3

About DML Statements and Transactions

Data manipulation language (DML) statements add, change, and delete Oracle Database table data. A **transaction** is a sequence of one or more SQL statements that Oracle Database treats as a unit: either all of the statements are performed, or none of them are.

About Data Manipulation Language (DML) Statements

Data manipulation language (DML) statements access and manipulate data in existing tables.

In the SQL*Plus environment, you can enter a DML statement after the `SQL>` prompt.

In the SQL Developer environment, you can enter a DML statement in the Worksheet. Alternatively, you can use the SQL Developer Connections frame and tools to access and manipulate data.

To see the effect of a DML statement in SQL Developer, you might have to select the schema object type of the changed object in the Connections frame and then click the Refresh icon.

The effect of a DML statement is not permanent until you commit the transaction that includes it. A **transaction** is a sequence of SQL statements that Oracle Database treats as a unit (it can be a single DML statement). Until a transaction is committed, it can be rolled back (undone). For more information about transactions, see "About Transaction Control Statements".

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about DML statements

About the INSERT Statement

The INSERT statement inserts rows into an existing table.

The simplest recommended form of the INSERT statement has this syntax:

```
INSERT INTO table_name (list_of_columns)
VALUES (list_of_values);
```

Every column in list_of_columns must have a valid value in the corresponding position in list_of_values. Therefore, before you insert a row into a table, you must know what columns the table has, and what their valid values are. To get this information using SQL Developer, see "Tutorial: Viewing EMPLOYEES Table Properties and Data with SQL Developer". To get this information using SQL*Plus, use the DESCRIBE statement. For example:

```
DESCRIBE EMPLOYEES;
```

Result:

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------

 EMPLOYEE_ID                               NOT NULL NUMBER(6)
 FIRST_NAME                                         VARCHAR2(20)
 LAST_NAME                                 NOT NULL VARCHAR2(25)
 EMAIL                                     NOT NULL VARCHAR2(25)
 PHONE_NUMBER                                       VARCHAR2(20)
 HIRE_DATE                                 NOT NULL DATE
 JOB_ID                                    NOT NULL VARCHAR2(10)
 SALARY                                             NUMBER(8,2)
 COMMISSION_PCT                                     NUMBER(2,2)
 MANAGER_ID                                         NUMBER(6)
 DEPARTMENT_ID                                      NUMBER(4)
```

The INSERT statement in Example 3-1 inserts a row into the EMPLOYEES table for an employee for which all column values are known.

You need not know all column values to insert a row into a table, but you must know the values of all NOT NULL columns. If you do not know the value of a column that can be NULL, you can omit that column from list_of_columns. Its value defaults to NULL.

The INSERT statement in Example 3-2 inserts a row into the EMPLOYEES table for an employee for which all column values are known except SALARY. For now, SALARY can have the value NULL. When you know the salary, you can change it with the UPDATE statement (see Example 3-4).

The INSERT statement in Example 3-3 tries to insert a row into the EMPLOYEES table for an employee for which LAST_NAME is not known.

Example 3-1 Using the INSERT Statement When All Information Is Available

```
INSERT INTO EMPLOYEES (
  EMPLOYEE_ID,
  FIRST_NAME,
  LAST_NAME,
  EMAIL,
  PHONE_NUMBER,
  HIRE_DATE,
  JOB_ID,
  SALARY,
  COMMISSION_PCT,
  MANAGER_ID,
  DEPARTMENT_ID
)
VALUES (
  10,              -- EMPLOYEE_ID
  'George',        -- FIRST_NAME
  'Gordon',        -- LAST_NAME
  'GGORDON',       -- EMAIL
  '650.506.2222',  -- PHONE_NUMBER
  '01-JAN-07',     -- HIRE_DATE
  'SA_REP',        -- JOB_ID
  9000,            -- SALARY
  .1,              -- COMMISSION_PCT
  148,             -- MANAGER_ID
  80               -- DEPARTMENT_ID
);
```

Result:

```
1 row created.
```

Example 3-2 Using the INSERT Statement When Not All Information Is Available

```
INSERT INTO EMPLOYEES (
  EMPLOYEE_ID,
  FIRST_NAME,
  LAST_NAME,
  EMAIL,
  PHONE_NUMBER,
  HIRE_DATE,
  JOB_ID,           -- Omit SALARY; its value defaults to NULL.
  COMMISSION_PCT,
  MANAGER_ID,
  DEPARTMENT_ID
)
VALUES (
  20,               -- EMPLOYEE_ID
  'John',           -- FIRST_NAME
  'Keats',          -- LAST_NAME
  'JKEATS',         -- EMAIL
  '650.506.3333',   -- PHONE_NUMBER
  '01-JAN-07',      -- HIRE_DATE
  'SA_REP',         -- JOB_ID
  .1,               -- COMMISSION_PCT
  148,              -- MANAGER_ID
  80                -- DEPARTMENT_ID
);
```

Result:

```
1 row created.
```

Example 3-3 Using the INSERT Statement Incorrectly

```
INSERT INTO EMPLOYEES (
  EMPLOYEE_ID,
  FIRST_NAME,       -- Omit LAST_NAME (error)
  EMAIL,
  PHONE_NUMBER,
  HIRE_DATE,
  JOB_ID,
  COMMISSION_PCT,
  MANAGER_ID,
  DEPARTMENT_ID
)
VALUES (
  20,               -- EMPLOYEE_ID
  'John',           -- FIRST_NAME
  'JOHN',           -- EMAIL
  '650.506.3333',   -- PHONE_NUMBER
  '01-JAN-07',      -- HIRE_DATE
  'SA_REP',         -- JOB_ID
  .1,               -- COMMISSION_PCT
  148,              -- MANAGER_ID
  80                -- DEPARTMENT_ID
);
```

Result:

```
ORA-01400: cannot insert NULL into ("HR"."EMPLOYEES"."LAST_NAME")
```

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for information about the INSERT statement
> - *Oracle Database SQL Language Reference* for information about data types
> - "Tutorial: Adding Rows to Tables with the Insert Row Tool"

About the UPDATE Statement

The UPDATE statement updates (changes the values of) a set of existing table rows.

A simple form of the UPDATE statement has this syntax:

```
UPDATE table_name
SET column_name = value [, column_name = value]...
[ WHERE condition ];
```

Each value must be valid for its column_name. If you include the WHERE clause, the statement updates column values only in rows that satisfy condition.

The UPDATE statement in Example 3-4 updates the value of the SALARY column in the row that was inserted into the EMPLOYEES table in Example 3-2, before the salary of the employee was known.

The UPDATE statement in Example 3-5 updates the commission percentage for every employee in department 80.

Example 3-4 Using the UPDATE Statement to Add Data

```
UPDATE EMPLOYEES
SET SALARY = 8500
WHERE LAST_NAME = 'Keats';
```

Result:

```
1 row updated.
```

Example 3-5 Using the UPDATE Statement to Update Multiple Rows

```
UPDATE EMPLOYEES
SET COMMISSION_PCT = COMMISSION_PCT + 0.05
WHERE DEPARTMENT_ID = 80;
```

Result:

```
34 rows updated.
```

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for information about the UPDATE statement
> - *Oracle Database SQL Language Reference* for information about data types
> - "Tutorial: Changing Data in Tables in the Data Pane"

About the DELETE Statement

The DELETE statement deletes rows from a table.

A simple form of the DELETE statement has this syntax:

```
DELETE FROM table_name [ WHERE condition ];
```

If you include the WHERE clause, the statement deletes only rows that satisfy condition. If you omit the WHERE clause, the statement deletes all rows from the table, but the empty table still exists. To delete a table, use the DROP TABLE statement.

The DELETE statement in Example 3-6 deletes the rows inserted in Example 3-1 and Example 3-2.

Example 3-6 Using the DELETE Statement

```
DELETE FROM EMPLOYEES
WHERE HIRE_DATE = TO_DATE('01-JAN-07', 'dd-mon-yy');
```

Result:

```
2 rows deleted.
```

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for information about the DELETE statement
> - *Oracle Database SQL Language Reference* for information about the DROP TABLE statement
> - "Tutorial: Deleting Rows from Tables with the Delete Selected Row(s) Tool"

About Transaction Control Statements

A **transaction** is a sequence of one or more SQL statements that Oracle Database treats as a unit: either all of the statements are performed, or none of them are. You need transactions to model business processes that require that several operations be performed as a unit.

For example, when a manager leaves the company, a row must be inserted into the JOB_HISTORY table to show when the manager left, and for every employee who reports to that manager, the value of MANAGER_ID must be updated in the EMPLOYEES table. To

model this process in an application, you must group the INSERT and UPDATE statements into a single transaction.

The basic **transaction control statements** are:

- SAVEPOINT, which marks a **savepoint** in a transaction—a point to which you can later roll back. Savepoints are optional, and a transaction can have multiple savepoints.
- COMMIT, which ends the current transaction, makes its changes permanent, erases its savepoints, and releases its locks.
- ROLLBACK, which rolls back (undoes) either the entire current transaction or only the changes made after the specified savepoint.

In the SQL*Plus environment, you can enter a transaction control statement after the `SQL>` prompt.

In the SQL Developer environment, you can enter a transaction control statement in the Worksheet. SQL Developer also has Commit Changes and Rollback Changes icons, which are explained in "Committing Transactions" and "Rolling Back Transactions".

Caution:

If you do not explicitly commit a transaction, and the program terminates abnormally, then the database automatically rolls back the last uncommitted transaction.

Oracle recommends that you explicitly end transactions in application programs, by either committing them or rolling them back.

See Also:

- *Oracle Database Concepts* for more information about transaction management
- *Oracle Database SQL Language Reference* for more information about transaction control statements

Committing Transactions

Committing a transaction makes its changes permanent, erases its savepoints, and releases its locks.

To explicitly commit a transaction, use either the COMMIT statement or (in the SQL Developer environment) the **Commit Changes** icon.

> **Note:**
>
> Oracle Database issues an implicit COMMIT statement before and after any data definition language (DDL) statement. For information about DDL statements, see "About Data Definition Language (DDL) Statements".

Before you commit a transaction:

- Your changes are visible to you, but not to other users of the database instance.
- Your changes are not final—you can undo them with a ROLLBACK statement.

After you commit a transaction:

- Your changes are visible to other users, and to their statements that run after you commit your transaction.
- Your changes are final—you cannot undo them with a ROLLBACK statement.

Example 3-7 adds one row to the REGIONS table (a very simple transaction), checks the result, and then commits the transaction.

Example 3-7 Committing a Transaction

Before transaction:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa

4 rows selected.
```

Transaction (add row to table):

```
INSERT INTO regions (region_id, region_name) VALUES (5, 'Africa');
```

Result:

```
1 row created.
```

Check that row was added:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
```

```
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

Commit transaction:

```
COMMIT;
```

Result:

```
Commit complete.
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the COMMIT statement

Rolling Back Transactions

Rolling back a transaction undoes its changes. You can roll back the entire current transaction, or you can roll it back only to a specified savepoint.

To roll back the current transaction only to a specified savepoint, you must use the ROLLBACK statement with the TO SAVEPOINT clause.

To roll back the entire current transaction, use either the ROLLBACK statement without the TO SAVEPOINT clause, or (in the SQL Developer environment) the **Rollback Changes** icon.

Rolling back the entire current transaction:

- Ends the transaction
- Reverses all of its changes
- Erases all of its savepoints
- Releases any transaction locks

Rolling back the current transaction only to the specified savepoint:

- Does not end the transaction
- Reverses only the changes made after the specified savepoint
- Erases only the savepoints set after the specified savepoint (excluding the specified savepoint itself)
- Releases all table and row locks acquired after the specified savepoint

  Other transactions that have requested access to rows locked after the specified savepoint must continue to wait until the transaction is either committed or rolled back. Other transactions that have not requested the rows can request and access the rows immediately.

To see the effect of a rollback in SQL Developer, you might have to click the **Refresh** icon.

As a result of Example 3-7, the REGIONS table has a region called 'Middle East and Africa' and a region called 'Africa'. Example 3-8 corrects this problem (a very simple transaction) and checks the change, but then rolls back the transaction and checks the rollback.

Example 3-8 Rolling Back an Entire Transaction

Before transaction:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

Transaction (change table):

```
UPDATE REGIONS
SET REGION_NAME = 'Middle East'
WHERE REGION_NAME = 'Middle East and Africa';
```

Result:

```
1 row updated.
```

Check change:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East
         5 Africa

5 rows selected.
```

Roll back transaction:

```
ROLLBACK;
```

Result:

```
Rollback complete.
```

Check rollback:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the ROLLBACK statement

Setting Savepoints in Transactions

The SAVEPOINT statement marks a **savepoint** in a transaction—a point to which you can later roll back. Savepoints are optional, and a transaction can have multiple savepoints.

Example 3-9 does a transaction that includes several DML statements and several savepoints, and then rolls back the transaction to one savepoint, undoing only the changes made after that savepoint.

Example 3-9 Rolling Back a Transaction to a Savepoint

Check REGIONS table before transaction:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

Check countries in region 4 before transaction:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 4
ORDER BY COUNTRY_NAME;
```

Result:

```
COUNTRY_NAME                             CO  REGION_ID
---------------------------------------- -- ----------
```

```
Egypt                                     EG          4
Israel                                    IL          4
Kuwait                                    KW          4
Nigeria                                   NG          4
Zambia                                    ZM          4
Zimbabwe                                  ZW          4

6 rows selected.
```

Check countries in region 5 before transaction:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 5
ORDER BY COUNTRY_NAME;
```

Result:

```
no rows selected
```

Transaction, with several savepoints:

```
UPDATE REGIONS
SET REGION_NAME = 'Middle East'
WHERE REGION_NAME = 'Middle East and Africa';

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'ZM';
SAVEPOINT zambia;

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'NG';
SAVEPOINT nigeria;

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'ZW';
SAVEPOINT zimbabwe;

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'EG';
SAVEPOINT egypt;
```

Check REGIONS table after transaction:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East
         5 Africa

5 rows selected.
```

Check countries in region 4 after transaction:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 4
ORDER BY COUNTRY_NAME;
```

Result:

```
COUNTRY_NAME                           CO  REGION_ID
---------------------------------------- -- ----------
Israel                                 IL          4
Kuwait                                 KW          4

2 rows selected.
```

Check countries in region 5 after transaction:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 5
ORDER BY COUNTRY_NAME;
```

Result:

```
COUNTRY_NAME                           CO  REGION_ID
---------------------------------------- -- ----------
Egypt                                  EG          5
Nigeria                                NG          5
Zambia                                 ZM          5
Zimbabwe                               ZW          5

4 rows selected.
```

```
ROLLBACK TO SAVEPOINT nigeria;
```

Check REGIONS table after rollback:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

Result:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East
         5 Africa

5 rows selected.
```

Check countries in region 4 after rollback:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 4
ORDER BY COUNTRY_NAME;
```

Result:

```
COUNTRY_NAME                             CO  REGION_ID
---------------------------------------- -- ----------
Egypt                                    EG          4
Israel                                   IL          4
Kuwait                                   KW          4
Zimbabwe                                 ZW          4

4 rows selected.
```

Check countries in region 5 after rollback:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 5
ORDER BY COUNTRY_NAME;
```

Result:

```
COUNTRY_NAME                             CO  REGION_ID
---------------------------------------- -- ----------
Nigeria                                  NG          5
Zambia                                   ZM          5

2 rows selected.
```

See Also:

Oracle Database SQL Language Reference for information about the SAVEPOINT statement