K.J. Somaiya College of Engineering, Mumbai-77

(A Constituent College of Somaiya Vidyavihar University)

Batch: D2 **Roll No.: 16010122323**

Experiment / assignment / tutorial No. 4

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: : DML – select, insert, update and delete

1.Group by, having clause, aggregate functions, Set Operations
2.Nested queries : AND,OR,NOT, IN, NOT IN, Exists, Not Exists, Between, Like, Alias, ANY,ALL,DISTINCT
3. Update
4. Delete

Objective: To perform various DML Operations and executing nested queries with various clauses.

Expected Outcome of Experiment:

CO 3: Use SQL for Relational database creation, maintenance and query processing

Books/ Journals/ Websites referred:

1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press
2. www.db-book.com
3. Korth, Slberchatz, Sudarshan : “Database Systems Concept”, 5th Edition , McGraw Hill
4. Elmasri and Navathe,”Fundamentals of database Systems”, 4th Edition PEARSON Education

Resources used: Postgres

Theory: Select: The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows –

SELECT column1, column2, columnN FROM table_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

SELECT * FROM table_name;

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

Insert: The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)

VALUES (value1, value2, value3,...valueN);

Example

The following statements would create record in the CUSTOMERS table.

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00);

Update: The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax:

The basic syntax of the UPDATE query with a WHERE clause is as follows –

UPDATE table_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

You can combine N number of conditions using the AND or the OR operators.

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;

Delete: The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

The basic syntax of the DELETE query with the WHERE clause is as follows −

DELETE FROM table_name

WHERE [condition];

The following code has a query, which will DELETE a customer, whose ID is 6.

SQL> DELETE FROM CUSTOMERS
WHERE ID = 6;

Clauses and Operators

1. **Group by clause:** These are circumstances where we would like to apply the aggregate functions to a single set of tuples but also to a group of sets of tuples we would like to specify this wish in SQL using the group by clause. The attributes or attributes given by the group by clause are used to form groups. Tuples with the same value on all attributes in the group by clause placed in one group.

Example:.

Select<attribute_name,avg(<attribute_name>)as
<new_attribute_name>l From <table_name>
Group by <attribute_name>

Example: select designation, sum(salary) as total_salary from employee group by Designation;

2. Having clause: A having clause is like a where clause but only applies only to groups as a whole whereas the where clause applies to the individual rows. A query can contain both where clause and a having clause. In that case

a. The where clause is applied first to the individual rows in the tables or table structures objects in the diagram pane. Only the rows that meet the conditions in the where clause are grouped.

b. The having clause is then applied to the rows in the result set that are produced by grouping. Only the groups that meet the having conditions appear in the query output.

Example:

```
select dept_no from EMPLOYEE group_by dept_no
having avg (salary) >=all (select avg (salary)
from EMPLOYEE group by dept_no);
```

3. Aggregate functions: Aggregate functions such as SUM, AVG, count, count (*), MAX and MIN generate summary values in query result sets. An aggregate functions (with the exception of count (*) processes all the selected values in a single column to produce a single result value

Example:
```
select dept_no,count (*)
from EMPLOYEE group by dept_no;
```

Example:
```
select max (salary)as maximum from EMPLOYEE;
```

Example:
```
select sum (salary) as total_salary from EMPLOYEE;
```

Example:
```
Select min (salary) as minsal from EMPLOYEE;
```

4. Exists and Not Exists: Subqueries introduced with exists and not queries can be used for two set theory operations: Intersection and Difference. The intersection of two sets contains all elements that belong to both of the original sets. The difference contains elements that belong to only first of the two sets.

Example:

```
Select *from DEPARTMENT
where exists(select * from PROJECT
        where DEPARTMENT.dept_no = PROJECT.dept_no) ;
```

5. **IN and Not In**: SQL allows testing tuples for membership in a relation. The “in‟ connective tests for set membership where the set is a collection of values produced by

select clause. The “not in“ connective tests for the absence of set membership. The in and not in connectives can also be used on enumerated sets.

Example:

1. Select fname, mname, lname from employee where designation In (“ceo“,“manager“,“hod“,“assistant“)

2. Select fullname from department where relationship not in(“brother”);

6. Between: The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive. Begin and end values are included.

Syntax:

SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;

Example:

SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;

7. LIKE: The LIKE **operator** is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

Syntax: SELECT *column1, column2, ...*
FROM *table_name*
WHERE *columnN* LIKE *pattern*

Examples:

1. selects all customers with a CustomerName starting with "a":

SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';

2. selects all customers with a CustomerName that have "r" in the second position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

8. Alias: The use of table aliases is to rename a table in a specific SQL statement. The renaming is a temporary change and the actual table name does not change in the database. The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

The basic syntax of a **table** alias is as follows.

```
SELECT column1, column2....
FROM table_name AS alias_name
WHERE [condition];
```

The basic syntax of a **column** alias is as follows.

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

Example:

```
SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
FROM CUSTOMERS AS C, ORDERS AS O
WHERE  C.ID = O.CUSTOMER_ID;
```

9. Distinct: The SELECT DISTINCT statement is used to return only distinct (different) values.

Syntax: SELECT DISTINCT *column1*, *column2*, ...
FROM *table_name*;

Example: SELECT DISTINCT Country FROM Customers;

10. Set Operations: 4 different types of SET operations, along with example:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

Query: SELECT * FROM First

UNION

SELECT * FROM Second;

UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.

Query: SELECT * FROM First

UNION ALL

SELECT * FROM Second;

INTERSECT

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

Query: SELECT * FROM First

INTERSECT

SELECT * FROM Second;

MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

Query: SELECT * FROM First

MINUS

SELECT * FROM Second;

11. ANY and ALL: The ANY and ALL operators are used with a WHERE or HAVING clause. The ANY operator returns true if any of the subquery values meet the condition. The ALL operator returns true if all of the subquery values meet the condition.

ANY

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name operator* ANY
(SELECT *column_name* FROM *table_name* WHERE *condition*);

Example: The following SQL statement returns TRUE and lists the productnames if it finds ANY records in the OrderDetails table that quantity = 10:

SELECT ProductName
FROM Products
WHERE ProductID
= ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

ALL

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name operator* ALL
(SELECT *column_name* FROM *table_name* WHERE *condition*);

Example: The following SQL statement returns TRUE and lists the product names if ALL the records in the OrderDetails table has quantity = 10:

SELECT ProductName
FROM Products
WHERE ProductID
= ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

JOIN OPERATIONS:

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Join operations take two relations and return as a result another relation.

These additional operations are typically used as subquery expressions in the **from** clause

Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join.

Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated

loan **join** *borrower* **on**

loan.loan_number = borrower.loan_number

CREATE [TEMP | TEMPORARY] VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE [condition];

Ex

CREATE VIEW COMPANY_VIEW AS

SELECT ID, NAME, AGE

FROM COMPANY;

Dropping Views

Syntax: DROP VIEW view_name;

Implementation details:

CREATE TABLE DONOR (

DONOR_ID INT NOT NULL,

DONOR_NAME VARCHAR(50) NOT NULL,

PASSKEY VARCHAR(16) NOT NULL,

```
    DONATION_HISTORY INTEGER,
    PHONE_NUMBER VARCHAR(10),
    EMAIL VARCHAR(30),
    PRIMARY KEY (DONOR_ID)
);

INSERT INTO DONOR VALUES(01, 'JIYA', 'jiya123', 2000, '8282828282', 'jiya@gmail.com');

INSERT INTO DONOR VALUES (02, 'Rahul', 'rahul456', 1500, '7878787878', 'rahul@gmail.com');

INSERT INTO DONOR VALUES (03, 'Aisha', 'aisha789', 3000, '9090909091', 'aisha@gmail.com');


CREATE TABLE NGO (
    NGO_ID INT NOT NULL,
    NGO_NAME VARCHAR(50) NOT NULL,
    ADDRESS VARCHAR(100) NOT NULL,
    DONATION_RECIEVED INTEGER,
    PHONE_NUMBER VARCHAR(10),
    EMAIL VARCHAR(30),
    DONOR_ID INT NOT NULL,
    PRIMARY KEY (NGO_ID),
    FOREIGN KEY (DONOR_ID) REFERENCES DONOR
);
```

```
INSERT INTO NGO VALUES(01, 'ROBIN HOOD ARMY', 'MUMBAI', 50000, 9090909090, 'robinhood@gmail.com', 01);

INSERT INTO NGO VALUES (02, 'Greenpeace', 'New Delhi', 200000, '8080808080', 'greenpeace@gmail.com', 02);

INSERT INTO NGO VALUES (03, 'CARE India', 'Kolkata', 100000, '7070707070', 'careindia@gmail.com', 03);


CREATE TABLE VOLUNTEER (
        VOLUNTEER_ID INT NOT NULL,
        VOLUNTEER_NAME VARCHAR(50) NOT NULL,
        GENDER CHAR(1),
        ASSIGNED_TASK VARCHAR(30),
        PHONE_NUMBER VARCHAR(10),
        EMAIL VARCHAR(30),
        NGO_ID INT,
        PRIMARY KEY (VOLUNTEER_ID),
        FOREIGN KEY (NGO_ID) REFERENCES NGO
);

INSERT INTO VOLUNTEER VALUES(01, 'Kartik', 'M', 'Food drive', '7282828282', 'kartik@gmail.com', 01);

INSERT INTO VOLUNTEER VALUES (02, 'Sara', 'F', 'Tree Plantation', '8888888888', 'sara@gmail.com', 02);

INSERT INTO VOLUNTEER VALUES (03, 'Raj', 'M', 'Education Program', '9999999999', 'raj@gmail.com', 03);
```

```
CREATE TABLE TASK (
        TASK_ID INT NOT NULL,
        STATUS VARCHAR(20) CHECK (STATUS IN ('Assigned', 'Not assigned')),
        DESCRIPTION VARCHAR(100),
        VOLUNTEER_ID INT NOT NULL,
        FOREIGN KEY (VOLUNTEER_ID) REFERENCES VOLUNTEER
);
INSERT INTO TASK VALUES(01, 'Assigned', 'Food distribution to the homeless in mumbai', 01)
INSERT INTO TASK VALUES(02, 'Assigned', 'Provide education resources to unprivileged children', 02);
INSERT INTO TASK VALUES(03, 'Assigned', 'Tree plantation', 03);

CREATE TABLE PROJECTS (
        PROJECT_ID INT NOT NULL,
        PROJECT_NAME VARCHAR(30) NOT NULL,
        PROJECT_LOCATION VARCHAR(100) NOT NULL,
        EXPENDITURE INT,
        MOTIVE VARCHAR(500),
        TASK_ID INT NOT NULL,
        PRIMARY KEY (PROJECT_ID),
        FOREIGN KEY (TASK_ID) REFERENCES TASK
);
```

```
INSERT INTO PROJECTS VALUES(1, 'Food Distribution', 'Mumbai', 5000, 'Food distribution to the homeless in Mumbai', 01);

INSERT INTO PROJECTS VALUES(2, 'Education for All', 'Delhi', 8000, 'Provide education resources to underprivileged children', 02);

INSERT INTO PROJECTS VALUES(3, 'Tree Plantation Drive', 'Ahmedabad', 3000, 'Increase green cover in the community', 03);


CREATE TABLE DONATES_TO (
   DONOR_ID INT NOT NULL,
   NGO_ID INT NOT NULL,
   MONEY DECIMAL(10, 2) NOT NULL,
   PRIMARY KEY (DONOR_ID, NGO_ID),
   FOREIGN KEY (DONOR_ID) REFERENCES DONOR(DONOR_ID),
   FOREIGN KEY (NGO_ID) REFERENCES NGO(NGO_ID)
);


CREATE TABLE ONSITE_VOLUNTEER (
   VOLUNTEER_ID INT NOT NULL,
   ONSITE_TASK VARCHAR(30),
   PRIMARY KEY (VOLUNTEER_ID),
   FOREIGN KEY (VOLUNTEER_ID) REFERENCES VOLUNTEER
);

INSERT INTO ONSITE_VOLUNTEER(VOLUNTEER_ID, ONSITE_TASK) VALUES
```

```
(1, 'Event Setup'),
(2, 'Cleanup Crew'),
(3, 'Registration Assistance');

CREATE TABLE REMOTE_VOLUNTEER (
   VOLUNTEER_ID INT NOT NULL,
   REMOTE_TASK VARCHAR(30),
   PRIMARY KEY (VOLUNTEER_ID),
   FOREIGN KEY (VOLUNTEER_ID) REFERENCES VOLUNTEER
);

ALTER TABLE TASK
ADD CONSTRAINT pk_task_id
PRIMARY KEY (TASK_ID);

ALTER TABLE VOLUNTEER
RENAME COLUMN ASSIGNED_TASK TO TASK_DESCRIPTION;

ALTER TABLE PROJECTS
ALTER COLUMN EXPENDITURE TYPE DECIMAL(12, 2);

--DROP TABLE NGO;
--DROP TABLE TASK;
```

```
-- ALTER TABLE DONOR

-- DROP COLUMN EMAIL;


SELECT DONOR_NAME, SUM(DONATION_HISTORY) AS TOTAL_DONATION

FROM DONOR

GROUP BY DONOR_NAME;


SELECT NGO_NAME, COUNT(*) AS VOLUNTEER_COUNT

FROM NGO n

JOIN VOLUNTEER v ON n.NGO_ID = v.NGO_ID

GROUP BY NGO_NAME;


SELECT PROJECT_LOCATION, AVG(EXPENDITURE) AS AVG_EXPENDITURE

FROM PROJECTS

GROUP BY PROJECT_LOCATION;


UPDATE DONOR

SET DONATION_HISTORY = DONATION_HISTORY + 48000

WHERE DONOR_ID = 01;


UPDATE DONOR

SET DONATION_HISTORY = DONATION_HISTORY + 198500
```

```
WHERE DONOR_ID = 02;


UPDATE DONOR

SET DONATION_HISTORY = DONATION_HISTORY + 97000

WHERE DONOR_ID = 03;


SELECT VOLUNTEER_NAME

FROM VOLUNTEER

WHERE NGO_ID IN (SELECT NGO_ID FROM NGO WHERE ADDRESS IN
('Delhi', 'Kolkata'));


SELECT * FROM DONOR;

SELECT * FROM NGO;

SELECT * FROM VOLUNTEER;

SELECT * FROM TASK;

SELECT * FROM PROJECTS;

SELECT * FROM DONATES_TO;

SELECT * FROM ONSITE_VOLUNTEER;

SELECT * FROM REMOTE_VOLUNTEER;
```

Screenshots:

	donor_name character varying (50)	total_donation bigint
1	Rahul	200000
2	JIYA	50000
3	Aisha	100000

	ngo_name character varying (50)	volunteer_count bigint
1	Greenpeace	1
2	ROBIN HOOD ARMY	1
3	CARE India	1

	project_location character varying (100)	avg_expenditure numeric
1	Ahmedabad	3000.0000000000000000
2	Delhi	8000.0000000000000000
3	Mumbai	5000.0000000000000000

	volunteer_name character varying (50)
1	Raj

Conclusion:

Performed various DML operations on our projects that were stated in the experiment

Post lab queries:

1. **W.r.t your table give SQL query to insert more than one record at a time**

```
INSERT INTO ONSITE_VOLUNTEER(VOLUNTEER_ID, ONSITE_TASK) VALUES
(1, 'Event Setup'),
(2, 'Cleanup Crew'),
(3, 'Registration Assistance');
```

2. **What is the difference between Join and full outer join operation**

→

A JOIN operation combines rows from two or more tables based on a related column. There are different types of joins, including INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN.
The result of a JOIN includes only the rows that have matching values in the specified columns from both tables.

A FULL OUTER JOIN combines rows from two tables, including unmatched rows from both tables.
If there is no match between the tables, the result will still include rows from both tables, with NULL values for columns from the table that doesn't have a match.
It essentially combines the results of both LEFT OUTER JOIN and RIGHT OUTER JOIN.