with the appropriate privileges can create additional schemas by using the CREATE SCHEMA command, but only the default schema can connect to the database. These schemas are owned by the user who created them and require the schema name prefix to access their objects.

When you create a database using the CREATEDB utility or the CREATE DATABASE command, Oracle Lite creates a special user called SYSTEM with password of MANAGER. This user has all database privileges. You can use SYSTEM as the default user name until you establish user names of your own as needed.

For encrypted databases, all user names and passwords are written to a file named `mydbname.opw`. Each user can then use their own password as a key to unlock the `.opw` file before the `.odb` file is accessed. When you copy or back up the database, you should include the `.opw` file and the `.plg` file.

Oracle Lite does not permit a user other than SYSTEM to access data or perform operations in a schema that is not its own. Users can only access data and perform operations in a different user's schema if one of the following conditions is met:

- The user is granted a pre-defined role in another user's schema, which permits the user to perform the operation.
- The user is granted specific privileges in another user's schema.

> **Note:** The user SYSTEM must grant DBA/DDL or RESOURCE privileges to a new user before the new user can create database objects. The DBA role is recommended as a replacement for the DDL role wherever possible.

Example

```
CREATE USER SCOTT IDENTIFIED BY TIGER;
```

Related Topics

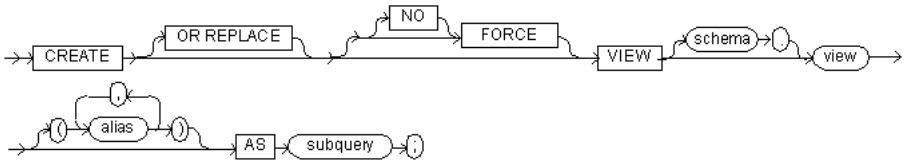
ALTER USER, GRANT

4.3.21 CREATE VIEW

Syntax

The syntax for `CREATE VIEW` is displayed in Figure 4–31.

Figure 4–31 The CREATE VIEW Command



BNF Notation

```
CREATE [OR REPLACE] [[NO] FORCE] VIEW [schema .] view
["("alias [, alias]..."")"] AS subquery ;
```

Prerequisite

You must be logged into the database as SYSTEM or as a user with DBA/DDL privileges.

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

NO FORCE creates the view only if the base tables exist and the owner of the schema containing the view has privileges on them. This is the default.

Purpose

Creates or replaces a view.

The arguments for the `CREATE VIEW` command are listed in Table 4–26.

Table 4–26 Arguments Used with the CREATE VIEW Command

Argument	Description
OR REPLACE	Recreates the view if it already exists. Used to change the definition of an existing view without dropping, recreating, or re-granting object privileges previously granted.
FORCE	Specify FORCE if you want to create the view regardless of whether the view's base tables or the referenced object types exist or the owner of the schema containing the view has privileges on them. These conditions must be true before any SELECT, INSERT, UPDATE, or DELETE statements can be issued against the view.
NO FORCE	Specify NO FORCE if you want to create the view only if the base tables exist and the owner of the schema containing the view has privileges on them. This is the default option.
schema	The schema to contain the view. If you omit *schema*, Oracle Lite creates the view in your own schema.
view	The name of the view.

Table 4–26 (Cont.) Arguments Used with the CREATE VIEW Command

Argument	Description
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 Oracle Lite's rules for naming schema objects. Each *alias* must be unique within the view.
AS *subquery*	Identifies columns and rows of the table(s) on which the view is based. 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.

Usage Notes

A view is updatable if:

- The subquery selects from a single base table or from another updatable view.
- Each selected expression is a column reference to that base table or updatable view.
- No two column references in the select list reference the same column.

CREATE ANY VIEW can be used to create a view in another schema, but this requires the DBA/DDL role.

The FORCE option of CREATE VIEW behaves differently under Oracle Database Lite. There are two cases:

1. A command issued to a view created by using CREATE FORCE VIEW without the base table must have the ALTER VIEW *view_name* COMPILE command issued first, otherwise an error message is thrown.
2. A CREATE FORCE VIEW created with a valid base table is no different than CREATE VIEW.

Example

The following example creates a view called EMP_SAL which displays the name, job, and salary of each row in the EMP table:

```
CREATE VIEW EMP_SAL (Name, Job, Salary) AS SELECT ENAME, JOB, SAL FROM EMP;

SELECT * FROM EMP_SAL;
```

Returns the following result:

```
NAME       JOB          SALARY
---------- --------- ---------
KING       PRESIDENT      5000
BLAKE      MANAGER        2850
CLARK      MANAGER        2450
JONES      MANAGER        2975
MARTIN     SALESMAN       1250
ALLEN      SALESMAN       1600
TURNER     SALESMAN       1500
JAMES      CLERK           950
WARD       SALESMAN       1250
FORD       ANALYST        3000
SMITH      CLERK           800
SCOTT      ANALYST        3000
ADAMS      CLERK          1100
```

```
MILLER     CLERK          1300

14 rows selected.
```

ODBC 2.0

Although the ODBC SQL syntax for CREATE VIEW does not support the OR REPLACE argument, ODBC passes the command through to your database.

Editing Data in a View

Most ODBC-based tools require a primary key before allowing updates on a view. Oracle Lite does not report primary keys for views, so you must issue SQL commands to perform updates or deletes on views using the WHERE clause to specify the target row or rows.

Related Topics

DROP SEQUENCE, CREATE TABLE, DROP VIEW

4.3.22 CURRVAL and NEXTVAL pseudocolumns

Purpose

A sequence is a schema object that can generate unique sequential values. These values are often used for primary and unique keys. You can use the `CURRVAL` and `NEXTVAL` pseudocolumns to refer to sequence values in SQL statmetments.

Prerequisite

You must have a sequence object.

Usage Notes

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 qualify the sequence with the schema containing it.

```
schema.sequence.CURRVAL
schema.sequence.NEXTVAL
```

You can use `CURRVAL` and `NEXTVAL` in:

- The SELECT list of a `SELECT` statement that is not contained in a subquery, materialized view, 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:

- A query of a view or of a `materialized` view.
- A `SELECT` statement with the DISTINCT operator.
- A `SELECT` statement with a `GROUP BY` clause 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 that uses CURRVAL or NEXTVAL, 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. Within a single SQL statement, Oracle Database Lite will increment the sequence only once for each row. If a statement contains more than one reference to NEXTVAL for a sequence, Oracle increments the sequence once and returns the same value for all occurrences of NEXTVAL. If a statement contains references to both CURRVAL and NEXTVAL, Oracle increments the sequence and returns the same value for both CURRVAL and NEXTVAL regardless of their order within the statement.

A sequence can be accessed by many users concurrently with no waiting or locking.

Example 1

This example selects the current value of the employee sequence in the sample schema hr:

```
SELECT employees_seq.currval
    FROM DUAL;
```

Example 2

This example increments the employee sequence and uses its value for a new employee inserted into the sample table hr.employees:

```
INSERT INTO employees
   VALUES (employees_seq.nextval, 'John', 'Doe', 'jdoe',
   '555-1212', TO_DATE(SYSDATE), 'PU_CLERK', 2500, null, null,
   30);
```

Example 3

This example adds a new order with the next order number to the master order table. It then adds suborders with this number to the detail order table:

```
INSERT INTO orders (order_id, order_date, customer_id)
   VALUES (orders_seq.nextval, TO_DATE(SYSDATE), 106);

INSERT INTO order_items (order_id, line_item_id, product_id)
   VALUES (orders_seq.currval, 1, 2359);

INSERT INTO order_items (order_id, line_item_id, product_id)
   VALUES (orders_seq.currval, 2, 3290);

INSERT INTO order_items (order_id, line_item_id, product_id)
   VALUES (orders_seq.currval, 3, 2381);
```

Related Topics

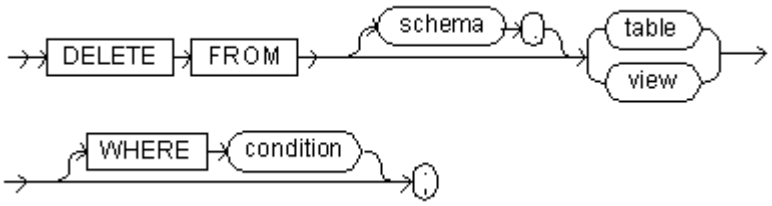
LEVEL pseudocolumn, ROWID pseudocolumn, ROWNUM pseudocolumn

4.3.23 DELETE

Syntax

The syntax for DELETE is displayed in Figure 4–32.

Figure 4–32 The DELETE Command



BNF Notation

```
DELETE FROM [schema .] {table|view}[WHERE condition] ;
```

Prerequisite

You can only delete rows from tables or views in your schema.

Purpose

Removes rows from a table or from a view's base table.

The arguments for the DELETE command are listed in Table 4–27.

Table 4–27 Arguments Used with the DELETE Command

Argument	Description
schema	The schema that contains the table or view. If you omit *schema*, Oracle Lite assumes the table or view is in your own schema.
table	The name of a table from which you want to delete rows.
view	The name of the view. If you specify *view*, Oracle Lite deletes rows from the view's base tables.
WHERE *condition*	Deletes only rows that satisfy a condition specified with the condition argument. For more information about creating a valid condition, see Section 1.7, "Specifying SQL Conditions".

Usage Notes

If no WHERE clause is specified, then all rows of the table are deleted.

A positioned DELETE requires that the cursor be updatable.

Example

```
DELETE FROM PRICE WHERE MINPRICE < 2.4;
```

ODBC 2.0

The ODBC SQL syntax for DELETE is the same as the SQL syntax. In addition, ODBC syntax includes the CURRENT OF *cursor_name* keyword and argument. These are used

in the WHERE clause to specify the cursor where the DELETE operation occurs, as follows:

```
WHERE CURRENT OF cursor_name
```

Related Topics

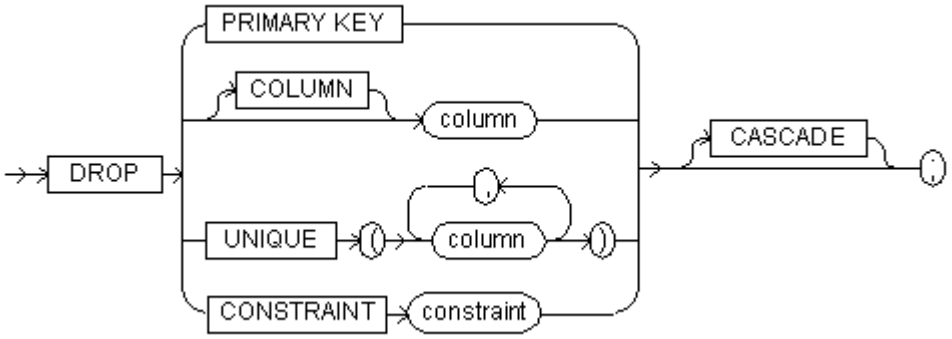
UPDATE

4.3.24 DROP clause

Syntax

The syntax for the DROP clause is displayed in Figure 4–33.

Figure 4–33 The DROP Clause



BNF Notation

```
DROP
{PRIMARY KEY
 | [COLUMN] column
 | UNIQUE "("column")" [, "("column")"]...
 |CONSTRAINT constraint }
[ CASCADE ]  ;
```

Prerequisite

The DROP clause only appears in an ALTER TABLE statement. To drop an integrity constraint, you must be logged into the database as SYSTEM or as a user with DBA/DDL privileges.

Purpose

Removes an integrity constraint from the database.

The arguments for the DROP clause are listed in Table 4–28.

Table 4–28 Arguments Used with the DROP Clause

Argument	Description
PRIMARY KEY	Drops the table's PRIMARY KEY constraint.
UNIQUE	Drops the UNIQUE constraint from the specified columns.
COLUMN	Drops a column from the table.

Table 4–28 (Cont.) Arguments Used with the DROP Clause

Argument	Description
column	Specifies the column from which a column constraint is removed, or in the case of DROP COLUMN, specifies the column to be dropped from the table.
CONSTRAINT	Drops the integrity constraint named constraint. For more information, see "CONSTRAINT clause".
constraint	The name of the integrity constraint to drop.
RESTRICT	If any integrity constraints depend on the constraint to drop, the DROP command fails.
CASCADE	Drops all other integrity constraints that depend on the constraint specified in the CONSTRAINT clause.

Example

```
ALTER TABLE EMP DROP COLUMN COMM;
```

Related Topics

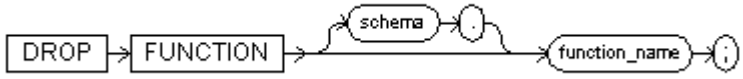
ALTER TABLE, CONSTRAINT clause

4.3.25 DROP FUNCTION

Syntax

The syntax for the DROP function is displayed in Figure 4–34.

Figure 4–34 The DROP Function



BNF Notation

```
DROP FUNCTION [schema .] function_name ;
```

Prerequisite

To drop a function, you must meet one of the following requirements:

- The function must be in your own schema.
- You must be connected to the database as SYSTEM.
- You must have DBA/DDL privileges.

Purpose

To remove a stand alone stored function from the database. For information on creating a function, see "CREATE FUNCTION".

The arguments for the DROP function are listed in Table 4–29.

Table 4–29 Arguments Used with the DROP Function

Argument	Description
schema	The schema containing the function. If you omit schema, Oracle Lite assumes the function is in your own schema.
function_name	The name of the function to drop. Oracle Lite invalidates any local objects that depend on, or call, the dropped function. If you subsequently reference one of these objects, Oracle Lite tries to recompile the object and returns an error if you have not recreated the dropped function.

Example

The following statement drops the PAY_SALARY function, which you created in the CREATE FUNCTION example. When you drop the PAY_SALARY function, you invalidate all objects that depend on PAY_SALARY.

```
DROP FUNCTION PAY_SALARY;
```

Related Topics

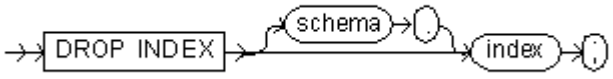
CREATE FUNCTION

4.3.26 DROP INDEX

Syntax

The syntax for `DROP INDEX` is displayed in Figure 4–35.

Figure 4–35 The DROP INDEX Command



BNF Notation

```
DROP INDEX [schema .] index ;
```

Prerequisite

To drop an index, you must be logged into the database as SYSTEM or as a user with DBA/DDL privileges.

Purpose

Removes an index from the database.

The arguments for the DROP INDEX command are listed in Table 4–30.

Table 4–30 Arguments Used with the DROP INDEX Command

Argument	Description
schema	The schema that contains the index to drop. If you omit the schema, Oracle Lite assumes that the index is in your own schema.
index	The name of the index to drop.

Example

The following example drops an index on the SAL column of the EMP table:

```
DROP INDEX SAL_INDEX;
```

Related Topics

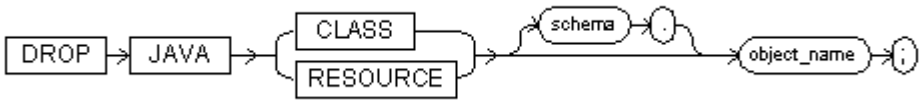
CREATE INDEX

4.3.27 DROP JAVA

Syntax

The syntax for DROP JAVA is displayed in Figure 4–36.

Figure 4–36 The DROP JAVA Command



BNF Notation

```
DROP JAVA { CLASS | RESOURCE } [schema .] object_name;
```

Prerequisite

To drop a class or resource schema object, you must meet the following requirements:

- The Java class, or resource must be in your own schema.
- You must be connected to the database as SYSTEM or have DBA/DDL privileges.

Purpose

To drop a Java class or resource schema object.

For more information on resolving Java classes, and resources, see the *Oracle Database Lite Java Developer's Guide*.

The arguments for the DROP JAVA command are listed in Table 4–31.

Table 4–31 Arguments Used with the DROP JAVA Command

Argument	Description
JAVA CLASS	Drops a Java class schema object.
JAVA RESOURCE	Drops a Java resource schema object.
object_name	Specifies the name of an existing Java class, source, or resource schema object.

Usage Notes

Oracle Lite recognizes *schema_name* when specified, but does not enforce it.

Example

The following statement drops the Java class `MyClass`:

```
DROP JAVA CLASS "MyClass";
```

Related Topics

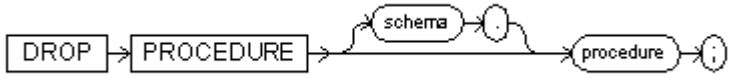
CREATE JAVA

4.3.28 DROP PROCEDURE

Syntax

The syntax for DROP PROCEDURE is displayed in Figure 4–37.

Figure 4–37 The DROP PROCEDURE Command



BNF Notation

```
DROP PROCEDURE [schema .] procedure ;
```

Prerequisite

The procedure must be connected to the database as schema or you must have DBA/DDL privileges.

Purpose

To remove a stand alone stored procedure from the database.

For information on creating a procedure, see "CREATE PROCEDURE".

The arguments for the DROP PROCEDURE command are listed in Table 4–32.

Table 4–32 Arguments Used with the DROP PROCEDURE Command

Argument	Description
schema	The schema containing the procedure. If you omit *schema*, Oracle Lite assumes the procedure is in your own schema.
procedure	The name of the procedure to drop. When you drop a procedure, Oracle Lite invalidates any local objects that depend on the dropped procedure. If you subsequently reference one of these objects, Oracle Lite tries to recompile the object and returns an error message if you have not recreated the dropped procedure.

Example

The following statement drops the procedure TRANSFER owned by the user KERNER and invalidates all objects that depend on TRANSFER:

```
DROP PROCEDURE kerner.transfer
```

Related Topics

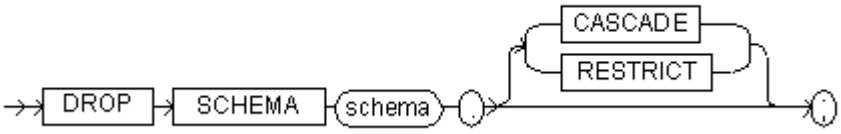
CREATE PROCEDURE

4.3.29 DROP SCHEMA

Syntax

The syntax for DROP SCHEMA is displayed in Figure 4–38.

Figure 4–38 The DROP SCHEMA Command



BNF Notation

```
DROP SCHEMA schema . [{CASCADE | RESTRICT}] ;
```

Prerequisite

To drop a schema, you must be logged into the database as SYSTEM or as a user with DBA/DDL or ADMIN privileges.

Purpose

Removes a schema from the database.

The arguments for the DROP SCHEMA command are listed in Table 4–33.

Table 4–33 Arguments Used with the DROP SCHEMA Command

Argument	Description
schema	The schema to drop from the database.
CASCADE	Specifies that all other objects whose definitions depend on the specified schema are automatically dropped with the schema.
RESTRICT	Specifies that if there are other objects whose definitions depend on the specified schema, the DROP SCHEMA operation fails.

Usage Notes

If no options are specified, the default behavior is determined by the RESTRICT argument.

Example

The following example drops the HOTEL_OPERATION schema you created in the CREATE SCHEMA example:

```
DROP SCHEMA HOTEL_OPERATION CASCADE;
```

Related Topics

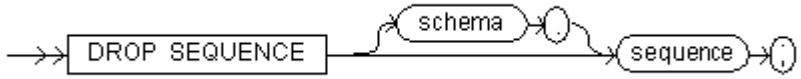
CREATE SCHEMA

4.3.30 DROP SEQUENCE

Syntax

The syntax for DROP SEQUENCE is displayed in Figure 4–39.

Figure 4–39 The DROP SEQUENCE Command



BNF Notation

```
DROP SEQUENCE [schema .] sequence ;
```

Prerequisite

You must be logged into the database as SYSTEM, or the sequence must be in your schema.

Purpose

Removes a sequence from the database.

The arguments for the DROP SEQUENCE command are listed in Table 4–34.

Table 4–34 Arguments Used with the DROP SEQUENCE Command

Argument	Description
schema	The schema that contains the sequence to drop. If you omit schema, Oracle Lite assumes that the sequence is in your own schema.
sequence	The name of the sequence to remove from the database.

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:

- Drop the Sequence.
- Create it with the same name and a START WITH value of 27.

Example

The following example drops the ESEQ sequence you created in the CREATE SEQUENCE example:

```
DROP SEQUENCE ESEQ;
```

ODBC 2.0

Although the DROP SEQUENCE command is not part of the ODBC SQL syntax, ODBC passes the command through to your database.

Related Topics

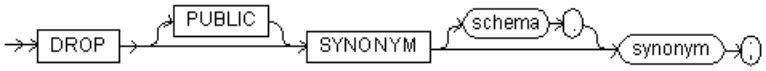
ALTER SEQUENCE, CREATE SEQUENCE

4.3.31 DROP SYNONYM

Syntax

The syntax for DROP SYNONYM is displayed in Figure 4–40.

Figure 4–40 The DROP SYNONYM Command



BNF Notation

```
DROP [PUBLIC] SYNONYM [schema .] synonym ;
```

Prerequisite

To drop a synonym from the database, you must be logged into the database as SYSTEM, or the synonym must be in your schema.

Purpose

Drops a public or private SQL sequence from the database.

The arguments for the DROP SYNONYM command are listed in Table 4–35.

Table 4–35 Arguments Used with the DROP SYNONYM Command

Argument	Description
PUBLIC	Specifies a public synonym. You must specify PUBLIC to drop a public synonym.
schema	The schema to contain the synonym. If you omit schema, Oracle Lite creates the synonym in your own schema. You cannot specify schema if you have specified PUBLIC.
synonym	The name of the synonym to be dropped.

Example

The following example drops the synonym named PROD, which you created in the CREATE SYNONYM example:

```
DROP SYNONYM PROD;
```

Related Topics

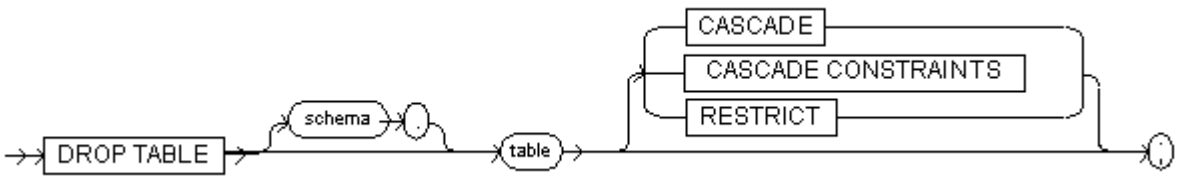
CREATE SYNONYM

4.3.32 DROP TABLE

Syntax

The syntax for DROP TABLE is displayed in Figure 4–41.

Figure 4–41 The DROP TABLE Command



BNF Notation

```
DROP TABLE [schema .] table [{CASCADE | CASCADE CONSTRAINTS | RESTRICT}] ;
```

Prerequisite

To drop a table from the database, you must be logged into the database as SYSTEM or as a user with DBA/DDL privileges.

Purpose

Removes a table from the database.

The arguments for the DROP TABLE command are listed in Table 4–36.

Table 4–36 Arguments Used with the DROP TABLE Command

Argument	Description
schema	The schema that contains the table to drop. If you omit schema, Oracle Lite assumes that the table is in your own schema.
table	The name of the table to remove from the database.
CASCADE	Specifies that, if the table is a base table for views, or if there are referential integrity constraints that refer to primary keys in the table, they are automatically dropped with the table.
CASCADE CONSTRAINTS	Specifies that all referential integrity constraints that refer to primary keys in the table are automatically dropped with the table.
RESTRICT	Specifies that, if the table is a base table for views, or if the table is referenced in any referential integrity constraints, the DROP TABLE operation fails.

Usage Notes

If no options are specified and there are no referential integrity constraints that refer to the table, Oracle Lite drops the table. If no options are specified and there are referential integrity constraints that refer to the table, Oracle Lite returns an error message.

Example

```
DROP TABLE EMP;
```

Related Topics

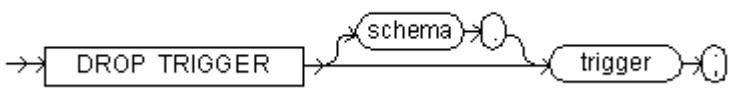
ALTER TABLE, CREATE TABLE

4.3.33 DROP TRIGGER

Syntax

The syntax for DROP TRIGGER is displayed in Figure 4–42.

Figure 4–42 The DROP TRIGGER Command



BNF Notation

```
DROP TRIGGER [schema .] trigger ;
```

Prerequisite

You must be logged into the database as SYSTEM or the trigger must be in your schema.

Purpose

Removes a database trigger from the database.

The arguments for the DROP TRIGGER command are listed in Table 4–37.

Table 4–37 Arguments Used with the DROP TRIGGER Command

Argument	Description
schema	The schema that contains the trigger. If you omit schema, Oracle Lite assumes that the trigger is in your own schema.
trigger	The name of the trigger.

Example

The following statement drops the `SAL_CHECK` trigger, which you created in the CREATE TRIGGER example:

```
DROP TRIGGER ruth.reorder
```

Related Topics

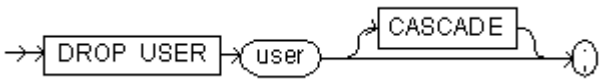
CREATE TRIGGER

4.3.34 DROP USER

Syntax

The syntax for DROP USER is displayed in Figure 4–43.

Figure 4–43 The DROP USER Command



BNF Notation

```
DROP USER user [CASCADE] ;
```

Prerequisite

To drop a user from the database, you must be logged into the database as SYSTEM, or you must have DBA/DDL or ADMIN privileges.

Purpose

Removes a user from the database.

The arguments for the DROP USER command are listed in Table 4–38.

Table 4–38 Arguments Used with the DROP USER Command

Argument	Description
user	Name of the user to be dropped.
CASCADE	Drops all objects associated with the user.

Usage Notes

You can drop users if you are connected to the database as SYSTEM, or if you are granted the ADMIN or DBA/DDL role.

Example

To drop a user when the user's schema does not contain any objects, use the syntax:

```
DROP USER <user>
```

To drop all objects in the user's schema before dropping the user, use the syntax:

```
DROP USER <user> CASCADE
```

The following statement drops the user Michael:

```
DROP USER MICHAEL;
```

Related Topics

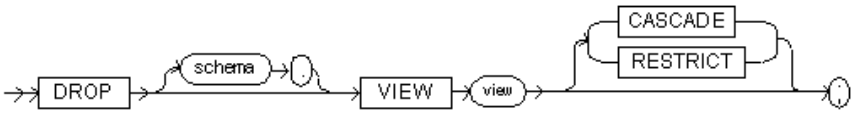
CREATE USER

4.3.35 DROP VIEW

Syntax

The syntax for DROP VIEW is displayed in Figure 4–44.

Figure 4–44 The DROP VIEW Command



BNF Notation

```
DROP [schema .] VIEW view [ {CASCADE | RESTRICT}] ;
```

Prerequisite

To drop a view from the database, you must be logged into the database and you must meet one of the following requirements:

- You must be logged into the database as SYSTEM.
- You must have DBA/DDL privileges.
- The view must be in your schema.

Purpose

Removes a view from the database.

The arguments for the DROP VIEW command are listed in Table 4–39.

Table 4–39 Arguments Used with the DROP VIEW Command

Argument	Description
schema	The schema that contains the view to drop. If you omit schema, Oracle Lite assumes that the view is in your own schema.
view	The name of the view to be removed from the database.
CASCADE	Specifies that all other views whose definitions depend on the specified view are automatically dropped with the view.
RESTRICT	Specifies that if there are other views whose definitions depend on the specified view, the DROP VIEW operation fails.

Usage Notes

If no options are specified, Oracle Lite drops only this view. Other dependent views are not affected.

Example

The following statement drops the EMP_SAL view you created in the CREATE VIEW example:

```
DROP VIEW EMP_SAL;
```

Related Topics

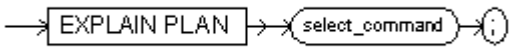
CREATE SYNONYM, CREATE TABLE, CREATE VIEW

4.3.36 EXPLAIN PLAN

Syntax

The syntax for `EXPLAIN PLAN` is displayed in Figure 4–45.

Figure 4–45 The EXPLAIN PLAN Command



BNF Notation

```
EXPLAIN PLAN select_command;
```

Purpose

Displays the execution plan chosen by the Oracle Lite database optimizer for subquery::= statements.

The arguments for the `EXPLAIN PLAN` command are listed in Table 4–40.

Table 4–40 Arguments Used with the EXPLAIN PLAN Command

Argument	Description
EXPLAIN PLAN	Determines an execution plan on a query.
select_command	The query for which you determine the execution plan.

Usage Notes

Oracle Lite outputs the execution plan to a file called **execplan.txt**. Oracle Lite appends each new execution plan to the file.

For every execution of the EXPLAIN PLAN command, Oracle Lite outputs a single line of the EXPLAIN COMMAND followed by one or more lines of the execution plan.

The execution plan contains one line for each query block. A query block begins with a subquery::= keyword.

The plan output is indented to indicate nesting. All siblings of UNION and MINUS are also indented. Each line of the plan output has the following general form:

```
table-name [(column-name)] [{NL(rows)|IL(rows)} table-name [(column-name)] ]
```

The parameters for the EXPLAIN PLAN command are listed in Table 4–41.

Table 4–41 Parameters of the EXPLAIN PLAN Output

Parameter	Definition
table-name	A fully qualified alias or table name.
column-name	The name of the first column of an index key.
NL	Nested loop join.
IL	Index loop join is an index used to join the table following "IL".
(rows)	Indicates the optimizer's estimate of rows for the result of the join.

The tables are executed from left to right. The left-most table forms the outer-most loop of iteration.

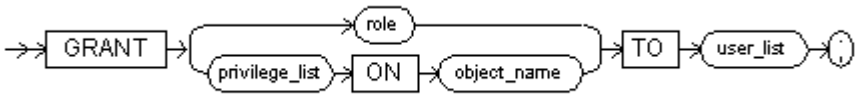
Oracle Lite uses row estimates to order tables, however, the actual values are not important. The optimizer estimates the best possible index. The object kernel may choose a different index since it is more accurate at execution time.

4.3.37 GRANT

Syntax

The syntax for GRANT is displayed in Figure 4–46.

Figure 4–46 The GRANT Command



BNF Notation

```
GRANT {role | privilege_list ON object_name} TO user_list ;
```

Prerequisite

To grant roles, you must be logged into the database as SYSTEM, or as a user with DBA/DDL and ADMIN privileges, or with RESOURCE privileges to GRANT privilege on your own objects to other users.

Purpose

Grants the ADMIN, DBA, DDL, or RESOURCE roles to users, or grants privileges on a database object to users. The DBA role is recommended as a replacement for the DDL role wherever possible.

The arguments for the `GRANT` command are listed in Table 4–42.

Table 4–42 Arguments Used with the GRANT Command

Argument	Description
role	The UNRESOLVED XREF TO ADMIN, UNRESOLVED XREF TO DBA/DDL, or UNRESOLVED XREF TO RESOURCE role.
user_list	One user, or a comma-separated list of users.
ON	Signifies the database object to which you grant roles.
privilege_list	Either a comma-separated list of the following privileges or a combination called ALL: INSERT, DELETE, UPDATE (*col_list*), SELECT, and REFERENCES.
TO	Signifies the users or user list to whom you grant roles.
object_name	A table name optionally prefixed with a schema name.

Pre-defined Roles

Oracle Lite combines some privileges into pre-defined roles for convenience. In many cases it is easier to grant a user a pre-defined role than to grant specific privileges in another schema. Oracle Lite does not support creating or dropping roles. The Oracle Lite pre-defined roles are listed in Table 4–43:

Table 4–43 Predefined Roles in Oracle Database Lite

Role Name	Privileges Granted To Role
ADMIN	Enables the user to create other users and grant privileges other than DDL and ADMIN on any object in the schema. The user can execute any of the following commands in a SQL statement: CREATE SCHEMA, CREATE USER, ALTER USER, DROP USER, DROP SCHEMA, GRANT, and REVOKE.
DBA/DDL	Enables the user to issue the following DDL statements which otherwise can only be issued by SYSTEM: All ADMIN privileges, CREATE TABLE, CREATE ANY TABLE, CREATE VIEW, CREATE ANY VIEW, CREATE INDEX, CREATE ANY INDEX, ALTER TABLE, ALTER VIEW, DROP TABLE, DROP VIEW, and DROP INDEX.
RESOURCE	The RESOURCE role grants the same level of control as the DBA/DDL role, but only over the user's own domain. The user can execute any of the following commands in a SQL statement: CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE CONSTRAINT, ALTER TABLE, ALTER VIEW, ALTER INDEX, ALTER CONSTRAINT, DROP TABLE, DROP VIEW, DROP INDEX, DROP CONSTRAINT, and GRANT or REVOKE privileges on any object under a user's own schema.

Usage Notes

If *privilege_list* is ALL, then the user can INSERT, DELETE, UPDATE, or SELECT from the table or view. If *privilege_list* is either INSERT, DELETE, UPDATE, or SELECT, then the user has that privilege on a table.

When you grant UPDATE on a table to a user and then subsequently alter the table by adding a column, the user is not able to update the new column. The user can only update the new column if you issue a grant statement after creating the new column. For example:

```
CREATE TABLE t1 (c1 NUMBER c2 INTEGER);
CREATE USER a IDENTIFIED BY a;
GRANT SELECT, UPDATE ON t1 TO a;
ALTER TABLE t1 ADD c3 INT;
COMMIT;
```

In the preceding example, the GRANT statement must be issued after the ALTER TABLE statement or the user cannot update the new column, c3.

Example 1

The following example creates a user named MICHAEL and grants the user the ADMIN role:

```
CREATE USER MICHAEL IDENTIFIED BY SWORD;

GRANT ADMIN TO MICHAEL;
```

Example 2

The following example creates a user named MICHAEL and grants INSERT and DELETE privileges on the EMP table the user.

```
CREATE USER MICHAEL IDENTIFIED BY SWORD;

GRANT INSERT, DELETE ON EMP TO MICHAEL;
```

Example 3

The following example grants ALL privileges on the PRODUCT table to the newly created user, MICHAEL:

```
GRANT ALL ON PRODUCT TO MICHAEL;
```

Related Topics

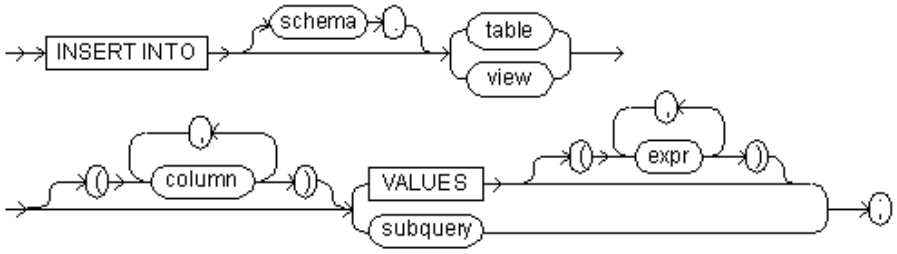
REVOKE

4.3.38 INSERT

Syntax

The syntax for `INSERT` is displayed in Figure 4–47.

Figure 4–47 The INSERT Command



BNF Notation

```
INSERT INTO [schema .] {table | view }
["("column [, column]...")"]
{ VALUES  "(" expr [, expr]...")" | subquery} ;
```

Prerequisite

To insert rows into a table or view, you must be logged into the database as SYSTEM, or the table and view must be in your schema.

Purpose

Adds rows to a table or to a view's base table.

The arguments for the `INSERT` command are listed in Table 4–44.

Table 4–44 Arguments Used with the INSERTCommand

Argument	Description
schema	The schema that contains the table or view. If you omit schema, Oracle Lite assumes that the table or view is in your own schema.
table	The name of the table into which you want to insert rows.
view	The name of the view into whose base tables you want to insert rows.
column	A column of a table or view. In the inserted row, each column listed in this argument is assigned a value from the VALUES clause or from the subquery. If you omit one of the table's columns from this argument, the column's value for the inserted row is the column's default value as specified when the table is created. If you omit the column argument, the VALUES clause or the query must specify values for all columns in the table.
VALUES	Specifies a row of values to be inserted into the table or view. You specify in the VALUES clause a value for each column in the column argument.
expr	The values assigned to the corresponding column. This can contain host variables. For more information, see Section 1.8, "Specifying Expressions".
subquery	A SELECT statement that returns rows that are inserted into the table. The SELECT list of this subquery must have the same number of columns as the column list of the INSERT statement.

Usage Notes

- The same column name may not appear more than once in the column argument.
- If you omit any columns from the column argument, Oracle Lite assigns the columns the default values specified when the table is created.

- The number of columns specified in the column argument must be the same as the number of values provided. If you omit the column argument, the number of values must be equal to the degree of the table.
- If a column does not have a user-defined default value, its default value is NULL. This is true even when there is a NOT NULL constraint on the column. If an INSERT statement does not provide an explicit value for such a column, Oracle Lite generates an integrity violation error message.

Example

```
INSERT INTO EMP (EMPNO, ENAME, DEPTNO) VALUES ('7010', 'VINCE', '20');
```

Related Topics

DELETE, UPDATE

4.3.39 LEVEL pseudocolumn

Purpose

The LEVEL pseudocolumn can be used in a SELECT statement that performs a hierarchical query. 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. In a hierarchical query, 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, and a leaf node is any node without children.

Prerequisites

None.

Usage Notes

The number of levels returned by a hierarchical query is limited to 32.

Example

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;
```

Returns the following result:

```
ORG_CHART              EMPNO       MGR JOB
------------------ --------- --------- ---------
                        7839           PRESIDENT
    JONES               7566      7839 MANAGER
    SCOTT               7788      7566 ANALYST
    ADAMS               7876      7788 CLERK
    FORD                7902      7566 ANALYST
    SMITH               7369      7902 CLERK
    CLARK               7782      7839 MANAGER
    MILLER              7934      7782 CLERK
```

```
BLAKE              7698      7839 MANAGER
WARD               7521      7698 SALESMAN
JAMES              7900      7698 CLERK
TURNER             7844      7698 SALESMAN
ALLEN              7499      7698 SALESMAN
MARTIN             7654      7698 SALESMAN

14 rows selected.
```

Related Topics

CURRVAL and NEXTVAL pseudocolumns, OL__ROW_STATUS pseudocolumn, ROWID pseudocolumn, ROWNUM pseudocolumn,

4.3.40 OL__ROW_STATUS pseudocolumn

Purpose

For each row in the database, the OL__ROW_STATUS pseudocolumn returns the status of a row from a snapshot table: new, updated, or clean.

Prerequisite

None.

Usage Notes

OL__ROW_STATUS enables you to select the column from any snapshot or regular table, but row status information is only returned for snapshot table rows. Regular table rows return the same value regardless of status.

The OL__ROW_STATUS pseudocolumn can be qualified with the table name in the same manner as other pseudocolumns. Thus you can determine row status in complex queries involving multiple tables as listed in Table 4–45.

Table 4–45 OL__ROW_STATUS Results

Table Type	OL__ROW_STATUS value	Description
Snapshot table	0	The row is clean or not dirty.
Snapshot table	16	The row is a new row created at the client side.
Snapshot table	32	The row has been updated.
Regular table	0	This value is static and never changes.

Example 1

```
Select OL__ROW_STATUS, Emp.* from Employee Emp Where Empno = 7900;
```

Example 2

```
Select Emp. OL__ROW_STATUS, ENAME, DNAME  from EMP,DEPT where
DEPT.DEPTNO=EMP.DEPTNO AND EMP.EMPNO=7900;
```

Related Topics

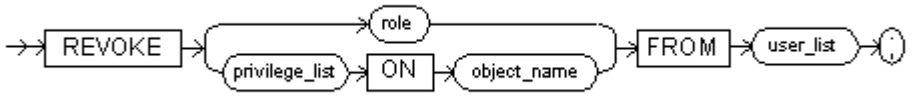
CURRVAL and NEXTVAL pseudocolumns, LEVEL pseudocolumn, ROWID pseudocolumn, ROWNUM pseudocolumn

4.3.41 REVOKE

Syntax

The syntax for REVOKE is displayed in Figure 4–48.

Figure 4–48 The REVOKE Command



BNF Notation

```
REVOKE  { role | privilige_list ON object_name } FROM user_list ;
```

Prerequisite

To revoke roles from users, you must be logged into the database as SYSTEM or as a user with DBA or ADMIN privileges.

Purpose

Revokes the ADMIN, DBA/DDL, or RESOURCE roles from users, or revokes privileges on a database object from users. The DBA role is recommended as a replacement for the DDL role.

The arguments for the REVOKE command are listed in Table 4–46.

Table 4–46 Arguments Used with the REVOKE Command

Argument	Description
role	The UNRESOLVED XREF TO ADMIN, UNRESOLVED XREF TO DBA/DDL, or UNRESOLVED XREF TO RESOURCE role.
user_list	One user, or a comma-separated list of users.
privilege_list	A comma-separated list of the following privileges or a combination called ALL: INSERT, DELETE, UPDATE (*col_list*), and SELECT.
object_name	A table name prefixed with a schema name.

Usage Notes

If *privilege_list* contains INSERT, DELETE, UPDATE, or SELECT, then the user has those privileges on a table or view. If *privilege_list* is ALL, then the user can INSERT, DELETE, UPDATE, or SELECT from the table or view.

Example 1

The following example creates a user named STEVE and grants the user the ADMIN role. Then, the example revokes the ADMIN role from the user, STEVE.

```
CREATE USER STEVE IDENTIFIED BY STINGRAY;
GRANT ADMIN TO STEVE;
REVOKE ADMIN FROM STEVE;
```

Example 2

The following example revokes the INSERT and DELETE privileges on the EMP table from the user, SCOTT.

```
REVOKE INSERT,DELETE ON EMP FROM SCOTT;
```

Example 3

The following example creates a user named CHARLES and grants the user the INSERT and DELETE privileges on the PRICE table, and ALL privileges on the ITEM table. Then the example revokes all privileges for the user CHARLES on the PRICE and ITEM tables.

```
CREATE USER CHARLES IDENTIFIED BY VORTEX;
GRANT INSERT, DELETE, UPDATE ON PRICE TO CHARLES;
GRANT ALL ON ITEM TO CHARLES;
REVOKE ALL ON PRICE FROM CHARLES;
REVOKE ALL ON ITEM FROM CHARLES;
```

Related Topics

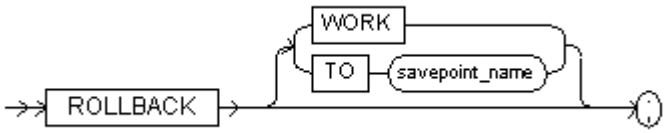
GRANT

4.3.42 ROLLBACK

Syntax

The syntax for ROLLBACK is displayed in Figure 4–49.

Figure 4–49 The ROLLBACK Command



BNF Notation

```
ROLLBACK [{ WORK | TO savepoint_name }] ;
```

Prerequisite

None.

Purpose

Undoes work performed in the current synonym.

The arguments for the ROLLBACK command are listed in Table 4–47.

Table 4–47 Arguments Used with the ROLLBACK Command

Argument	Description
work	An optional argument supported to provide ANSI compatibility.
TO	An optional argument that enables you to roll back to a savepoint.
savepoint_name	The name of the savepoint you roll back to.

Usage Notes

If you are not already in a transaction, Oracle Lite starts one the first time you issue a SQL statement. All the statements you issue are considered part of the transaction until you use a COMMIT or ROLLBACK command.

The COMMIT command makes permanent changes to the data in the database, saving everything up to the start of the transaction. Before changes are committed, both the old and new data exist so that changes can be stored or the data can be restored to its prior state.

The ROLLBACK command discards pending changes made to the data in the current transaction, restoring the database to its state before the start of the transaction. You can ROLLBACK a portion of a transaction by identifying a SAVEPOINT.

Important: Oracle Lite does *not* automatically commit DDL commands, except for CREATE DATABASE. DDL commands in Oracle Lite are subject to rollback.

Example

The following example inserts a new row into the DEPT table and then rolls back the transaction. This example returns the same results for both ROLLBACK and ROLLBACK WORK.

```
INSERT INTO DEPT (deptno, dname, loc) VALUES (50, 'Design', 'San Francisco');
SELECT * FROM dept;
```

Returns the following result:

```
   DEPTNO DNAME          LOC
--------- -------------- -------------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
       40 OPERATIONS     BOSTON
       50 DESIGN         SAN FRANCISCO

ROLLBACK WORK;
SELECT * FROM dept;
```

Returns the following result:

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

ODBC 2.0

Although the ROLLBACK command is not part of the ODBC SQL syntax, ODBC passes the command through to your database.

An ODBC program typically uses the API call `SQLTransact()` with the `SQL_ROLLBACK` flag.

Related Topics

SAVEPOINT

4.3.43 ROWID pseudocolumn

Purpose

For each row in the database, the ROWID pseudocolumn returns a row address. A ROWID value uniquely identifies a row in the database. Values of the ROWID pseudocolumn have the datatype ROWID.

Prerequisite

None.

Usage Notes

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.

You should not use ROWID as a table's primary key. If you delete and reinsert a row with the Import and Export utilities, for example, its rowid may change. If you delete a row, Oracle Database Lite may reassign its ROWID to a new row inserted later.

Although you can use the ROWID pseudocolumn in the SELECT and WHERE clause 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 1

This statement selects the address of all rows that contain data for employees in department 20:

```
SELECT ROWID, last_name
   FROM employees
   WHERE department_id = 20;
```

Related Topics

CURRVAL and NEXTVAL pseudocolumns, LEVEL pseudocolumn, ROWNUM pseudocolumn, OL__ROW_STATUS pseudocolumn

4.3.44 ROWNUM pseudocolumn

Purpose

For each row returned by a query, the ROWNUM pseudocolumn returns a number indicating the order in which Oracle Lite 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.

Prerequisite

None.

Usage Notes

If an ORDER BY clause follows ROWNUM in the same subquery, the rows are reordered by the ORDER BY clause. The results can vary depending on the way the rows are accessed. For example, if the ORDER BY clause causes Oracle Lite to use an index to access the data, Oracle Lite may retrieve the rows in a different order than without the index.

If you embed the ORDER BY clause in a subquery and place the ROWNUM condition in the top-level query, you can force the ROWNUM condition to be applied after the ordering of the rows. See Example 3.

Example 1

The following example uses ROWNUM to limit the number of rows returned by a query:

```
SELECT * FROM emp WHERE ROWNUM < 10;
```

Example 2

The following example follows the ORDER BY clause with ROWNUM in the same query. As a result, the rows are reordered by the ORDER BY clause and do not have the same effect as the preceding example:

```
SELECT * FROM emp WHERE ROWNUM < 11 ORDER BY empno;
```

Example 3

The following query returns the ten smallest employee numbers. This is sometimes referred to as a "top-N query":

```
SELECT * FROM
   (SELECT empno FROM emp ORDER BY empno)
   WHERE ROWNUM < 11;
```

Example 4

The following query returns no rows:

```
SELECT * FROM emp WHERE ROWNUM > 1;
```

The first fetched row 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, this makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

Example 5

The following statement assigns unique values to each row of a table:

```
UPDATE tabx SET col1 = ROWNUM;
```

Related Topics

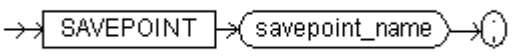
CURRVAL and NEXTVAL pseudocolumns, LEVEL pseudocolumn, ROWID pseudocolumn, OL__ROW_STATUS pseudocolumn

4.3.45 SAVEPOINT

Syntax

The syntax for `SAVEPOINT` is displayed in Figure 4–50.

Figure 4–50 The SAVEPOINT Command



BNF Notation

```
SAVEPOINT savepoint_name ;
```

Purpose

To identify a point in a transaction to which you can later roll back.

Prerequisites

None.

Usage Notes

Once you set a savepoint you can either roll back to it or remove it later. To roll back to a savepoint use the statement:

```
ROLLBACK TO <savepoint_name>
```

To remove a savepoint use the statement:

```
REMOVE SAVEPOINT <savepoint_name>
```

When you roll back to remove a savepoint, all nested savepoints are also rolled back or removed. Savepoints should be removed as soon as possible to reduce memory usage.

A user defined savepoint enables you to name and mark the current point in the processing of a transaction. Used with ROLLBACK, SAVEPOINT lets you undo parts of a transaction instead of the entire transaction. When you roll back to a savepoint, any savepoint marked after that savepoint is erased. The COMMIT statement erases any savepoints marked since the last commit or rollback.

The number of *active* savepoints you define for each session is unlimited. An active savepoint is one marked since the last commit or rollback.

Example

The following example updates the salary for two employees, Blake and Clark. It then checks the total salary in the EMP table. The example rolls back to savepoints for each employee's salary, and updates Clark's salary.

```
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, SAVEPOINT, ROLLBACK

4.3.46 SELECT

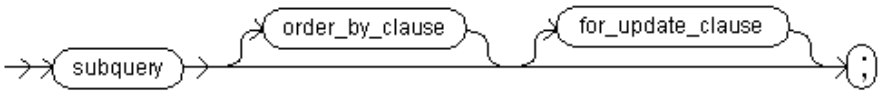
The SELECT statement retrieves data from one or more tables or views. You can also use the select statement to invoke Java stored procedures. To select data from a table or view, you must be logged into the database as SYSTEM, or the table(s) and view(s) must be part of your schema.

Syntax

`select::=`

The syntax for `SELECT` is displayed in Figure 4–51.

Figure 4–51 The SELECT Command



BNF Notation

```
subquery [order_by_clause] [ for_update_clause] ;
```

Related Topics

CONSTRAINT clause, DELETE, UPDATE

The following sections describe the different operations you can use within a select statement:

- Section 4.3.46.1, "SELECT Command Arguments"
- Section 4.3.46.2, "The SUBQUERY Expression"
- Section 4.3.46.3, "The FOR_UPDATE Clause"
- Section 4.3.46.4, "The ORDER_BY Clause"
- Section 4.3.46.5, "The TABLE_REFERENCE Expression"
- Section 4.3.46.6, "The ODBC_JOIN_TABLE Expression"
- Section 4.3.46.7, "The JOINED_TABLE Expression"
- Section 4.3.46.8, "The HINT Expression"
- Section 4.3.46.9, "The LIMIT and OFFSET Clauses"

4.3.46.1 SELECT Command Arguments

The arguments for the `SELECT` command are listed in Table 4–48.

Table 4–48 Arguments Used with the SELECT Command

Argument	Description
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.

Table 4–48 (Cont.) Arguments Used with the SELECT Command

Argument	Description
*	Selects all columns from all tables, views, or snapshots listed in the FROM clause.
table.*	Selects all columns from the selected table. Use the *schema* qualifier to select from a schema other than your own.
view.*	Selects all columns from the selected view. Use the schema qualifier to select from a schema other than your own.
expr	Selects an expression, usually based on column values, from one of the tables or views in the FROM clause. A column name in this list can be qualified with a schema only if the table or view that contains the column is itself qualified with a schema in the FROM clause. For more information, see Section 1.8, "Specifying Expressions".
hint	Hints are processed by the Oracle Database Lite optimizer to suggest choices for statement execution. See "The HINT Expression" for more information.
/*+ ... */	Hint processed by both Oracle and Oracle Database Lite.
/*% ...%*/	Hint processed as a comment in Oracle, processed by Oracle Database Lite.
// ... //	Hint processed by both Oracle and Oracle Database Lite.
c_alias	Provides a column alias, which is a different name for the column expression, and causes the column alias to be used in the column heading. A column alias does not affect the actual name of the column. The alias can only be used in the ORDER BY clause. It cannot be used by other clauses in the query.
schema	The schema that contains the selected table, view, or snapshot. If you omit *schema*, Oracle Lite assumes that the table, view, or snapshot resides in your own schema.
table	The table from which data is selected.
view	The view from which data is selected
t_alias	Provides a different name or alias for the table, view, or snapshot, for evaluating the query. 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 specified condition is TRUE. If you omit the WHERE clause, Oracle Lite returns all rows from the tables, views, or snapshots in the FROM clause. WHERE specifies a conditional expression that evaluates to TRUE or FALSE. For more information, see Section 1.8, "Specifying Expressions".
condition	A search condition. For more information about creating a valid condition, see Section 1.7, "Specifying SQL Conditions".
START WITH	Returns rows in a hierarchical order.
CONNECT BY	Specifies the relationship between parent and child rows in a hierarchical query. The condition defines this relationship, and must use the PRIOR operator to refer to the parent row. To find the children of the parent row, Oracle Lite evaluates the PRIOR expression for each row in the table. Rows for which the condition is TRUE are the children of the parent. For more information, see the details of the PRIOR operator in Section 2.7, "Other Operators".

Table 4–48 (Cont.) Arguments Used with the SELECT Command

Argument	Description
GROUP BY	Groups the selected rows based on the value of the *expr* argument for each row, and returns a single row of summary information for each group.
HAVING	Restricts the groups of rows returned to those groups for which the specified condition is TRUE. If you omit this clause, Oracle Lite returns summary rows for all groups. For more information, see Section 1.7, "Specifying SQL Conditions".
INTERSECT	Returns all distinct rows selected by both queries. INTERSECT has a higher precedence than UNION.
INTERSECT ALL	Returns all distinct rows selected by both queries, the same result as INTERSECT. This syntax is supported, but has no function.
UNION	Returns all distinct rows selected by either query.
UNION ALL	Returns all rows selected by either query, including duplicates.
MINUS	Returns all distinct rows selected by the first query but not the second.
command	Refers to all parameters of a SELECT command which is itself a parameter of another SELECT command. When entering parameters for a SELECT command within a SELECT command, you cannot use the WHERE statement.
ORDER BY	Orders rows returned by the SELECT statement, according to the following arguments: *expr* (expression) orders rows based on their value for *expr*. The expression is based on columns in the select list, or based on columns in the tables, views, or snapshots in the FROM clause. *position* orders rows based on their value for the expression in this position in the select list. ASC specifies an ascending sort order. ASC is the default. DESC specifies a descending sort order.
FOR UPDATE	Locks the selected rows. The column list in the FOR UPDATE clause is ignored. The FOR UPDATE clause can be used either before or after the ORDER BY clause.
column	The column to be updated.

Usage Notes

If you do not specify a WHERE clause and there is more than one table in the FROM clause, Oracle Lite computes a Cartesian product of all the tables involved.

You can use the LEVEL pseudocolumn in a SELECT statement to perform a hierarchical query. For more information, see LEVEL pseudocolumn. A hierarchical query cannot perform a join, nor can it select data from a view.

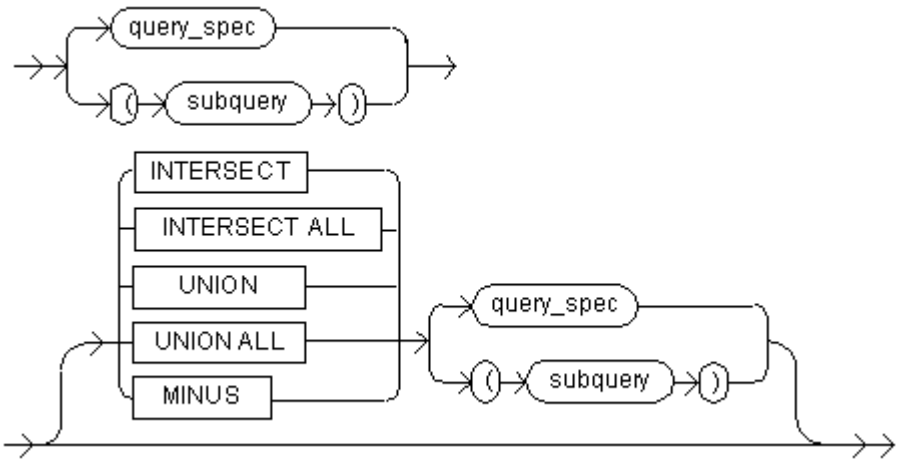
When you select columns with an expression, those columns must have an alias. An alias specifies names for the column expressions selected by the query. The number of aliases must match the number of expressions selected by the query. Aliases must be unique within the query.

4.3.46.2 The SUBQUERY Expression

subquery::=

The syntax for the `subquery` expression is displayed in Figure 4–52.

Figure 4–52 The subquery Expression



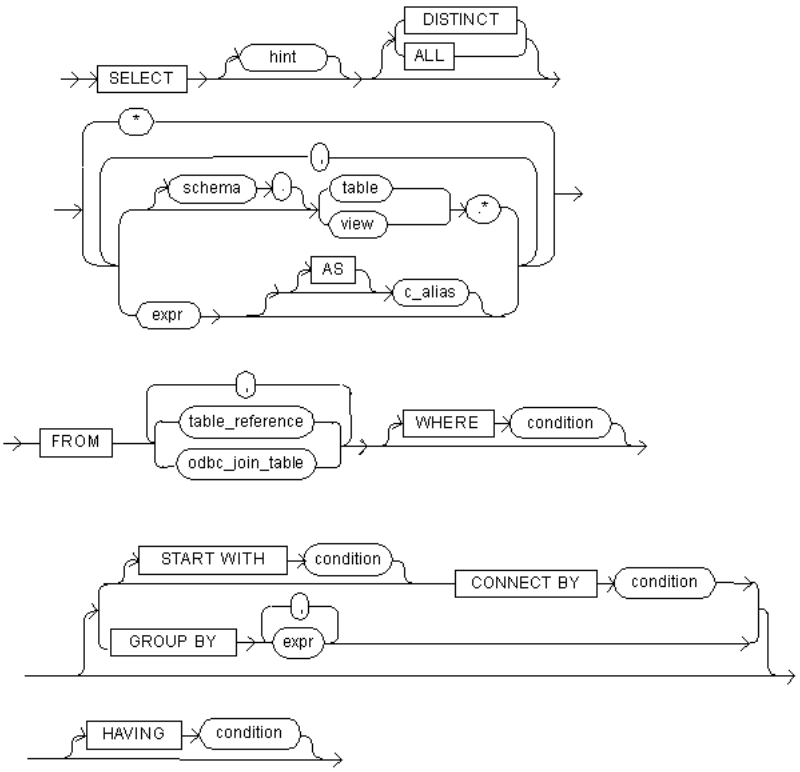
BNF Notation

```
{query_spec | "("subquery")" }
 [{ INTERSECT | INTERSECT ALL | UNION | UNION ALL | MINUS }
  {query_spec |"(" subquery ")" } ]
```

query_spec::=

The syntax for the `query_spec` expression is displayed in Figure 4–53.

Figure 4–53 The query spec Expression



BNF Notation

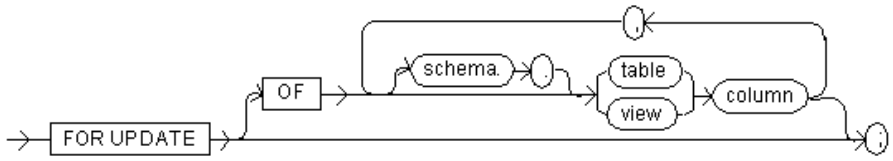
```
SELECT [ hint ] [ { DISTINCT | ALL ]
{ *
| { [schema.] { table | view } .*
     | expr [[AS] c_alias]
  }
  [, {
     | [schema .] { table | view  } .*
     | expr [[AS] c_alias]
     }
  ]...
}
FROM  [schema .] { "("subquery [order_by_clause] ")" | table | view  }
[ t_alias ] [ WHERE condition]
[
 { [ START WITH condition ] CONNECT BY condition
 | GROUP BY expr [, expr]...
 | [HAVING condition]
  }]
```

4.3.46.3 The FOR_UPDATE Clause

for_update_clause::=

The syntax for the `update_clause` expression is displayed in Figure 4–54.

Figure 4–54 The for_update_clause Expression



BNF Notation

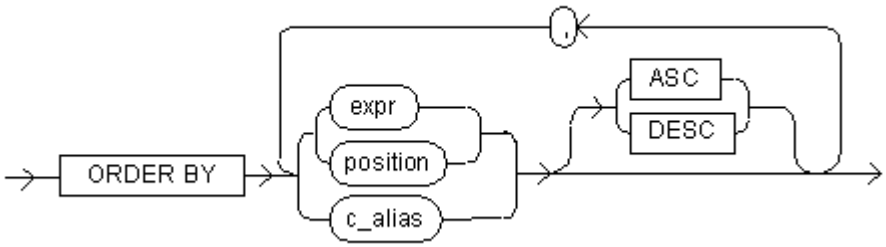
```
FOR UPDATE
[OF [[schema .] { table | view } .] column
 [, [[schema .] { table | view } .] column]...]
```

4.3.46.4 The ORDER_BY Clause

order_by_clause::=

The syntax for the `order_by_clause` expression is displayed in Figure 4–55.

Figure 4–55 The order_by_clause Expression



BNF Notation

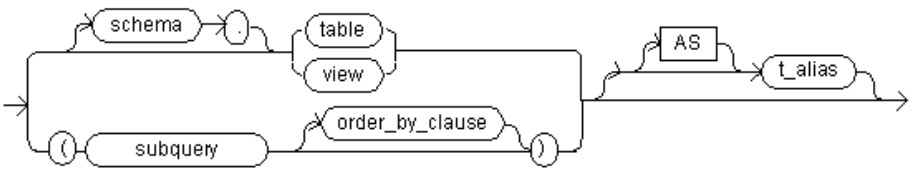
```
ORDER  BY
{ expr | position | c_alias } [ ASC | DESC ]
[, { expr | position | c_alias } [ ASC | DESC ] ]...
```

4.3.46.5 The TABLE_REFERENCE Expression

table_reference::=

The syntax for the `table_reference` expression is displayed in Figure 4–56.

Figure 4–56 The table_reference Expression



BNF Notation

```
{ [schema .] {table | view}
 | "("subquery [order_by_clause] ")"
} [[AS] t_alias]
```

4.3.46.6 The ODBC_JOIN_TABLE Expression

odbc_join_table::=

The syntax for the `odbc_join_table` expression is displayed in Figure 4–57.

Figure 4–57 The odbc_join_table Expression

BNF Notation

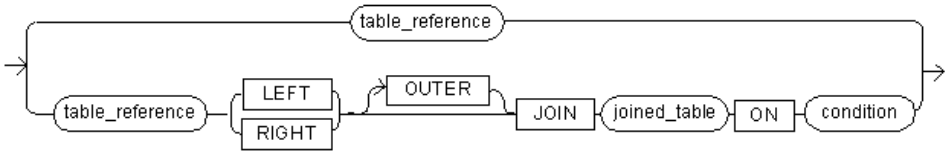
```
"{" OJ joined_table "}"
```

4.3.46.7 The JOINED_TABLE Expression

joined_table::=

The syntax for the `joined_table` expression is displayed in Figure 4–58.

Figure 4–58 The join_table Expression



BNF Notation

```
"{"
  { table_reference
   | OJ table_refernce { LEFT | RIGHT } [OUTER] JOIN joined_table ON conditon
  }
"}"
```

4.3.46.8 The HINT Expression

You can use comments in a SQL statement to pass instructions, or hints, to the Oracle Database Lite optimizer. The optimizer uses these hints as suggestions for choosing an execution plan for the statement.

A statement block can have only one comment containing hints, and that comment must follow the SELECT, UPDATE, INSERT, or DELETE keyword. The following syntax shows hints contained in the styles of comments that Oracle Database Lite supports within a statement block.

```
{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */
```

or

```
{DELETE|INSERT|SELECT|UPDATE} // hint [text] [hint[text]]... //
```

or

```
{DELETE|INSERT|SELECT|UPDATE} /*% hint [text] [hint[text]]...%*/
```

Where:

DELETE, INSERT, SELECT or UPDATE is a DELETE, INSERT, SELECT or UPDATE keyword that beings a statement block. Comments containing hints ca nappear only after these keywords. the `/*+`, `//`, or `/*%` causes Oracle to interpret the comment as a list of hints. The plus sign must follow immediately after the comment delimiter and no space is permitted. However, the space between the plus sign and the hint is optional. If the comment contains multiple hints, then separate the hints by at least one space.

The text is other commenting text that can be interspersed with the hints. Oracle Database Lite treats misspelled hints as regular comments and does not return an error.

To share the same code between Oracle Database Lite and Oracle database and to specify a hint to Oracle Database Lite only, use the syntax `/*% hint %*/`. To give hints to both Oracle Database Lite and Oracle optimizers, use the syntax `/*+ hint */`.

4.3.46.8.1 ORDERED Hints The ORDERED hint causes Oracle Database Lite to join tables in the order in which they appear in the FROM clause. If you omit the ORDERED hint from a SQL statemetn performing a join, then the optimizer chooses the order in which to join the tables. You can use the ORDERED hint to specify a join order if you know how the number of rows are selected from each table. You can choose an inner and outer table for best performance.

```
ordered_hint::=/*+ ORDERED */
```

The following query is an example of the use of the ORDERED hint:

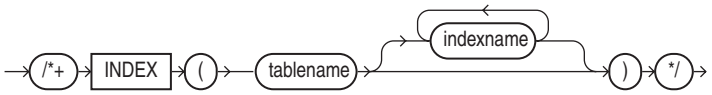
```
SELECT /*+ORDERED */ o.order_id, c.customer_id, 1.unit_price * 1.quantity
FROM customers c, order_items 1, orders o
WHERE c.cust_last_name = ?
AND o.customer_id = c.customer_id
AND o.order_id = 1.order_id;
```

4.3.46.8.2 INDEX Hints Index hints explicitly choose an index scan for the specified table. The following are Index hints:

- INDEX
- INDEX_ASC
- INDEX_DESC

Each INDEX hint is fully described in the Oracle Database SQL Reference.

The INDEX hint explicitly chooses an index scan for the specified table.




```
index_hint::= table_name index_name
```

where

- index specifies an index name
- table specifies the name or alias of the table

Either name cannot be a qualified name, such as SYSTEM.EMP. Only one index_name can be provided for a given table_name. If you provide more than one index name,then only the first one is selected for optimization.

> **Note:** For full details on the INDEX hint, see the Oracle Database SQL Reference.

For example:

```
SELECT /*+ INDEX (employees emp_department_ix)*/
       employee_id, department_id
  FROM employees
  WHERE department_id > 50;
```

4.3.46.9 The LIMIT and OFFSET Clauses

Because client devices have software and hardware limitations—such as CPU, memory, screen size, and so on—you may wish to limit the number of rows returned from your SQL query, especially if the returned result set contains a huge number of rows. The retrieval of all rows could take a long time to complete and affect performance. Also, your application may not be able to display all results, due to the limitation of the device, the requirement of the business logic, or the slow response time of the query.

You can limit the number of rows returned by a query, as follows:

- LIMIT clause: Enables you to return only a specified number of rows, so that you do not overwhelm the limitations of your device or application.
- OFFSET clause: Enables you to start at a certain point within the returned result set.
- ORDER BY clauses: Enables you to retrieve rows in a specified order.
- Creating indexes: If you create the right indexes, the performance can be improved significantly for small devices.

Syntax

```
Cursor_spec::=subquery [order_by_clause][for_update_clause][limit_clause]
subquery::= see Section 4.3.46.2, "The SUBQUERY Expression" for more details
limit_clause::={LIMIT number [offset_clause] | offset_clause}
offset_clause::=OFFSET number
```

The LIMIT clause can be used to limit the number of rows returned by a query. LIMIT takes an integer constant between 0 and 4294967295, which specifies the maximum number of rows to return. The OFFSET clause takes an integer constant between 0 and 4294967295, which specifies the offset of the first row to return. If OFFSET clause is not present, it defaults to 0.

For example, the following SQL statement retrieves rows from 5 to 9:

```
SELECT * FROM table LIMIT 5 OFFSET 4;
```

With only the LIMIT argument, the value specifies the number of rows to return from the beginning of the result set. The following SQL statement retrieves rows from 1 to 5;

```
SELECT * FROM table LIMIT 5;
```

If the LIMIT argument is 0, the OFFSET value is ignored even if it was specified. The following SQL statement retrieves nothing:

```
SELECT * FROM table LIMIT 0 OFFSET 4;
```

If only the OFFSET clause is present, then there is not a limit on the number of rows returned. The following SQL statement retrieves rows starting from the second row of the result set:

```
SELECT * FROM table OFFSET 1;
```

You can use the ORDER BY clause together with LIMIT clause to constrain the order of the output rows. That is, when both the LIMIT and ORDER BY clauses are present in a statement, then the optimizer takes this into account when generating the execution plan. By creating indexes on the ORDER BY column(s), you can avoid inserting the whole result set into a temporary table and performing the sorting just to retrieve a few rows from the query. The EXPLAIN PLAN command can be used to see wheather a sorting is performed when LIMIT and ORDER BY are used in a query. See Section 1.11, "Tuning SQL Statement Execution Performance With the EXPLAIN PLAN" for more information on the EXPLAIN PLAN.

Limit and Offset Clause Example

A customer uses an order entry application, where there is a product table with over 3,000 rows with a primary index on the product number. The user can select an individual product by scanning a barcode with a scanner, or by entering a product number manually in a text field. The script opens a cursor to select one product using the barcode or product number as an equality selection (both are indexed). In this case, Oracle Database Lite performs well. However, the database access is very slow in other actions. After a product is selected, the user can click a "next" or "prev" button to find the next or previous product number, with product number being the primary index. This is necessary because the customer often wants to view related items with similar product numbers.

The SQL statement when user clicks a "next" button is as follows:

```
SELECT * FROM PRODUCT WHERE PARTNUM > partnum ORDER BY PARTNUM;
```

Where partnum is the product number scanned or entered by the end user.

When the current product is the first one (in the index) doing a "next" takes a long time, since there are more than 3,000 rows that need to be sorted and returned by this query. On the other hand, the actual SQL statement when the user clicks a "prev" button is similar to the one above. In addition, when the current product is the last one or near the end of the product table, the response time is also slow for the same reason.

```
SELECT * FROM PRODUCT WHERE PARTNUM < partnum ORDER BY PARTNUM DESC;
```

Where partnum is the product number scanned or entered by the end user.

What the customer wants is a SELECT statement that will do the equivalent of "find the first few products where `partnum > [value]`", so it reads a few records using the primary index, not 3000.

With the LIMIT clause, the customer can rewrite the query and use the LIMIT clause to limit the number of rows returned by the query, as follows:

```
SELECT * FROM PRODUCT WHERE PARTNUM > partnum ORDER BY PARTNUM LIMIT 5;
```

This limits the number of rows returned by this query to 5 rows. When an ORDER BY clause is used with proper indexes created, the performance is faster than the original query.

4.3.46.10 Examples For the SELECT Command

The following examples demonstrate how you can use the select command:

- Example 1
- Example 2
- Example 3
- Example 4
- Example 5
- Example 6

Example 1

```
SELECT * FROM EMP WHERE SAL = 1300;
```

Returns the following result:

```
    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
--------- ---------- --------- --------- --------- --------- ---------- ----------
     7782 CLARK      MANAGER        7839 1981-06-0      1300                    10
     7934 MILLER     CLERK          7782 1982-01-2      1300                    10
```

Example 2

```
SELECT 'ID=',EMPNO, 'Name=',ENAME, 'Dept=',DEPTNO
FROM EMP ORDER BY DEPTNO;
```

Returns the following result:

```
'ID     EMPNO 'NAME ENAME      'DEPT    DEPTNO
--- --------- ----- ---------- ----- ---------
ID=      7839 Name= KING       Dept=        10
ID=      7934 Name= MILLER     Dept=        10
ID=      7782 Name= CLARK      Dept=        10
ID=      7566 Name= JONES      Dept=        20
ID=      7876 Name= ADAMS      Dept=        20
ID=      7788 Name= SCOTT      Dept=        20
ID=      7369 Name= SMITH      Dept=        20
ID=      7902 Name= FORD       Dept=        20
ID=      7521 Name= WARD       Dept=        30
ID=      7900 Name= JAMES      Dept=        30
ID=      7844 Name= TURNER     Dept=        30
ID=      7499 Name= ALLEN      Dept=        30
ID=      7654 Name= MARTIN     Dept=        30
ID=      7698 Name= BLAKE      Dept=        30

14 rows selected.
```

Example 3

```
SELECT 'ID=', EMPNO,
'Name=', ENAME,
'Dept=', DEPTNO
FROM EMP WHERE SAL >= 1300;
```

Returns the following result:

```
'ID     EMPNO 'NAME ENAME      'DEPT    DEPTNO
--- --------- ----- ---------- ----- ---------
ID=      7839 Name= KING       Dept=        10
ID=      7698 Name= BLAKE      Dept=        30
ID=      7782 Name= CLARK      Dept=        10
ID=      7566 Name= JONES      Dept=        20
ID=      7499 Name= ALLEN      Dept=        30
ID=      7844 Name= TURNER     Dept=        30
ID=      7902 Name= FORD       Dept=        20
ID=      7788 Name= SCOTT      Dept=        20
ID=      7934 Name= MILLER     Dept=        10

9 rows selected.
```

Example 4

```
SELECT * FROM (SELECT ENAME FROM EMP WHERE JOB = 'CLERK'
UNION
SELECT ENAME FROM EMP WHERE JOB = 'ANALYST');
```

Returns the following result:

```
ENAME
----------
ADAMS
FORD
JAMES
MILLER
SCOTT
SMITH
```

Example 5

In this example, the "ordered" hint selects the EMP table as the outermost table in the join ordering. The optimizer still attempts to pick the best possible indexes to use for execution. All other optimizations, such as view replacement and subquery unnesting are still attempted.

```
Select //ordered//  Eno, Ename, Loc from Emp, Dept
where Dept.DeptNo = Emp.DeptNo and Emp.Sal > 50000;
```

Example 6

In this example, the hint joins the tables (Product, Item, and Ord) in the given order: Product, Item, and Ord. The hint is limited only to the subquery.

```
Select CustId, Name, Phone from Customer
Where CustId In ( Select //ordered// Ord.CustId from Product, Item, Ord
Where Ord.OrdId = Item.OrdId And
Item.ProdId = Product.ProdId And
Product.Descrip like '%TENNIS%')
```

4.3.47 SET TRANSACTION

Syntax

The syntax for `SET TRANSACTION` is displayed in Figure 4–59.

Figure 4–59 The SET TRANSACTION Command

BNF Notation

```
SET TRANSACTION ISOLATION LEVEL
{ READ COMMITTED
 | REPEATABLE READ
 | SERIALIZABLE
 | SINGLE USER}
;
```

Prerequisite

If you use a SET TRANSACTION statement, it must be the first statement in your transaction. However, a transaction need not have a SET TRANSACTION statement.

> **Note:** Oracle Lite implicitly commits the current transaction before and after executing a data definition language statement.

Purpose

Establishes the isolation level of the current transaction.

The arguments for the SET TRANSACTION command are listed in Table 4–49.

Table 4–49 Arguments Used with the SET TRANSACTION Command

Argument	Description
SET TRANSACTION	Establishes the isolation level of the current transaction. 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.
ISOLATION LEVEL	Specifies how transactions containing database modifications are handled.
READ COMMITTED	An isolation level. The transaction does not take place until rows write locked by other transactions are unlocked. The transaction holds a read lock when it reads the current row and a write lock when it updates or deletes the current row. This prevents other transactions from updating or deleting it. The transaction releases read locks when it moves off the current row, and releases write locks when it is either committed or rolled back.
REPEATABLE READ	An isolation level. The transaction does not take place until rows write locked by other transactions are unlocked. The transaction maintains read locks on all rows it returns to the application, and maintains write locks on all rows it inserts, updates, or deletes. The transaction only releases its locks when it is committed or rolled back.

Table 4–49 (Cont.) Arguments Used with the SET TRANSACTION Command

Argument	Description
SERIALIZABLE	An isolation level. The transaction does not take place until rows write locked by other transactions are unlocked. The transaction holds a read lock when it reads a range of rows and a write lock when it updates or deletes a range of rows. This prevents other transactions from updating or deleting the rows.
SINGLEUSER	An isolation level. The transaction has no locks and therefore consumes less memory. This is recommended for bulk loading of the database.

Usage Notes

None.

Example

```
SET TRANSACTION ISOLATION LEVEL SINGLEUSER;
```

Related Topics

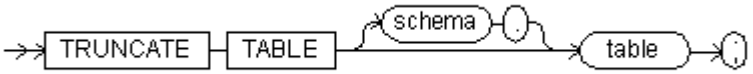
COMMIT, ROLLBACK

4.3.48 TRUNCATE TABLE

Syntax

The syntax for `TRUNCATE TABLE` is displayed in Figure 4–60.

Figure 4–60 The TRUNCATE TABLE Command



BNF Notation

```
TRUNCATE TABLE [schema .] table ;
```

Purpose

This command deletes all rows from the table. The statement is provided to be compatible with Oracle database. This statement performs the same action as the following:

```
DELETE FROM table_name ;
```

The arguments for the `TRUNCATE TABLE` command are listed in Table 4–50.

Table 4–50 Arguments Used with the TRUNCATE TABLE Command

Argument	Description
schema	The schema that contains the table.
table	The name of the table to be truncated.

Usage Notes

A table cannot be truncated if it has a primary key and there are rows in the dependent tables.

Example

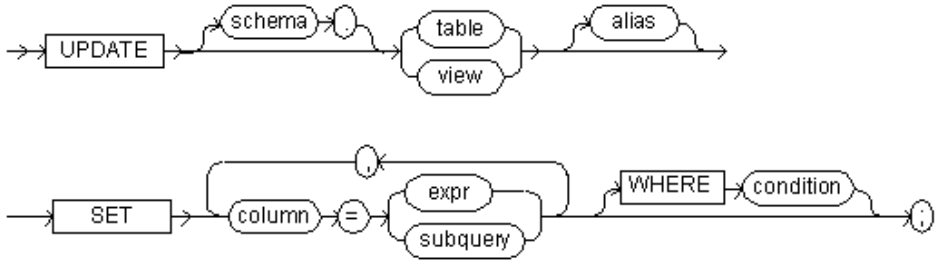
```
TRUNCATE TABLE emp;
```

4.3.49 UPDATE

Syntax

The syntax for UPDATE is displayed in Figure 4–61.

Figure 4–61 The Update Command



BNF Notation

```
UPDATE [schema .] { table | view} [ alias ]
SET column = { expr | subquery }
  [, column = { expr | subquery }]...
[WHERE condition] ;
```

Prerequisite

To update existing values in a database table or view, you must be logged into the database as SYSTEM, or the table(s) and view(s) must be part of your schema.

Purpose

Changes existing values in a table or in a view's base table.

The arguments for the UPDATE command are listed in Table 4–51.

Table 4–51 Arguments Used with the UPDATE Command

Argument	Description
schema	The schema that contains the table or view. If you omit *schema*, Oracle Lite assumes that the table or view resides in your own schema.
table	The name of the table to be updated.
view	The name of the view whose base tables you want to update.
alias	Relabels the name of the table or view in the other clauses of the UPDATE command.
SET	Indicates that the columns that follow be set to specific values.

Table 4–51 (Cont.) Arguments Used with the UPDATE Command

Argument	Description
column	The name of a column of the table or view to be updated. If you omit one of the table's columns in the SET clause, that column's value remains unchanged.
expr	The new values assigned to the corresponding column. This can contain host variables.
subquery	The subquery to be updated.
WHERE	Restricts the rows updated to those for which the specified condition is TRUE. If you omit the WHERE clause, Oracle Lite updates all rows in the table or view.
condition	A search condition. For more information about creating a valid condition, see Section 1.7, "Specifying SQL Conditions".

Usage Notes

- The same column name may not appear more than once in the SET clause.
- If no WHERE clause is specified, then all rows of the table are updated.
- A positioned UPDATE requires that the cursor be updatable.

Example

```
UPDATE EMP SET SAL = SAL * .45 WHERE JOB = 'PRESIDENT';
```

ODBC 2.0

The ODBC SQL syntax for UPDATE is the same as specified. In addition, the following syntax is supported:

```
WHERE CURRENT OF CURSOR cursor_name
```

Related Topics

DELETE, INSERT

A

Oracle Database Lite Keywords and Reserved Words

This appendix lists Oracle Database Lite keywords and reserved words.

A.1 Oracle Database Lite Keywords

Keywords are not reserved words, but have special meanings in certain contexts. They can be used to define table and column names.

> **Note:** Exercise caution when defining a new method for a Java class as aggregate names (AVG, MAX, MIN, COUNT, SUM) and function names (UPPER, LOWER, and so on) take precedence over user defined method names.

Oracle Database Lite keywords are listed in Table A–1:

Table A–1 Oracle Database Lite Keywords

Letter	Keywords	
A	ADD_MONTHS	ASCII
	AFTER	AUTOCOMMIT
	ARGS	AVG
B	BEFORE	BIT
	BIGINT	BIT_LENGTH
	BINARY	
C	CASCADE	COMPILE
	CAST	CONCAT
	CATALOG	CONSTRUCTOR
	CEIL	CONVERT
	CHAR	COUNT
	CHAR_LENGTH	CURDATE
	CHARACTER	CURTIME
	CHR	CURTIMESTAMP
	COMMIT	CURVAL
	COMMITTED	

Table A–1 (Cont.) Oracle Database Lite Keywords

Letter	Keywords	
D	DATABASE_ID	DEC
	DATABASE_SIZE	DECIMAL
	DATE	DOUBLE
	DAY	DUAL (Do not use as a table name as this is already the name of a dummy table.)
	DAYOFMONTH	
	DAYOFWEEK	DUMP$
	DAYOFYEAR	
	DEBUG_LITE	
E	EXTENT_SIZE	
	EXTRACT	
F	FLOOR	
G	GREATEST	
H	HIDE	
	HOUR	
I	IFNULL	INT
	INITCAP	INTERVAL
	INSTR	ISOLATION
	INSTRB	
K	KEY	
L	LAST_DAY	LEVEL
	LCASE	LOCATE
	LEAST	LOWER
	LENGTH	LPAD
	LENGTHB	LTRIM
M	MAX	MINVALUE
	MAXVALUE	MOD
	MIN	MONTH
	MINUTE	MONTHS_BETWEEN
N	NEXT_DAY	NUMBER
	NEXTVAL	NUMERIC
	NOMAXVALUE	NVL
	NOMINVALUE	
	NOW	
O	OCTET_LENGTH	
	OJ	
P	POSITION	
	PRECISION	
Q	QUARTER	

Table A–1 (Cont.) Oracle Database Lite Keywords

Letter	Keywords	
R	READ	ROUND
	REAL	ROWID
	REPEATABLE	RPAD
	REPLACE	RTRIM
	RESTRICT	
S	SAVEPOINT	SQL_TSI_MONTH
	SCHEMA	SQL_TSI_QUARTER
	SECOND	SQL_TSI_SECOND
	SEQUENCE	SQL_TSI_WEEK
	SERIALIZABLE	SQL_TSI_YEAR
	SMALLINT	START
	SOURCE	STDDEV
	SQL_TSI_DAY	SUBSTR
	SQL_TSI_FRAC_SECOND	SUBSTRB
	SQL_TSI_HOUR	SUBSTRING
	SQL_TSI_MINUTE	SUM
T	TIME	TO_DATE
	TIMESTAMP	TO_NUMBER
	TIMESTAMPADD	TRANSACTION
	TIMESTAMPDIFF	TRANSLATE
	TINYINT	TRIMBOTH
	TO_CHAR	TRUNC
U	UCASE	UPPER
	UNCOMMITTED	
V	VARBINARY	VARYING
	VARIANCE	
W	WEEK	
	WORK	
Y	YEAR	
Z	ZONE	

Note: You can use the keywords NEXTVAL and CURVAL as column names of a table. However, Oracle Database Lite treats *tablename*.NEXTVAL and *tablename*.CURVAL as referring to a sequence.

A.2 Oracle Database Lite Reserved Words

Reserved words cannot be used as the name of any database object or part. The Oracle Database Lite reserved words are listed in Table A–2. Some Oracle Database Lite reserved words are also Oracle reserved words. Any words followed by an asterisk (*) are only Oracle Database Lite reserved words (not Oracle):

Table A–2 Oracle Database Lite Reserved Words

Letter	Reserved Words	
A	ADD	ANY
	ALL	AS
	ALTER	ASC
	AND	ATTACH*
B	BETWEEN	BY
	BOTH*	
C	CALL*	CONNECT
	CASE*	CONSTRAINT*
	CAST*	CONSTRAINTS*
	CHECK	CREATE
	CLASS*	CURRENT
	COLUMN	CURRENT_DATE*
	COMMENT	CURRENT_TIME*
		CURRENT_TIMESTAMP*
D	DATABASE	DESC
	DECODE*	DETACH*
	DEFAULT	DISTINCT
	DELETE	DROP
E	EACH*	ESCAPE*
	ELSE	EXISTS
	END*	
F	FLOAT	FROM
	FOR	FULL*
	FOREIGN*	
G	GRANT	
	GROUP	
H	HAVING	
I	IN	INTERSECT
	INCREMENT	INTERVAL*
	INDEX	INTO
	INSERT	IS
	INTEGER	
J	JOIN*	
	JAVA	
L	LEADING*	LOCK
	LEFT*	LONG
	LIKE	
M	MINUS	
N	NOT	NULL
	NOWAIT	

Table A–2 (Cont.) Oracle Database Lite Reserved Words

Letter	Reserved Words	
O	OF	ORDER,
	OFF*	OUTER*
	ON	OPTION
	OR	
P	PRIMARY*	PUBLIC
	PRIOR	
R	RAW	RIGHT*
	REFERENCES*	ROLLBACK*
	REVOKE	ROW
S	SELECT	SQL_INTEGER*
	SESSION	SQL_LONGVARCHAR*
	SET	SQL_REAL*
	SOME*	SQL_SMALLINT*
	SQL_BIGINT*	SQL_TIME*
	SQL_CHAR*	SQL_TIMESTAMP*
	SQL_DATE*	SQL_VARCHAR*
	SQL_DECIMAL*	START
	SQL_DOUBLE*	SYNONYM
	SQL_FLOAT*	SYSDATE
T	TABLE	TIMEZONE_MINUTE*
	THEN	TO
	TIMESTAMPADD*	TRAILING*
	TIMESTAMPDIFF*	TRIGGER
	TIMEZONE_HOUR*	TRIM*
U	UNION	UPDATE
	UNIQUE	USER
V	VALUES	VARCHAR2
	VARCHAR	VIEW
W	WHEN*	WITH
	WHERE	

B

SQL Limitations For Oracle Database Lite

There are limitations to SQL that is different than the Oracle database, as follows:

Table B–1 Datatype Limits

Datatypes	Limit	Comments
BFILE	Maximum size: 2 GB Maximum size of the directory or file names: no database imposed limit	All BFILE objects are stored as LOB
BLOB	Maximum size: 2 GB	
CHAR	Maximum size: 4096 bytes	
CHAR VARYING	Maximum size: 4096 bytes	
CLOB	Maximum size: 2 GB	
Literals	No limit	
LONG	Maximum size: 2 GB	A table can have any number of long columns
NUMBER	Operating system limit	NUMBER is converted to a double precision number on the native platform
NUMBER (p, s)	999 ... (38 9's) x 10 ^ 125 maximum -999... (38 9's) x 10 ^125 minimum	Maximum precision of 38 decimal digits
VARCHAR	Maximum size: 4096 bytes	
VARCHAR2	Maximum size: 4096 bytes	

Table B–2 Physical Database Limits

Item	Limit	Comments
Database Block Size	4096 bytes	Fixed size
Database File	1 database file for each catalog	An application can open any number of catalogs.
Database File Size	4 GB	Affected by the operating system. Maximum file size allowed by the operating system.
Max Object or Row Length	4040	When an object (row) exceeds this length, it is converted into a binary long object. So, UNION will not work on this table.
DSN Name	31 bytes (31 US chars)	Limit is 31 bytes
Database Path Name	129 bytes	_MAX_PATH -5, which is 255 on Win32
Database filename	129 bytes	_MAX_PATH -8, which is 252 on Win32.

Table B–3 Logical Database Limits

Item	Limit	Comments
Indexes	Maximum for each table	unlimited
Columns	table	1000
	index	32 columns maximum
Constraints	Maximum for each column	unlimited
Nested queries	Maximum number	unlimited
Rows	Maximum number for the table	no limit
SQL statement length	Maximum length of statements	unlimited, particular tools may impose lower limits

Table B–4 Process/Runtime Limits

Item	Limit	Comments
Shared Memory	128 MB Maximum	
Cache	64 4K blocks by default	Used for caching database pages

C

Oracle Database Lite Datatypes

Oracle Lite supports the datatypes listed in Table C–1:

Table C–1 Datatypes Supported by Oracle Database Lite

Datatype	Description
BIGINT	An integer datatype with a precision of 19 decimal digits.
BINARY	Enables storage of binary data up to 4,096 bytes.
BIT	Enables your application to store a bit unconstrained by character semantics.
BLOB	A binary large object. Maximum size is 2 gigabytes.
CHAR	Fixed length character data of length size bytes. Maximum size is 4,096 bytes. Default and minimum size is 1 byte.
CLOB	A character large object containing single-byte characters. Both fixed-width and variable-width character sets are supported, both using the CHAR database character set. Maximum size is 2 gigabytes.
DATE	Valid date range from January 1, 4712 BC to December 31, 4712 AD.
DECIMAL	A number that can be measured in terms of precision (decimal value) or scale (fractional value). You can measure precision by using DECIMAL (*p*). You can measure scale by using NUMERIC (*p*, *s*). Precisions larger than the one you specify are acceptable, but smaller ones are not.
DOUBLE PRECISION	Contains a precision defined during implementation which must be greater than the precision of REAL.
FLOAT	Enables you to specify the precision. The resulting precision must be at least as large as the precision you request. You can specify a precision of some value by typing FLOAT (*p*). For example, a portable application, may use a single precision on one platform and double precision on another.
INTEGER	An integer value whose precision (the number of decimal values or bits that can be stored) is defined upon implementation.
LONG	Character data of variable length up to 2 gigabytes, or 231 -1 bytes.
LONG RAW	Raw binary data of variable length up to 2 gigabytes.
LONG VARBINARY	Stores but does not interpret up to 2 gigabytes of variable binary data.

Table C–1 (Cont.) Datatypes Supported by Oracle Database Lite

Datatype	Description
LONG VARCHAR	Variable-length character string having maximum length size bytes. Maximum size is 2 gigabytes, and minimum is 1. You must specify size for a VARCHAR2.
NUMBER	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. A number with no precision now maps to DOUBLE PRECISION in Oracle compatibility mode.
NUMERIC	A number that can be measured in terms of precision (decimal value) or scale (fractional value). You can measure precision by using DECIMIAL (*p*). You can measure scale by using NUMERIC (*p*, *s*). The scale cannot be negative and cannot be larger than the number itself.
RAW	Raw binary data of length size bytes. Maximum size is 4,096 bytes. You must specify size for a RAW value.
REAL	Enables you to request a single-precision floating point with no options. The precision is chosen by the implementation and is normally the default single-precision datatype on the hardware platform.
ROWID	A 16-byte hexadecimal string representing the unique address of a row in its table. ROWID is primarily for values returned by the ROWID pseudocolumn.
SMALLINT	An integer value whose precision is defined upon implementation but whose value is no greater than the implementation of INTEGER.
TIME	Stores a time value in terms of hours minutes and seconds. Hours are represented by two digits ranging from 00 through 23. Minutes are also represented by two digits ranging from 00 through 59. The seconds value ranges from 00 through 60.
TIMESTAMP	Stores the year, month, and day values of a date and the hour, minute, second value of time. TIMESTAMP length and restrictions correspond to DATE and TIME values, except that in TIME the default is 0 and in TIMESTAMP it is 6.
TINYINT	An integer with a precision of 1 byte (-128 to +127).
VARBINARY	Stores but does not interpret variable binary data.
VARCHAR	See VARCHAR2
VARCHAR2	Variable-length character string with a maximum length size of 4,096 bytes (minimum is 1). You must specify size for a VARCHAR2.

C.1 BIGINT

[ODBC]

Big integer type. Binds with SQL_C_CHAR or SQL_C_BINARY variables.

Syntax

```
BIGINT
```

Usage Notes

A BIGINT is an exact numeric value with precision 19 and scale 0, typically 8 bytes. `-10^19 < n < 10^19`, where *n* is the value of a BIGINT.

Example

```
BIGINT
```

C.2 BINARY

[ODBC]

Variable length binary datatype. Binds with a SQL_C_CHAR or SQL_C_BINARY array.

Syntax

```
BINARY [( <precision> )]
```

Keywords and Parameters

<*precision*> is the maximum number of bytes.

Usage Notes

BINARY is synonymous with VARBINARY and RAW.

Example

```
BINARY(1024)
```

C.3 BIT

Bit datatype.

Syntax

```
BIT
```

Usage Notes

Precision is 1.

Example

```
BIT
```

C.4 BLOB

The BLOB datatype can store large and unstructured data such as text, image, video, and spatial data up to 2 gigabytes in size.

Syntax

```
BLOB
```

Usage Notes

When creating a table, you can optionally specify different tablespace and storage characteristics for BLOB columns.

You can initialize a column with the BLOB datatype by inserting an EMPTY_BLOB. See Example 2.

BLOB columns contain LOB locators that can refer to out-of-line or in-line LOB values. Selecting a LOB from a table actually returns the LOB's locator and not the entire LOB value.

BLOB is similar to LONG and LONG RAW types, but differs in the following ways:

- BLOBs can be attributes of a user-defined datatype (object).
- The BLOB locator is stored in the table column, either with or without the actual BLOB value. BLOB values can be stored in separate tablespaces.
- When you access a BLOB column, the locator is returned.
- A BLOB can be up to 2 gigabytes in size.
- BLOBs permit efficient, random, piece-wise access to and manipulation of data.
- You can define more than one BLOB column in a table.
- You can define one or more BLOB attributes in an object.
- You can declare BLOB bind variables.
- You can select BLOB columns and BLOB attributes.
- You can insert a new row or update an existing row that contains one or more BLOB columns and/or an object with one or more BLOB attributes. (You can set the internal BLOB value to NULL, empty, or replace the entire BLOB with data.
- You can update a BLOB row/column intersection or a BLOB attribute with another BLOB row/column intersection or BLOB attribute.
- You can delete a row containing a BLOB column or BLOB attribute. This also deletes the BLOB value.

To access and populate rows of an internal BLOB column (a BLOB column stored in the database), use the INSERT statement first to initialize the internal BLOB value to empty.

Example 1

The following example creates a table with a BLOB column:

```
CREATE TABLE  PERSON_TABLE (NAME CHAR(40),
                            PICTURE BLOB);
```

Example 2

The following example initializes a column with the BLOB datatype by inserting an EMPTY_BLOB:

```
INSERT INTO PERSON_TABLE (NAME, PICTURE) VALUES ('Steve', EMPTY_BLOB());
```

C.5 CHAR

[ODBC] [SQL-92] [Oracle]

Fixed length character string type. CHAR columns allocate a fixed space in a database row, allowing for the maximum length. Strings shorter than the maximum are padded with trailing blanks.

Syntax

```
CHAR
CHARACTER
CHAR ( <length> )
CHARACTER ( <length> )
```

Keywords and Parameters

<*length*> is the number of characters in a string. The limit is 4,096 bytes.

Usage Notes

If <*length*> is omitted, 1 is assumed.

Examples

```
CHAR
CHAR(20)
```

C.6 CLOB

The CLOB datatype can store large and unstructured data, such as text and spatial data up to 2 gigabytes in size.

Syntax

```
CLOB
```

Usage Notes

When creating a table, you can optionally specify different tablespace and storage characteristics for CLOB columns.

You can initialize a column with the CLOB datatype by inserting an EMPTY_CLOB. See Example 2.

CLOB columns contain LOB locators that can refer to out-of-line or in-line LOB values. Selecting a LOB from a table actually returns the LOB's locator and not the entire LOB value.

CLOB is similar to LONG and LONG RAW types, but differs in the following ways:

- CLOBs can be attributes of a user-defined datatype (object).
- The CLOB locator is stored in the table column, either with or without the actual CLOB value. CLOB values can be stored in separate tablespaces.
- When you access a CLOB column, the locator is returned.
- A CLOB can be up to 2 gigabytes in size.
- CLOBs permit efficient, random, piece-wise access to and manipulation of data.
- You can define more than one CLOB column in a table.
- You can define one or more CLOB attributes in an object.
- You can declare CLOB bind variables.
- You can select CLOB columns and CLOB attributes.
- You can insert a new row or update an existing row that contains one or more CLOB columns and/or an object with one or more CLOB attributes. (You can set the internal CLOB value to NULL, empty, or replace the entire CLOB with data.
- You can update a CLOB row/column intersection or a CLOB attribute with another CLOB row/column intersection or CLOB attribute.
- You can delete a row containing a CLOB column or CLOB attribute and thereby also delete the BLOB value.

To access and populate rows of an internal CLOB column (a CLOB column stored in the database), use the INSERT statement first to initialize the internal CLOB value to empty.

Example 1

The following example creates a table with a CLOB column:

```
CREATE TABLE WORK_HISTORY (NAME CHAR (40),
                           RESUME CLOB);
```

Example 2

The following example initializes a column with the CLOB datatype by inserting EMPTY_CLOB:

```
INSERT INTO WORK_HISTORY (NAME, RESUME) VALUES ('Steve', EMPTY_CLOB());
```

C.7 DATE

[ODBC] [SQL-92]

Stores day, month, and year in SQL-92 and ODBC. In Oracle, it also stores the time.

Syntax

```
DATE
```

Example

```
DATE
```

C.8 DECIMAL

[ODBC] [SQL-92]

Decimal number type.

Syntax

```
DECIMAL [ ( <precision>[, <scale> ] ) ] | DEC [ ( <precision>[, <scale> ] ) ]
```

Keywords and Parameters

<*precision*> is the precision of a decimal number.

<*scale*> is the scale of a decimal number (the number of digits to the right of the decimal point).

Usage Notes

A DECIMAL is an exact numeric value. By default, DECIMAL data is returned as a character string or SQL_C_CHAR, but conversion into SQL_C_LONG or SQL_C_FLOAT or other datatypes is supported. If <*precision*> is not specified, 38 is assumed. If <*scale*> is not specified, 0 is assumed. 0 <= <*scale*> <= <*precision*> <= 38.

DECIMAL is synonymous with NUMERIC and NUMBER.

Examples

```
DECIMAL
DEC (5)
DECIMAL (10, 5)
```

C.9 DOUBLE PRECISION

[ODBC]

Double precision floating point number type. Binds with a SQL_C_DOUBLE variable.

Syntax

```
DOUBLE PRECISION
```

Usage Notes

A DOUBLE PRECISION is a signed, approximate, numeric value with a mantissa decimal precision 15. Its absolute value is either zero or between 10^-308 and 10^308.

Example

```
DOUBLE PRECISION
```

C.10 FLOAT

[ODBC]

Floating point number type. Binds with a SQL_C_DOUBLE variable.

Syntax

```
FLOAT [ ( <precision> ) ]
```

Keywords and Parameters

<*precision*> is the precision of a floating point number.

Usage Notes

A FLOAT is a signed approximate numeric value with a mantissa decimal precision 15. Its absolute value is either zero or between 10^-308 and 10^308. In the current implementation, the precision of a FLOAT is always set to 15.

Examples

```
FLOAT
FLOAT (10)
```

C.11 INTEGER

[ODBC] [SQL-92]

Integer type.

Syntax

```
INTEGER
INT
```

Usage Notes

An INTEGER is an exact numeric value with precision 10 and scale 0, typically 4 bytes. Binds with SQL_C_LONG or SQL_C_ULONG and SQL_C_SLONG. -2^31 < n < 2^31, where *n* is the value of an INTEGER.

Examples

```
INTEGER
```

```
INT
```

C.12 LONG

[Oracle]

Variable-length character string type. Used when the length of the string exceeds 4,096 bytes.

Syntax

```
LONG
```

Keywords and Parameters

<length> is the maximum number of characters in a string.

Usage Notes

The maximum length of a LONG is 2 billion bytes. If *<length>* is omitted, 2 megabytes is assumed. You can create an index on a LONG column, but only the first 2,000 bytes are used in the index.

Example

```
LONG
```

C.13 LONG RAW

[Oracle]

Variable length binary datatype. Similar to LONG VARBINARY. Use this type when a VARBINARY column exceeds 4,096 bytes.

Syntax

```
LONG RAW [( <precision> )]
```

Keywords and Parameters

<precision> is the maximum number of bytes. If not specified, the default is 2 megabytes.

Usage Notes

The maximum length of a LONG RAW is 2 billion bytes.

Examples

```
LONG RAW(1048576)
```

C.14 LONG VARBINARY

[ODBC]

Variable length binary datatype.

Syntax

```
LONG BINARY [( <precision> )]
```

Keywords and Parameters

<*precision*> is the maximum number of bytes. If not specified, the default is 2 megabytes.

Usage Notes

1 <= <*precision*> <= 2G.

Examples

```
LONG VARBINARY(1048576)
```

C.15 LONG VARCHAR

[ODBC]

Variable-length character string type. Used when the length of the string exceeds 4,096 bytes.

Syntax

```
LONG VARCHAR
LONG VARCHAR ( <length> )
```

Keywords and Parameters

<*length*> is the maximum number of characters in a string.

Usage Notes

The maximum length of a LONG VARCHAR is 2 billion bytes. If <*length*> is omitted, 2 megabytes is assumed. You can create an index on a LONG VARCHAR column, but only the first 2,000 bytes are used in the index.

Example

```
LONG VARCHAR
```

C.16 NUMBER

[Oracle]

DECIMAL number type.

Syntax

```
NUMBER [ ( <precision>[, <scale> ] ) ]
```

Keywords and Parameters

<*precision*> is the precision of a decimal number.

<*scale*> is the scale of a decimal number (the number of digits to the right of the decimal point).

Usage Notes

A NUMBER is an exact numeric value. By default, NUMBER data is returned as a character string or SQL_C_CHAR, but conversion into SQL_C_LONG or SQL_C_FLOAT or other datatypes is supported. If <*precision*> is not specified, 38 is assumed. If <*scale*> is not specified, 0 is assumed. 0 <= <*scale*> <= <*precision*> <= 38.

NUMBER is synonymous with DECIMAL and NUMERIC.

Examples

```
NUMBER
NUMBER (10, 5)
```

C.17 NUMERIC

[ODBC] [SQL-92]

DECIMAL number type.

Syntax

```
NUMERIC [ ( <precision>[, <scale> ] ) ]
```

Keywords and Parameters

<*precision*> is the precision of a decimal number.

<*scale*> is the scale of a decimal number (the number of digits to the right of the decimal point).

Usage Notes

A NUMERIC is an exact numeric value. By default, NUMERIC data is returned as a character string or SQL_C_CHAR, but conversion into SQL_C_LONG or SQL_C_FLOAT or other datatypes is supported. If <*precision*> is not specified, 38 is assumed. If <*scale*> is not specified, 0 is assumed. 0 <= <*scale*> <= <*precision*> <= 38.

NUMERIC is synonymous with DECIMAL and NUMBER.

Examples

```
NUMERIC
NUMERIC (10, 5)
```

C.18 RAW

[Oracle]

Variable length binary datatype. Binds with a SQL_C_CHAR or SQL_C_BINARY array.

Syntax

```
RAW [( <precision> )]
```

Keywords and Parameters

<*precision*> is the maximum number of bytes.

Usage Notes

RAW is synonymous with BINARY and VARBINARY, but has a limit of 4,096 bytes.

Examples

```
RAW(1024)
```

C.19 REAL

[ODBC]

Floating point number type. Binds with SQL_C_REAL variables.

Syntax

```
REAL
```

Usage Notes

A REAL is a signed approximate numeric value with a mantissa decimal precision 7. Its absolute value is either zero or between 10^-38 and 10^38.

Example

```
5600E+12
```

C.20 ROWID

A 16-byte hexadecimal string representing the unique address of a row in its table. ROWID is primarily for values returned by the ROWID pseudocolumn.

Usage Notes

In Oracle Lite, the ROWID is the hexadecimal string representing the unique object identifier. It is not compatible with the Oracle ROWID, but it may be used to uniquely identify a row for updating. ROWID literals should be enclosed in single quotes.

Example

```
A80000.00.03000000
```

C.21 SMALLINT

[ODBC] [SQL-92]

Small integer type.

Syntax

```
SMALLINT
```

Usage Notes

A SMALLINT is an exact numeric value with precision 5 and scale 0, typically 2 bytes or 16 bits. If signed, the range can be -32,768 to +32,767 (SQL_C_SSHORT or SQL_C_SHORT) or, if unsigned, 0 to 65,535 (SQL_C_USHORT). -32,768 <= *n* <= 32,767, where *n* is the value of a SMALLINT.

Example

```
SMALLINT
```

C.22 TIME

[ODBC] [SQL-92]

Stores hour, minutes, seconds, and possibly, fractional seconds.

Syntax

```
TIME
TIME ( <precision> ) [SQL-92]
```

Keywords and Parameters

<*precision*> is the number of fractional digits in seconds.

Examples

```
TIME
TIME (3)
```

C.23 TIMESTAMP

[ODBC] [SQL-92]

Stores both date and time in SQL-92 and is comparable to the Oracle DATE datatype.

Syntax

```
TIMESTAMP [ ( <precision> ) ]
```

Keywords and Parameters

<*precision*> is the number of fractional digits in seconds. 0 <= <*precision*> <= 6

Usage Notes

During replication of an Oracle table, DATE columns in Oracle are stored as TIMESTAMP columns in Oracle Lite.

Examples

```
TIMESTAMP
TIMESTAMP (3)
```

C.24 TINYINT

[ODBC]

A one byte integer type.

Syntax

```
TINYINT
```

Usage Notes

A one byte integer with range 0 to 127. If unsigned (SQL_C_UTINYINT) or - 128 to + 127, and if signed (SQL_C_STINYINT).

Example

```
TINYINT
```

C.25 VARBINARY

[ODBC]

Variable length binary datatype. Binds with a SQL_C_CHAR or SQL_C_BINARY array.

Syntax

```
VARBINARY [( <precision> )]
```

Keywords and Parameters

<*precision*> is the maximum number of bytes.

Usage Notes

VARBINARY is synonymous with BINARY and RAW.

Example

```
VARBINARY(1024)
```

C.26 VARCHAR

[ODBC] [SQL-92] [Oracle]

Variable-length character string type.

Syntax

```
VARCHAR ( <length> )
CHAR VARYING ( <length> )
CHARACTER VARYING ( <length> )
```

Keywords and Parameters

<*length*> is the maximum number of characters in a string, between 1 and 4,096.

Usage Notes

If <*length*> is omitted, 1 is assumed.

Examples

```
VARCHAR(20)
CHAR VARYING(20)
CHARACTER VARYING(20)
```

C.27 VARCHAR2

[Oracle]

Variable-length character string type. VARCHAR and VARCHAR2 are stored exactly as passed, provided the length does not exceed the maximum. No blank padding is added. VARCHAR and VARCHAR2 are equivalent.

Syntax

```
VARCHAR2 ( <length> )
CHAR VARYING ( <length> )
CHARACTER VARYING ( <length> )
```

Keywords and Parameters

<*length*> is the maximum number of characters in a string, between 1 and 4,096.

Usage Notes

If <*length*> is omitted, 1 is assumed.

Examples

```
VARCHAR2(20)
CHAR VARYING(20)
```

```
CHARACTER VARYING(20)
```

D

Oracle Database Lite Literals

Oracle Lite supports the following literals:

D.1 CHAR, VARCHAR

Character string literal value.

Syntax

```
'<letters>'
```

Keywords and Parameters

<letters> a sequence of zero or more printable characters excluding new-line.

Usage Notes

If a single quote is part of a literal, it must be preceded by another single quote (used as an escape character). The maximum length of a character literal is 1024.

Examples

```
'a string'
'a string containing a quote '''
```

D.2 DATE

Date literal value.

Syntax

```
[DATE] ' <year1 ><month1 ><day >' [SQL-92]
{ d ' <year1 ><month1 ><day >' [ODBC]
--(* d ' <year1 ><month1 ><day >' *)-- [ODBC]
' <day ><month2 ><year2 >' [Oracle]
```

```
' <day ><month2 ><year1 >' [Oracle]
' <month1 ><day ><year2 >' [Oracle]
' <month1 ><day ><year1 >' [Oracle]
```

Keywords and Parameters

<*year1*> a four-digit number representing a year, for example, 1994.

<*year2*> a two-digit number representing the last two digits of a year.

<*month1*> a two-digit number between 01 and 12.

<*month2*> a three-letter initial of a month (this is not case-sensitive).

<*day*> a two-digit number between 01 and 31 (depending on the month).

Examples

```
'1994-11-07' [SQL-92]
{ d '1994-11-07' }
--(* d '1994-11-07' *)--
DATE '10-23-94'
'23-Nov-1994' [Oracle]
'23-Nov-94'
```

D.3 DECIMAL, NUMERIC, NUMBER

Decimal number literal value.

Syntax

```
[+|- ]<digits>
[+|- ]<digits>.[<digits>]
[+|- ].<digits>
```

Keywords and Parameters

<*digits*> a sequence of one or more digits.

Examples

```
54321
-123.
+456
64591.645
+.12345
0.12345
```

D.4 REAL, FLOAT, DOUBLE PRECISION

Floating point number literal value.

Syntax

```
[+|- ]<digits ><exp >[+|- ]<digits >
[+|- ]<digits >. [<digits >]<exp >[+|- ]<digits >
[+|- ].<digits ><exp >[+|- ]<digits >
```

Keywords and Parameters

<*digits*> a sequence of one or more digits.

<*exp*>'E' or 'e'.

Examples

```
+1.5e-7
12E-5
-.12345e+6789
```

D.5 SMALLINT, INTEGER, BIGINT, TINYINT

[ODBC]

Integer literal value.

Syntax

```
[+|- ]<digits>
```

Keywords and Parameters

<*digits*> a sequence of one or more digits.

Usage Notes

Let *n* be the number the literal represents.

For TINYINT, -128 <= *n* <= 127

For a SMALLINT, -32768 <= *n* <= 32767

For an INTEGER, -2^31 < *n* < 2^31

For a BIGINT, -10^19 < *n* < 10^19

Example

```
12345
```

D.6 TIME

Time literal value.

Syntax

[TIME]' <*hour*>:<*minute*>:<*second*>[.[<*fractional_second*>]]'

Keywords and Parameters

<*hour*> a two-digit number between 00 and 23.

<*minute*> a two-digit number between 00 and 59.

<*second*> a two-digit number between 00 and 59.

<*fractional_second*> a number containing up to 6 digits.

Examples

```
'23:00:00'
TIME '23:00:00.'
TIME '23:01:59.134343'
```

D.7 TIMESTAMP

Timestamp literal value.

Syntax

```
TIMESTAMP ' <DATE_literal_value > <TIME_literal_value >'
```

Keywords and Parameters

<*DATE_literal_value*> a Date literal.

<*TIME_literal_value*> a Time literal.

Usage Notes

In a timestamp literal, there is exactly one space character between the Date literal and the Time literal.

Examples

```
TIMESTAMP '1994-11-07 23:00:00'
'94-06-01 12:02:00'
Examples: CHAR (10)
```

E

Index Creation Options

In prior releases, Oracle Lite enforced uniqueness on a set of columns by creating a unique index on all table columns. This method required a large volume of disk space for long keys or for keys containing many columns.

E.1 Uniqueness Constraint in Oracle Lite

With Oracle Lite, applications can now enforce a uniqueness constraint on a large number of columns without using a large volume of disk space. This benefits applications that require a uniqueness constraint on a large number of columns, but have table rows with the same values in a smaller subset of these columns.

E.1.1 The Address Table Example

The explanations and examples in this section all refer to the following table:

```
ADDRESS (STREET VARCHAR(40), CITY VARCHAR(40), STATE VARCHAR(20), ZIP
VARCHAR(12));
```

E.1.2 Using Uniqueness Constraints

If you want to enforce a uniqueness constraint to prevent any two rows in the ADDRESS from containing identical values for all columns, you can create a unique index on all the table's columns. However, this method requires a large volume of disk space.

If you know that very few rows have the same values in the STREET and CITY columns, you can create a unique index on STREET and CITY only. If two rows have the same values for the STREET and CITY columns, then Oracle Lite locks them and tests the rows' remaining column values for uniqueness.

Although this method requires less disk space, it also has some disadvantages. Since the database must search all records to ensure that no unique or primary key columns are identical, the following actions decrease database performance:

- inserting and updating rows
- querying rows based on primary keys

When Oracle Lite locks indexed columns that have the same values, those columns cannot be accessed by concurrent database users.

E.1.3 Specifying the Number of Columns in an Index

You can specify the number of columns in an index in the **POLITE.INI** file, or in one of the following SQL statements:

- CREATE INDEX
- CREATE TABLE
- ALTER TABLE

E.1.3.1 The POLITE.INI File

The `MAXINDEXCOLUMNS` value in the **POLITE.INI** file specifies the maximum number of columns in an index. When a user creates a new index, the index only contains the number of columns specified in the `MAXINDEXCOLUMNS` variable. For example, the following line in the **POLITE.INI** file specifies that any newly created index must contain the first two columns of the table it refers to:

```
MAXINDEXCOLUMNS=2
```

When you apply the preceding example to the ADDRESS table, the following statement creates an index that contains the columns STREET and CITY.

```
CREATE INDEX IDX1 ON ADDRESS(STREET, CITY, STATE, ZIP);
```

The following statement also creates a unique index that contains the columns, STREET and CITY:

```
CREATE UNIQUE INDEX IDX1 ON ADDRESS(STREET, CITY, STATE, ZIP);
```

Since the statement contains the UNIQUE clause, Oracle Lite designates all of the specified columns as a unique key.

E.1.3.2 The CREATE UNIQUE INDEX Statement

In Oracle Lite, the CREATE UNIQUE INDEX statement contains the following optional clause for specifying the number of indexed columns:

KEY COLUMNS = <*number_of_columns*>

Oracle Lite creates an index that contains the number of columns you specify in the KEY COLUMNS clause. For example, the following statement creates an index that contains two columns, STREET and CITY:

```
CREATE UNIQUE INDEX IDX1 ON ADDRESS (STREET, CITY, STATE, ZIP) KEY COLUMNS = 2;
```

Since the statement contains the UNIQUE clause, Oracle Lite designates all of the specified columns as a unique key

E.1.3.3 The CREATE TABLE and ALTER TABLE Statements

The PRIMARY KEY clause in the statements, CREATE TABLE and ALTER TABLE supports the following clause for specifying the number of indexed columns:

KEY COLUMNS = <*number_of_columns*>

The following example creates a table and designates four of its columns as primary keys. However the index that enforces the primary key only contains the first two columns:

```
CREATE TABLE ADDRESS (STREET VARCHAR(40), CITY VARCHAR(40), STATE VARCHAR(20), ZIP
VARCHAR(12), PRIMARY KEY(STREET, CITY, STATE, ZIP)  KEY COLUMNS = 2);
```

E.1.3.4 Usage Notes

If the **POLITE.INI** file does not include a value for the `MAXINDEXCOLUMNS` variable and the SQL statements do not use the KEY COLUMNS option, then Oracle Lite uses all of the specified columns to create an index.

If the **POLITE.INI** file specifies a `MAXINDEXCOLUMNS` value, then Oracle Lite uses this value to create all indexes and primary keys unless the KEY COLUMNS clause in a SQL statement overrides it.

F

Syntax Diagram Conventions

This document discusses the syntax diagrams used in the Oracle Database Lite SQL Reference. Topics include:

F.1 Introduction

Syntax diagrams are drawings that illustrate valid SQL syntax. To read a diagram, trace it from left to right, in the direction shown by the arrows.

Commands and other keywords appear in UPPERCASE inside rectangles. Type them exactly as shown in the rectangles. Parameters appear in lowercase inside ovals. Variables are used for the parameters. Punctuation, operators, delimiters, and terminators appear inside circles.

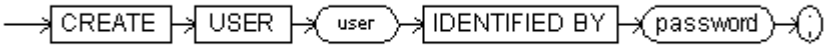
If the syntax diagram has more than one path, you can choose any path to travel.

If you have the choice of more than one keyword, operator, or parameter, your options appear in a vertical list.

F.2 Required Keywords and Parameters

Required keywords and parameters can appear singly or in a vertical list of alternatives. Single required keywords and parameters appear on the main path, that is, on the horizontal line you are currently traveling. In Figure F–1, *user* and *password* are required parameters:

Figure F–1 Syntax for Required Keywords and Parameters



BNF Notation

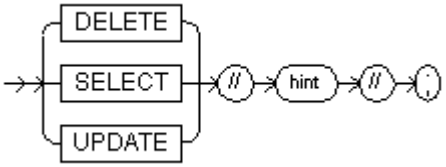
```
CREATE USER user IDENTIFIED  BY password;
```

According to the diagram, the following syntax is valid:

```
CREATE USER hannibal IDENTIFIED BY hanna;
```

In Figure F–2, either DELETE, SELECT, or UPDATE is a required parameter:

Figure F–2 Syntax for Required Parameters



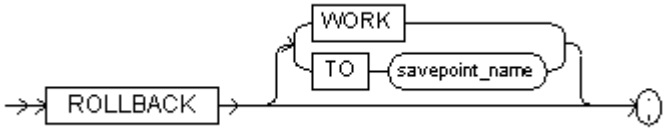
BNF Notation

```
{ DELETE | SELECT | UPDATE } //hint// ;
```

F.3 Optional Keywords and Parameters

If keywords and parameters appear in a vertical list above the main path, they are optional. In Figure F–3, you can choose from the vertical list of options or you can continue along the main path:

Figure F–3 Syntax for Required Optional Keywords and Parameters



BNF Notation

```
ROLLBACK [{ WORK | TO savepoint_name }] ;
```

According to the diagram, all of the following statements are valid:

```
ROLLBACK WORK;
ROLLBACK TO savepoint_1;
ROLLBACK;
```

F.4 Syntax Loops

Loops enable you to repeat the syntax within them as many times as you like. In Figure F–4, after choosing one expression, you can go back repeatedly to choose another, separated by commas.

Figure F–4 A Syntax Loop

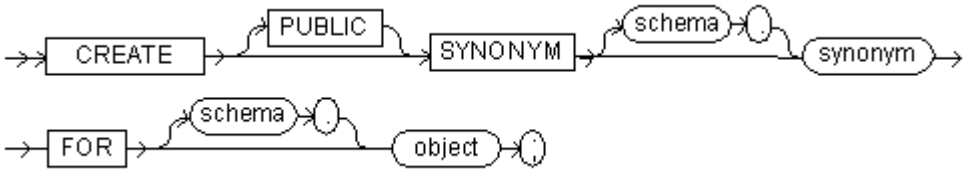
BNF Notation

```
[ expr [, expr]...]
```

F.5 Multipart Diagrams

Read a multipart diagram as if all the main paths were joined end to end. The example in Figure F–5 is a two-part diagram:

Figure F–5 Syntax for a Multipart Diagram



BNF Notation

```
CREATE [PUBLIC] SYNONYM [schema ] synonym
FOR [schema ] object ;
```

According to the diagram, the following statements are valid:

```
CREATE SYNONYM prod FOR product;
CREATE SYNONYM prod FOR scott.product;
CREATE SYNONYM scott.prod FOR scott.product;
```

F.6 Database Objects

The names of Oracle Lite identifiers, such as tables and columns, must not exceed 30 characters in length. The first character must be a letter, but the rest can be any combination of letters, numerals, dollar signs ($), pound signs (#), and underscores (_).

However, if an Oracle Lite identifier is enclosed by double quotation marks ("), it can contain any combination of legal characters, including spaces but excluding quotation marks. Oracle Lite identifiers are not case-sensitive except when enclosed by double quotation marks.

F.7 BNF Notation

The syntax diagrams in this document use a variation of Backus-Nauer Form (BNF) notation. For a description of the convention used in this document, please see Section 4.2.6, "BNF Notation Conventions".

Glossary

base table

A source of data, either a table or a view, that underlies a view. When you access data in a view, you are really accessing data from its base tables. You specify a view's base tables in CREATE VIEW.

database object

A database object is a named database structure: a table, view, sequence, index, snapshot, or synonym.

foreign key

A foreign key is a column or group of columns in one table or view whose values provide a reference to the rows in another table or view. A foreign key generally contains a value that matches a primary key value in another table.

index

An index is a database object that provides fast access to individual rows in a table. You create an index to accelerate the queries and sorting operations performed against the table's data. You also use indexes to enforce certain constraints on tables, such as unique and primary key constraints.

Indexes, once created, are automatically maintained and used for data access by the database engine whenever possible.

integrity constraint

An integrity constraint is a rule that restricts the values that can be entered into one or more columns of a table.

join

A relationship established between keys (both primary and foreign) in two different tables or views. Joins are used to link tables that have been normalized to eliminate redundant data in a relational database. A common type of join links the primary key in one table to the foreign key in another table to establish a master-detail relationship. A join corresponds to a WHERE clause condition in a SQL statement.

master-detail relationship

A master-detail relationship exists between tables or views in a database when multiple rows in one table or view (the detail table or view) are associated with a single master row in another table or view (the master table or view).

Master and detail rows are normally joined by a primary key column in the master table or view that matches a foreign key column in the detail table or view.

When you change values for the primary key, the application should query a new set of detail records, so that values in the foreign key match values in the primary key. For example, if detail records in the EMP table are to be kept synchronized with master records in the DEPT table, the primary key in DEPT should be DEPTNO, and the foreign key in EMP should be DEPTNO.

positioned DELETE

A positioned DELETE statement deletes the current row of the cursor. Its format is:

```
DELETE FROM table
   WHERE CURRENT OF cursor_name
```

positioned UPDATE

A positioned UPDATE statement updates the current row of the cursor. Its format is:

```
UPDATE table SET set_list
   WHERE CURRENT OF cursor_name
```

primary key

A table's primary key is a column or group of columns used to uniquely identify each row in the table. The primary key provides fast access to the table's records, and is frequently used as the basis of a join between two tables or views. Only one primary key may be defined for each table.

To satisfy a PRIMARY KEY constraint, no primary key value can appear in more than one row of the table, and no column that is part of the primary key can contain a NULL value.

referential integrity

Referential integrity is defined as the accuracy of links between tables in a master-detail relationship that is maintained when records are added, modified, or deleted.

Carefully defined master-detail relationships promote referential integrity. Constraints in your database enforce referential integrity at the database (the server in a client/server environment).

The goal of referential integrity is to prevent the creation of an orphan record, which is a detail record that has no valid link to a master record. Rules that enforce referential integrity prevent the deletion or update of a master record, or the insertion or update of a detail record, that creates an orphan record.

schema

A schema is a named collection of database objects, including tables, views, indexes, and sequences.

sequence

A sequence is a database object that generates a series of unique integers. Sequences are typically used to generate data values that are required to be unique, such as primary key values.

SQL

SQL, or Structured Query Language, is a non-procedural database access language used by most relational database engines. Statements in SQL describe operations to be performed on sets of data. When a SQL statement is sent to a database, the database engine automatically generates a procedure to perform the specified tasks.

synonym

A synonym is an alternative name, or alias, for a table, view, sequence, snapshot, or another synonym.

table

A table is a database object that stores data that is organized into rows and columns. In a well designed database, each table stores information about a single topic (such as company employees or customer addresses).

transaction

A set of changes made to selected data in a relational database. Transactions are usually executed with a SQL statement such as ADD, UPDATE, or DELETE. A transaction is complete when it is either committed (the changes are made permanent) or rolled back (the changes are discarded).

A transaction is frequently preceded by a query, which selects specific records from the database that you want to change.

unique key

A table's unique key is a column or group of columns that are unique in each row of a table. To satisfy a UNIQUE KEY constraint, no unique key value can appear in more than one row of the table. However, unlike the PRIMARY KEY constraint, a unique key made up of a single column can contain NULL values.

view

A view is a customized presentation of data selected from one or more tables (or other views). A view is like a "virtual table" that enables you to relate and combine data from multiple tables (called base tables) and views. A view is a kind of "stored query" because you can specify selection criteria for the data that the view displays.

Views, like tables, are organized into rows and columns. However, views contain no data themselves. Views allow you to treat multiple tables or views as one database object.

Index

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y