



Lab 03: DML Queries

Objective(s) :

To learn the Data Manipulation Language

- Insert rows into a table
- Update rows in a table
- Delete rows from a table

DML

Data Manipulation Language (DML) besides of the **SELECT** statement that retrieves information from databases includes also statements modifying data state. These statements are:

INSERT	Inserts rows into database table
UPDATE	Changes values in columns of database table
DELETE	Deletes rows from database table

Data Manipulation Language


Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a transaction.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decrease the savings account, increase the checking account, and record the transaction in the transaction journal. The Oracle server must guarantee that all three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

Adding a New Row

				70	Public Relations	100	1700	New row
DEPARTMENTS								
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID					
10	Administration	200	1700					
20	Marketing	201	1800					
50	Shipping	124	1500					
60	IT	103	1400					
80	Sales	149	2500					
90	Executive	100	1700					
110	Accounting	205	1700					
190	Contracting		1700					

...insert a new row into the DEPARTMENTS table...



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID					
10	Administration	200	1700					
20	Marketing	201	1800					
50	Shipping	124	1500					
60	IT	103	1400					
80	Sales	149	2500					
90	Executive	100	1700					
110	Accounting	205	1700					
190	Contracting		1700					
70	Public Relations	100	1700					

You can add new rows to a table by issuing the `INSERT` statement.

```
INSERT INTO table [(column [, column...])]
VALUES (value [, value...]);
```

In the syntax:

table is the name of the table

column is the name of the column in the table to populate

value is the corresponding value for the column

Note: This statement with the `VALUES` clause adds only one row at a time to a table.

Because you can insert a new row that contains values for each column, the column list is not required in the `INSERT` clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

For clarity, use the column list in the `INSERT` clause. Enclose character and date values within single quotation marks; it is not recommended to enclose numeric values within single quotation marks.

Number values should not be enclosed in single quotes, because implicit conversion may take place for numeric values assigned to `NUMBER` data type columns if single quotes are included.

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
```

1 row created.

Inserting Rows with Null Values

Methods for Inserting Null Values

Implicit: Omit the column from the column list.

```
INSERT INTO departments (department_id,
                        department_name )
VALUES (30, 'Purchasing');
```

Explicit: Specify the `NULL` keyword in the `VALUES` list, specify the empty string (`' '`) in the `VALUES` list for character strings and dates.

```
INSERT INTO departments
VALUES (100, 'Finance', NULL, NULL);
```

1 row created.

Be sure that you can use null values in the targeted column by verifying the `Null?` status with the `iSQL*Plus DESCRIBE` command.

The Oracle Server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

Common errors that can occur during user input:

- Mandatory value missing for a `NOT NULL` column
- Duplicate value violates uniqueness constraint
- Foreign key constraint violated
- `CHECK` constraint violated
- Data type mismatch
- Value too wide to fit in column

Inserting Special Values

```
INSERT INTO employees (employee_id,first_name, last_name, email,
phone_number,hire_date, job_id, salary, commission_pct, manager_id,
department_id)
VALUES (113, 'Louis', 'Popp', 'LPOPP', '515.124.4567', SYSDATE, 'AC_ACCOUNT', 6900,
NULL, 205, 100);
```

1 row created.

You can use functions to enter special values in your table. The above example records information for employee Popp in the `EMPLOYEES` table. It supplies the current date and time in the `HIRE_DATE` column. It uses the `SYSDATE` function for current date and time.

You can also use the `USER` function when inserting rows in a table. The `USER` function records the current username.

Inserting Specific Date Values

The `DD-MON-YY` format is usually used to insert a date value. With this format, recall that the century defaults to the current century. Because the date also contains time information, the default time is midnight (00:00:00).

If a date must be entered in a format other than the default format, for example, with another century, or a specific time, you must use the `TO_DATE` function.

```
INSERT INTO employees
VALUES (114, 'Den', 'Rapealy', 'DRAPHEAL', '515.127.4561', TO_DATE('FEB 3,
1999', 'MON DD, YYYY'), 'AC_ACCOUNT', 11000, NULL, 100, 30);
```

1 row created.

The example on the above records information for employee Rapealy in the `EMPLOYEES` table. It

sets the `HIRE_DATE` column to be February 3, 1999.

Copying Rows from Another Table

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

You can use the `INSERT` statement to add rows to a table where the values are derived from existing tables. In place of the `VALUES` clause, you use a subquery.

Syntax

```
INSERT INTO table [ column (, column) ] subquery;
```

In the syntax:

table is the table name

column is the name of the column in the table to populate

subquery is the subquery that returns rows into the table

The number of columns and their data types in the column list of the `INSERT` clause must match the

number of values and their data types in the subquery. To create a copy of the rows of a table, use `SELECT *` in the subquery.

```
INSERT INTO copy_emp
SELECT *
FROM employees;
```

Changing Data in a Table

```
UPDATE table
```

```
SET column = value [, column = value, ...]
```

```
[WHERE condition];
```

You can modify existing rows by using the `UPDATE` statement.

In the syntax:

table is the name of the table

column is the name of the column in the table to populate

value is the corresponding value or subquery for the column

condition identifies the rows to be updated and is composed of column names expressions, constants, subqueries, and comparison operators

Confirm the update operation by querying the table to display the updated rows.

Note: In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the `EMPLOYEES` table by name is dangerous, because more than one employee may have the same name.

```
UPDATE employees
SET department_id = 70
WHERE employee_id = 113;
1 row updated.
```

The `UPDATE` statement modifies specific rows if the `WHERE` clause is specified. The slide example transfers employee 113 (Popp) to department 70.

If you omit the `WHERE` clause, all the rows in the table are modified.

```
UPDATE copy_emp
SET department_id = 110;
22 rows updated.
```

Updating Two Columns with a Subquery

```
UPDATE employees
SET job_id = (SELECT job_id
FROM employees
WHERE employee_id = 205),
salary = (SELECT salary
FROM employees
WHERE employee_id = 205)
WHERE employee_id = 114;
```

1 row updated.

You can update multiple columns in the `SET` clause of an `UPDATE` statement by writing multiple subqueries.

Syntax

```
UPDATE table
SET column =
(SELECT column
FROM table
WHERE condition)
[ ,
column =
(SELECT column
FROM table
WHERE condition) ]
[WHERE condition] ;
```

Note: If no rows are updated, a message “0 rows updated.” is returned.

Updating Rows Based on Another Table

```
UPDATE copy_emp
SET department_id = (SELECT department_id
FROM employees
WHERE employee_id = 100)
WHERE job_id = (SELECT job_id
FROM employees
WHERE employee_id = 200);
```

You can use subqueries in `UPDATE` statements to update rows in a table. The above example updates the `COPY_EMP` table based on the values from the `EMPLOYEES` table. It changes the department number of all employees with employee 200's job ID to employee 100's current department number.

Updating Rows: Integrity Constraint Error

```
UPDATE employees
SET department_id = 55
WHERE department_id = 110;
```

```
UPDATE employees
```

```
*
```

```
ERROR at line 1:
```

```
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

If you attempt to update a record with a value that is tied to an integrity constraint, an error is returned.

In the above example, department number 55 does not exist in the parent table, `DEPARTMENTS`, and so you receive the *parent key violation* `ORA-02291`.

Note: Integrity constraints ensure that the data adheres to a predefined set of rules. A subsequent lesson covers integrity constraints in greater depth.

Removing a Row from a Table

Deleting Rows

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

You can remove existing rows by using the `DELETE` statement.

In the syntax:

table is the table name

condition identifies the rows to be deleted and is composed of column names, expressions, constants, subqueries, and comparison operators

Note: If no rows are deleted, a message “0 rows deleted.” is returned:

```
DELETE FROM departments
WHERE department_name = 'Finance';
```

You can delete specific rows by specifying the `WHERE` clause in the `DELETE` statement. The above example deletes the Finance department from the `DEPARTMENTS` table. You can confirm the delete operation by displaying the deleted rows using the `SELECT` statement.

```
SELECT *
FROM departments
WHERE department_name = 'Finance';
no rows selected.
```

If you omit the `WHERE` clause, all rows in the table are deleted. The following example deletes all the rows from the `COPY_EMP` table, because no `WHERE` clause has been specified.

```
DELETE FROM copy_emp;
```

22 rows deleted.

Example

Remove rows identified in the `WHERE` clause.

```
DELETE FROM employees
WHERE employee_id = 114;
1 row deleted.
DELETE FROM departments
WHERE department_id IN (30, 40);
```

Deleting Rows Based on Another Table

You can use subqueries to delete rows from a table based on values from another table. The example below deletes all the employees who are in a department where the department name contains the string “Public.” The subquery searches the `DEPARTMENTS` table to find the department number based on the department name containing the string “Public.” The subquery then feeds the department number to the main query, which deletes rows of data from the `EMPLOYEES` table based on this department number.

```
DELETE FROM employees
WHERE department_id =
(SELECT department_id
FROM departments
WHERE department_name LIKE '%Public%');
```

Deleting Rows: Integrity Constraint Error

```
DELETE FROM departments
WHERE department_id = 60;
DELETE FROM departments
*
ERROR at line 1:
ORA-02292: integrity constrain
```

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

The example above tries to delete department number 60 from the `DEPARTMENTS` table, but it results in an error because department number is used as a foreign key in the `EMPLOYEES` table. If the parent record that you attempt to delete has child records, then you receive the *child record found* violation `ORA-02292`.

The following statement works because there are no employees in department 70:

```
DELETE FROM departments
WHERE department_id = 70;
```

Using a Subquery in an `INSERT` Statement

```
INSERT INTO
(SELECT employee_id, last_name,
email, hire_date, job_id, salary,
department_id
FROM employees
WHERE department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
TO_DATE('07-JUN-99', 'DD-MON-RR'),
'ST_CLERK', 5000, 50);
1 row created.
```

You can use a subquery in place of the table name in the `INTO` clause of the `INSERT` statement. The select list of this subquery must have the same number of columns as the column list of the `VALUES` clause. Any rules on the columns of the base table must be followed in order for the `INSERT` statement to work successfully. For example, you could not put in a duplicate employee ID, nor leave out a value for a mandatory not null column.

Using the `WITH CHECK OPTION` Keyword on DML Statements

Specify `WITH CHECK OPTION` to indicate that, if the subquery is used in place of a table in an `INSERT`, `UPDATE`, or `DELETE` statement, no changes that would produce rows that are not included in the subquery are permitted to that table.

```
INSERT INTO (SELECT employee_id, last_name, email,
hire_date, job_id, salary
FROM employees
WHERE department_id = 50 WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',
TO_DATE('07-JUN-99', 'DD-MON-RR'),
'ST_CLERK', 5000);
```

In the example shown, the `WITH CHECK OPTION` keyword is used. The subquery identifies rows that are in department 50, but the department ID is not in the `SELECT` list, and a value is not provided for it in the `VALUES` list. Inserting this row would result in a department ID of null, which is not in the

subquery.

```
WHERE StandardCost > 1000.00;
```


Exercises

Using the EMP and DEPT table, apply the following queries:

1. Create a new table EMPLOYEE replica of EMP table with no records in it.
Create a new table DEPART replica of DEPT table with all the records in it.
Create a primary foreignkey relationship between EMPLOYEE and DEPART tables.
2. Insert following records into EMPLOYEE using following values:
EMPNO = 101, ENAME = WASEEEM, SAL = 5000, JOB = CLERK and DEPTNO = 20.
EMPNO = 102, ENAME = SAJID, SAL = 15000, JOB = ANALYST and DEPTNO = 10.
3. Change the record inserted in question 2 from ENAME 'WASEEM' to 'KHALID' and SAL 5000 to 10000.
4. Remove all the records from EMPLOYEE table having job = 'ANALYST'.
5. Try to update the table DEPART and change the value of DEPTNO from 10 to 30. Find the error and give the reason for the error.
6. Insert a new record into EMPLOYEE using following values:
EMPNO = 103, ENAME = SALEEM, SAL = 60000, JOB = ANALYST and DEPTNO = 60.
Find the error and give the reason for the error.