Oracle® Database Lite SQL Reference
10*g* (10.2.0)
Part No. B15917-01

4 SQL Commands

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

- Section 4.1, "SQL Command Types"
- Section 4.2, "SQL Commands Overview"
- Section 4.3, "SQL Commands Alphabetical Listing"

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::=
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:

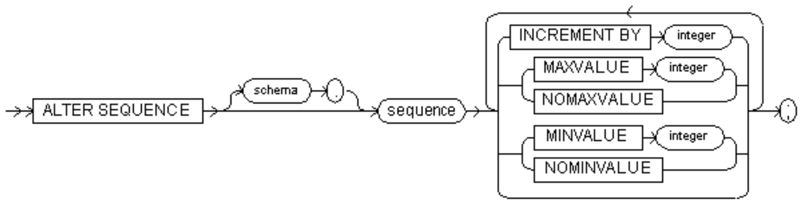
All examples refer to sample database objects supplied with Oracle Database Lite. Some DDL examples may alter the structure and data of the sample database objects. To avoid altering the sample database objects, use the `ROLLBACK` command after each DDL example that you try in the database.

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.
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

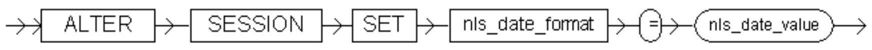
CREATE SEQUENCE, DROP SEQUENCE

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.

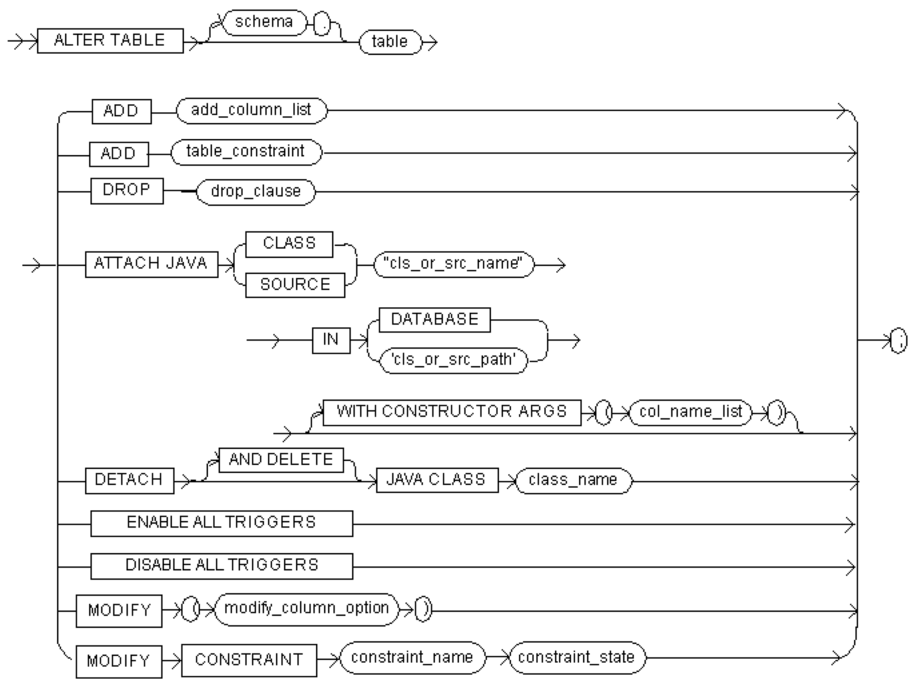
```
SELECT TO_CHAR(SYSDATE) Today FROM DUAL;

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



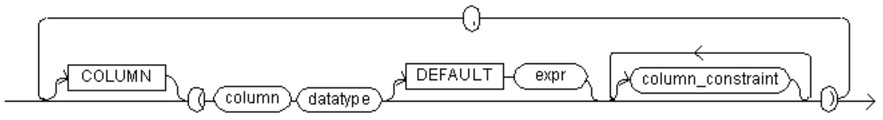
BNF Notation

```
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



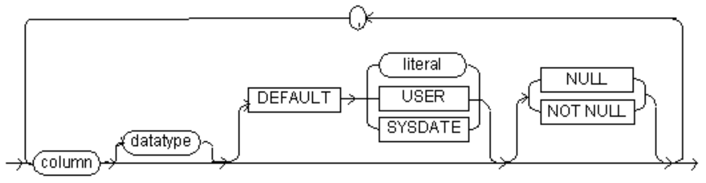
BNF Notation

```
[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



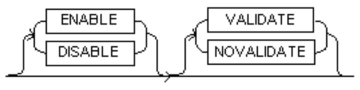
BNF Notation

```
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.
`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.

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

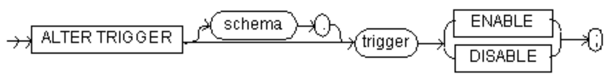
CONSTRAINT clause, CREATE TABLE, CREATE VIEW

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

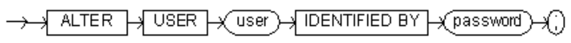
CREATE TRIGGER

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

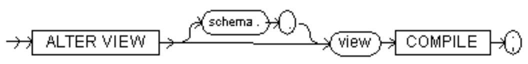
CREATE USER, DROP USER

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.
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

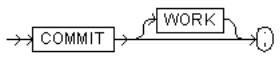
CREATE VIEW, DROP VIEW

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

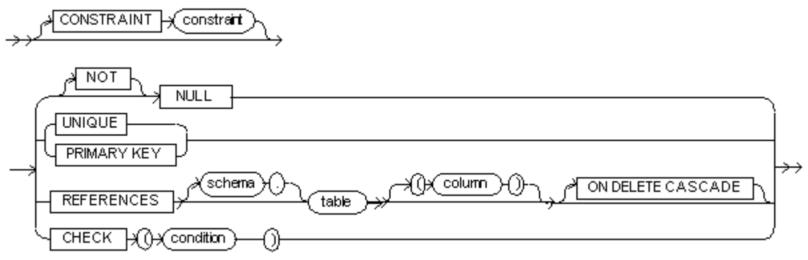
ROLLBACK

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



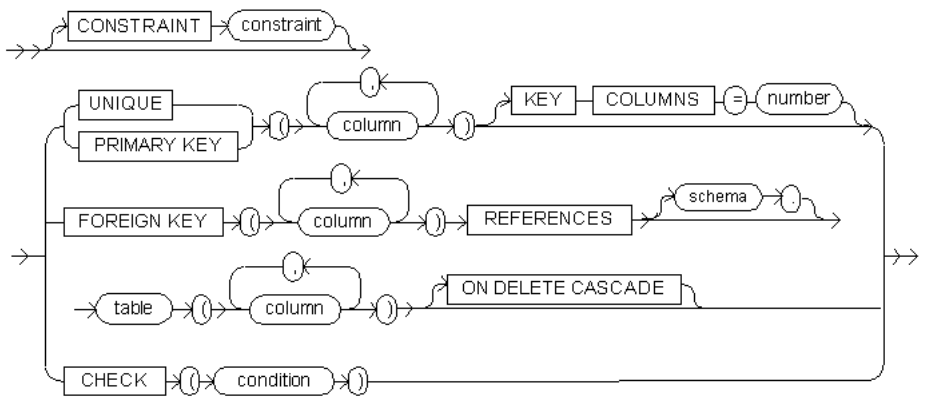
BNF Notation

```
 [CONSTRAINT constraint] { [NOT] NULL    | {UNIQUE | PRIMARY KEY}     | REFERENCES [schema .] table ["("column")"] [ON DELETE CASCADE]
```

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

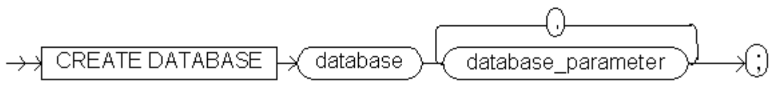
ALTER TABLE, CREATE TABLE

4.3.9 CREATE DATABASE

Syntax

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

Figure 4-13 The CREATE DATABASE Command



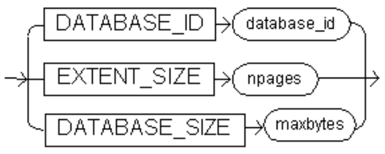
BNF Notation

```
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

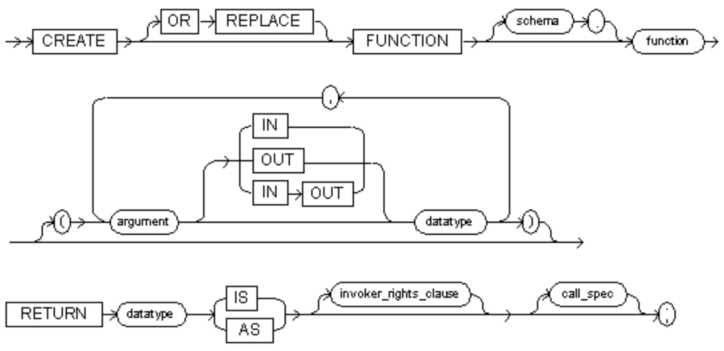
ROLLBACK

4.3.10 CREATE FUNCTION

Syntax

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

Figure 4-15 The CREATE FUNCTION Command



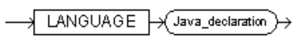
BNF Notation

```
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



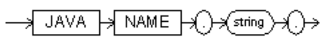
BNF Notation

```
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.
`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.

Argument	Description
`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

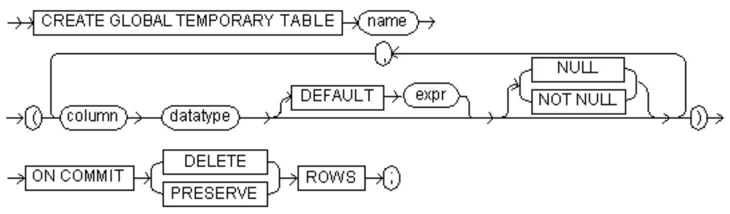
DROP FUNCTION

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.
`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.

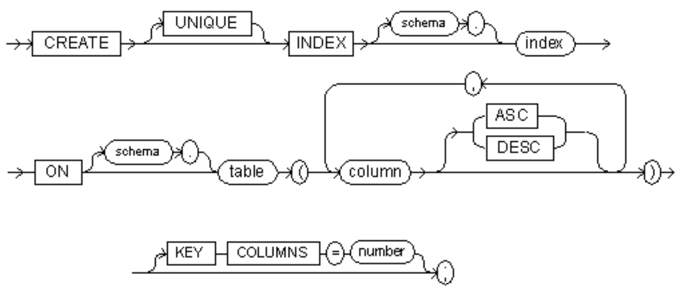
```
CREATE GLOBAL TEMPORARY TABLE flight_schedule (
startdate DATE,
enddate DATE,
cost NUMBER)
ON COMMIT PRESERVE ROWS;
```

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

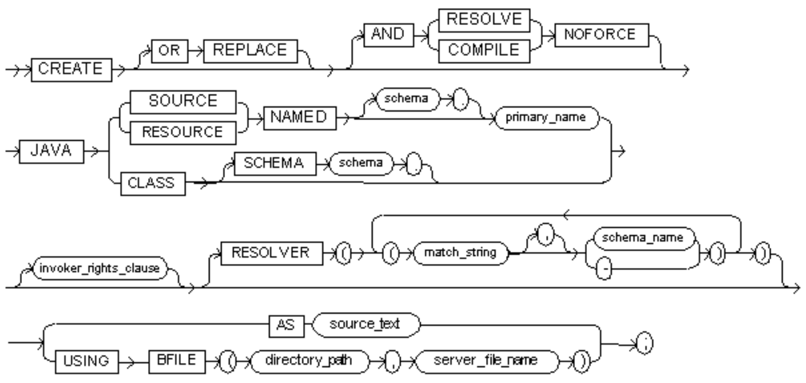
CONSTRAINT clause, CREATE TABLE, DROP INDEX

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.
`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

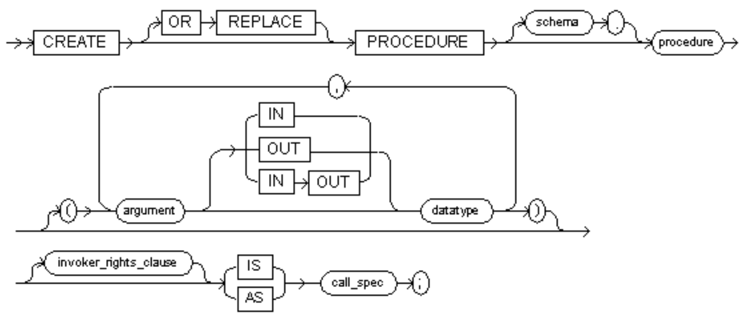
DROP JAVA

4.3.14 CREATE PROCEDURE

Syntax

The syntax for `CREATE PROCEDURE` is displayed in Figure 4-21.

Figure 4-21 The CREATE PROCEDURE Command



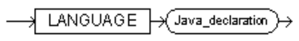
BNF Notation

```
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



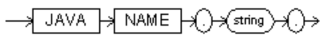
BNF Notation

```
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.

Argument	Description
`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

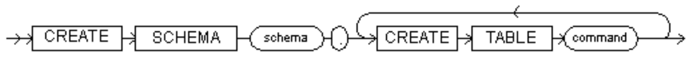
DROP PROCEDURE

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

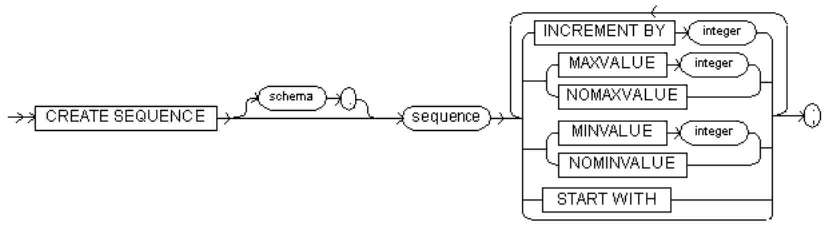
GRANT, CREATE SEQUENCE, CREATE VIEW

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

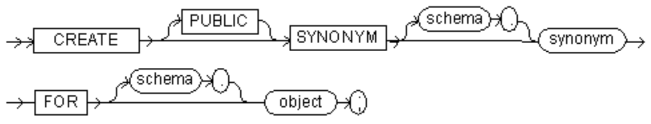
ALTER SEQUENCE, DROP SEQUENCE

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

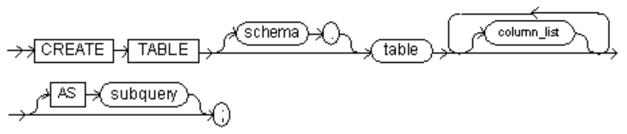
CREATE TABLE, CREATE VIEW, CREATE SEQUENCE, DROP SYNONYM

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



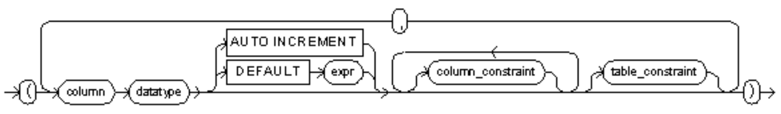
BNF Notation

```
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

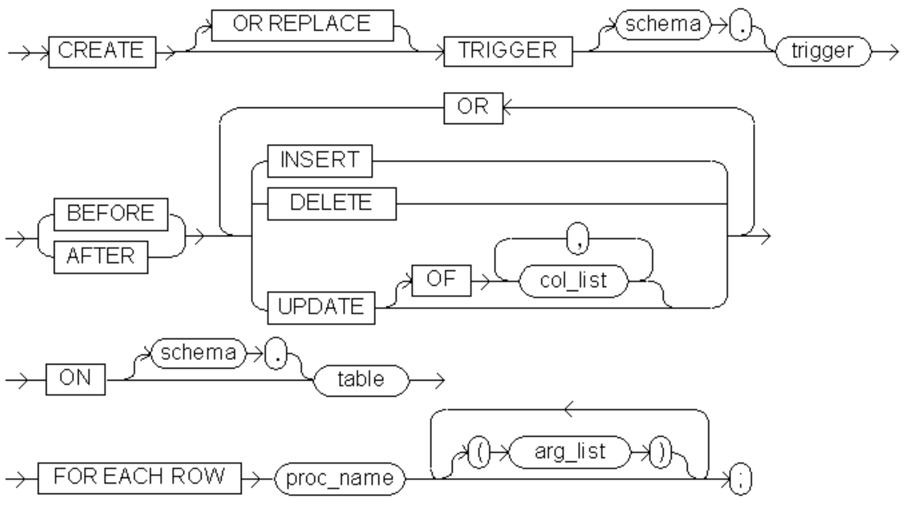
CONSTRAINT clause, DROP TABLE, Transaction Control Commands, SELECT

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

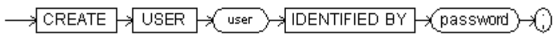
ALTER TRIGGER, ALTER VIEW, CREATE VIEW, DROP TRIGGER

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.
`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

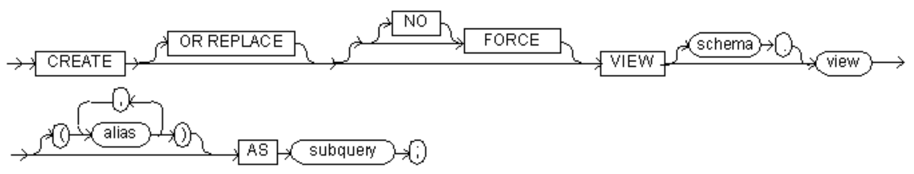
ALTER USER, GRANT

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.
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

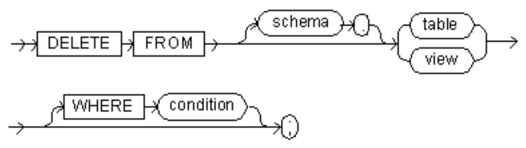
LEVEL pseudocolumn, ROWID pseudocolumn, ROWNUM pseudocolumn

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

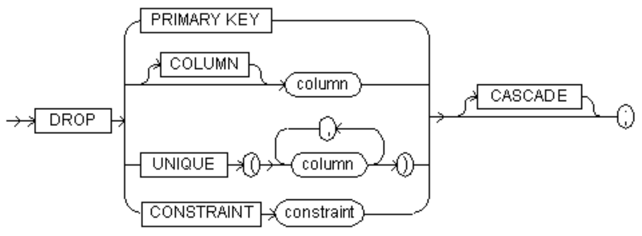
UPDATE

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.
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

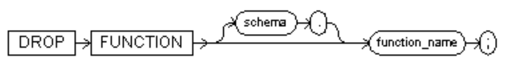
ALTER TABLE, CONSTRAINT clause

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

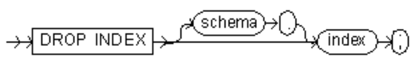
CREATE FUNCTION

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

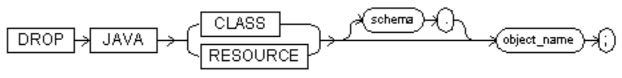
CREATE INDEX

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

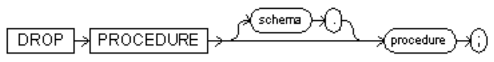
CREATE JAVA

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

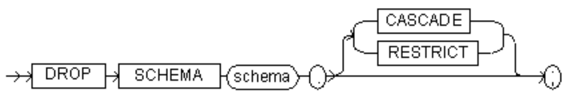
CREATE PROCEDURE

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

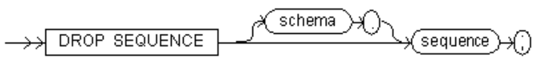
CREATE SCHEMA

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

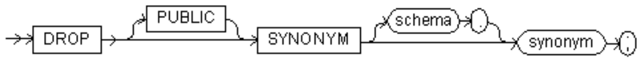
ALTER SEQUENCE, CREATE SEQUENCE

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

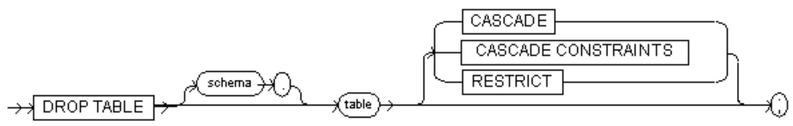
CREATE SYNONYM

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.

Argument	Description
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

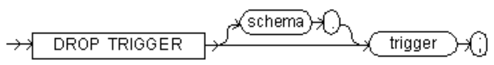
ALTER TABLE, CREATE TABLE

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

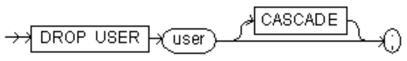
CREATE TRIGGER

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

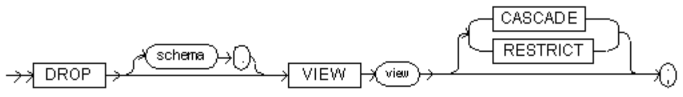
CREATE USER

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

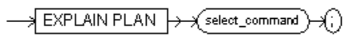
CREATE SYNONYM, CREATE TABLE, CREATE VIEW

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.

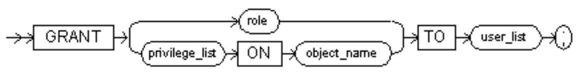
Oracle Lite uses row estimates to order tables, however, the actual values are not important. The optimizer estimates the best possible index. The object kernel may choose a different index since it is more accurate at execution time.

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

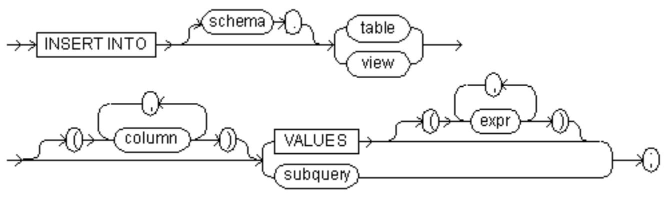
REVOKE

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

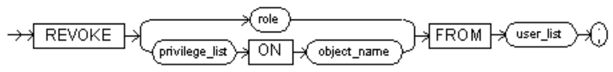
CURRVAL and NEXTVAL pseudocolumns, LEVEL pseudocolumn, ROWID pseudocolumn, ROWNUM pseudocolumn

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

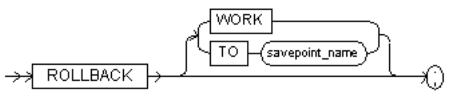
GRANT

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

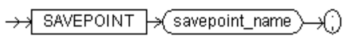
CURRVAL and NEXTVAL pseudocolumns, LEVEL pseudocolumn, ROWID pseudocolumn, OL__ROW_STATUS pseudocolumn

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

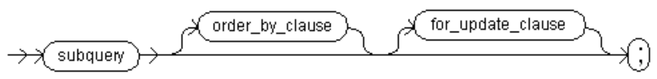
The SELECT statement retrieves data from one or more tables or views. You can also use the select statement to invoke Java stored procedures. To select data from a table or view, you must be logged into the database as SYSTEM, or the table(s) and view(s) must be part of your schema.

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.
*	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".
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.

Argument	Description
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.

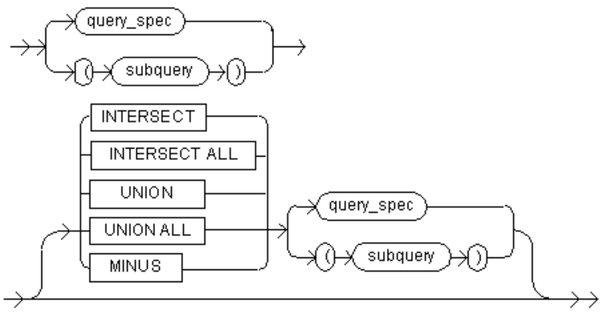
When you select columns with an expression, those columns must have an alias. An alias specifies names for the column expressions selected by the query. The number of aliases must match the number of expressions selected by the query. Aliases must be unique within the query.

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



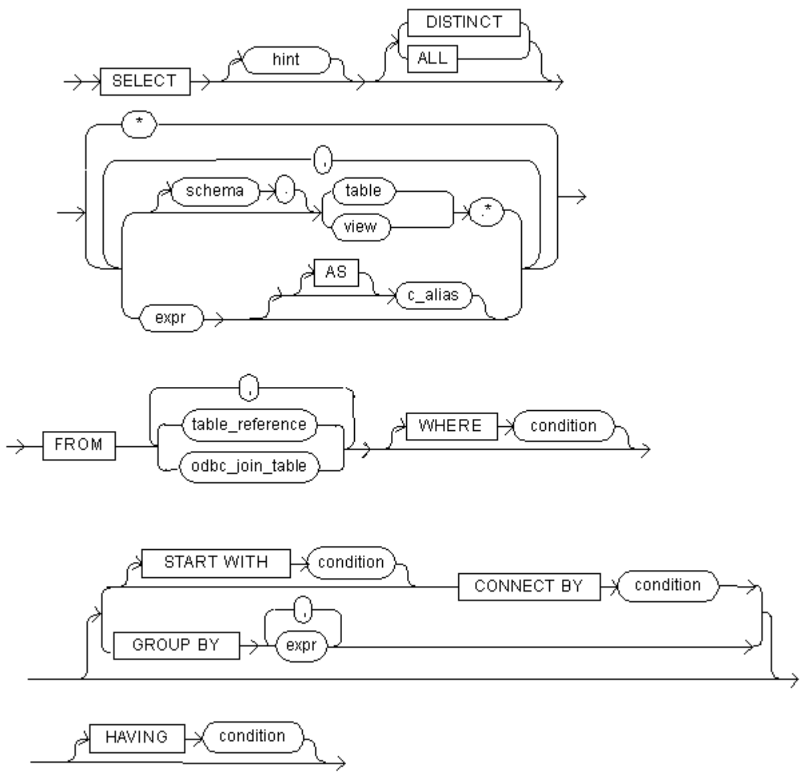
BNF Notation

```
{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

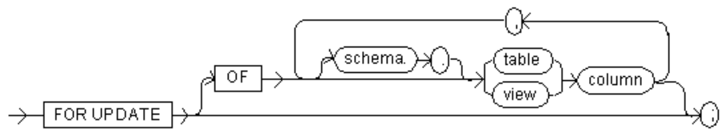
```
SELECT [ hint ] [ { DISTINCT | ALL ]{ * | { [schema.] { table | view } .*       | expr [[AS] c_alias]  }   [, {
     | [schema .] { table | view  } .*       | expr [[AS] c_alias]     }  ]...}FROM  [schema .] { "("subquery [order_by_clause] ")" | tabl
```

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

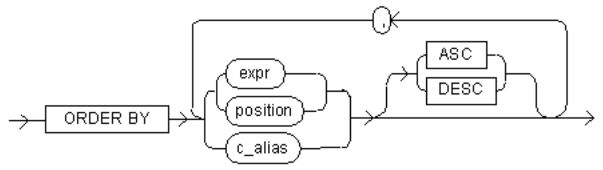
```
FOR UPDATE[OF [[schema .] { table | view } .] column  [, [[schema .] { table | view } .] column]...]
```

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

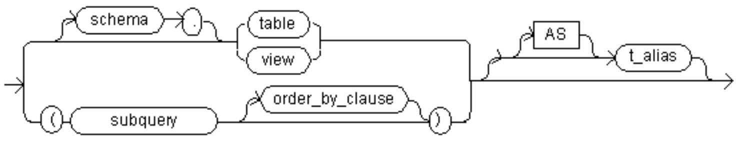
```
ORDER  BY { expr | position | c_alias } [ ASC | DESC ] [, { expr | position | c_alias } [ ASC | DESC ] ]...
```

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

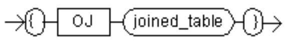
```
{ [schema .] {table | view} | "("subquery [order_by_clause] ")"} [[AS] t_alias]
```

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

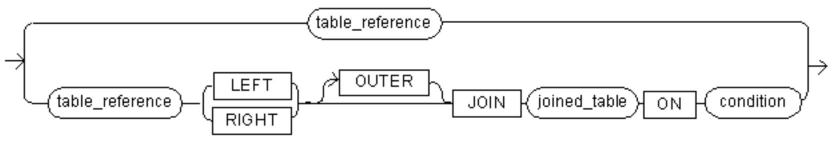
```
"{" OJ joined_table "}"
```

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, l.unit_price * l.quantity
FROM customers c, order_items l, orders o
WHERE c.cust_last_name = ?
AND o.customer_id = c.customer_id
AND o.order_id = l.order_id;
```

4.3.46.8.2 INDEX Hints

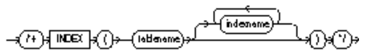
Index hints explicitly choose an index scan for the specified table. The following are Index hints:

- 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.

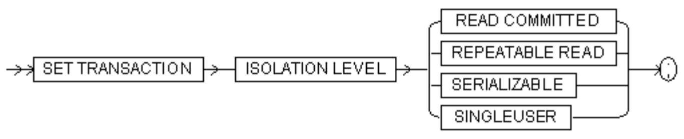
```
Select CustId, Name, Phone from Customer
Where CustId In ( Select //ordered// Ord.CustId from Product, Item, Ord

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.
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

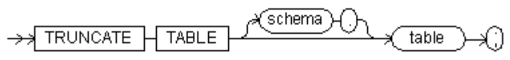
COMMIT, ROLLBACK

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

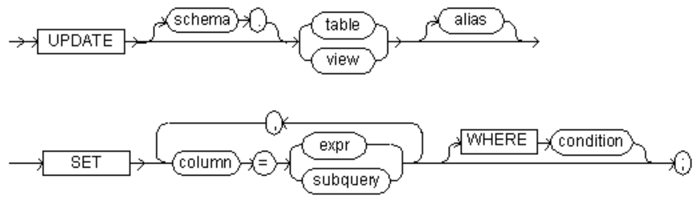
```
TRUNCATE TABLE emp;
```

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.
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

Previous Next

Contents Index