

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. Join ,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, Silberchatz, Sudarshan : “Database Systems Concept”, 5th Edition , McGraw Hill
4. Elmasri and Navathe, “Fundamentals of database Systems”, 4th Edition PEARSON Education.

Department of Computer Engineering**RDBMS –Sem-IV- Jan –April 2025**

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>| 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

- 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.
- 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 (*)

Department of Computer Engineering

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)

Department of Computer Engineering

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....
```

Department of Computer Engineering

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 returns 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:

<i>Join types</i>	<i>Join Conditions</i>
inner join left outer join right outer join full outer join	natural on <predicate> using (A_1, A_1, \dots, 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

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

SELECT ID, NAME, AGE

FROM COMPANY;

Dropping Views

Syntax: DROP VIEW view_name;

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Implementation details

```
1 SELECT d.dname, COUNT(e.essn) AS num_employees
2 FROM dept d
3 JOIN emp e ON d.dnumber = e.dno
4 GROUP BY d.dname;
```

Data Output Messages Notifications

	dname character varying (10)	num_employees bigint
1	Marketing	1
2	IT	1
3	Billing	1
4	PR Team	2

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1 select *
2 from emp
3 where dno=1
```

Data Output Messages Notifications

	ename character varying (30)	essn [PK] character (9)	bdate date	dno integer	superssn character (9)
1	Anirudh Navuduri	0	2008-11-26	1	2

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1 SELECT essn, ename AS name, dno
2 FROM emp
3 GROUP BY essn,ename,dno;
```

Data Output Messages Notifications

	essn [PK] character (9)	name character varying (30)	dno integer
1	2	Aditey Kshirsagar	3
2	0	Anirudh Navuduri	1
3	4	Aditi Sharma	4
4	3	Vedika Padliya	4
5	1	Aaditya Singh	2

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query

Query History

1

▼

SELECT COUNT(*) AS TotalEmp

2

FROM emp

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	totalemp	bigint
1		5

Query Query History

```
1 SELECT PNUMBER, SUM(HOURS) AS Total_Hours
2 FROM WORKS_ON
3 GROUP BY PNUMBER
4 HAVING SUM(HOURS) > 5;
```

Data Output Messages Notifications

	pnumber integer	total_hours bigint
1	223	25
2	221	10
3	222	40

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query

Query History

1

2

▼

SELECT AVG(hours) AS AVERAGEHOURS

FROM works_on

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	averagehours numeric	🔒
1	25.0000000000000000	

Query Query History

```
1  SELECT MAX(hours) AS MAXIMUMHOURS
2  FROM works_on
3
```

Data Output Messages Notifications



	maximumhours integer	
1		40

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1  SELECT E.ENAME, E.ESSN
2  FROM EMP E
3  WHERE EXISTS (
4      SELECT 1
5      FROM DEPENDENT D
6      WHERE D.ESSN = E.ESSN
7  );
```

Data Output Messages Notifications

	ename character varying (30)	essn [PK] character (9)
1	Anirudh Navuduri	0
2	Aaditya Singh	1
3	Aditey Kshirsagar	2
4	Vedika Padliya	3
5	Aditi Sharma	4










Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1  SELECT e.ename,e.essn FROM emp e
2  WHERE NOT EXISTS (
3      SELECT 1
4      FROM DEPENDENT d
5      WHERE d.essn=e.essn
6  );
```

Data Output Messages Notifications

									SQL
ename					essn				
character varying (30)					[PK] character (9)				










Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1  SELECT P.PNAME, P.PNUMBER, P.PLOCATION, P.PNUMBER
2  FROM PROJECT P
3  WHERE P.PNUMBER NOT IN (
4      SELECT W.PNUMBER
5      FROM WORKS_ON W
6  );
```

Data Output Messages Notifications

									SQL
pname	pnumber	plocation	pnumber						
character varying (10)	[PK] integer	character varying (10)	integer						

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1 SELECT E.ENAME, E.ESSN
2 FROM EMP E
3 WHERE E.ESSN NOT IN (
4     SELECT D.MGRSSN
5     FROM DEPT D
6 );
```

Data Output Messages Notifications

	ename character varying (30)	essn [PK] character (9)
1	Vedika Padliya	3

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query

Query History

1

2

3

SELECT

essn,

ename,

dno

FROM

emp e

WHERE

CAST(essn AS INT)

BETWEEN

95

AND

105;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

essn

[PK] character (9)

✎

ename

character varying (30)

✎

dno










integer

✎

Query Query History

```
1  SELECT essn, ename, dno
2  FROM emp e
3  WHERE ename LIKE '%ay'
```

Data Output Messages Notifications

									SQL
essn	ename	dno							
[PK] character (9)	character varying (30)	integer							

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

[Query](#) [Query History](#)

```
1  SELECT essn, ename, dno
2  FROM emp e
3  WHERE ename NOT LIKE '%ay'
```

[Data Output](#) [Messages](#) [Notifications](#)

	essn [PK] character (9)	ename character varying (30)	dno integer
1	0	Anirudh Navuduri	1
2	1	Aaditya Singh	2
3	2	Aditey Kshirsagar	3
4	3	Vedika Padliya	4
5	4	Aditi Sharma	4

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1 SELECT P.PNAME, P.PNUMBER, P.PLOCATION, P.PNUMBER
2 FROM PROJECT AS P;
```

Data Output Messages Notifications

	pname character varying (10)	pnumber [PK] integer	plocation character varying (10)	pnumber integer
1	Zoniq	222	Kandivali	222
2	Sponsorway	223	Malad	223
3	Sponsorway	221	Malad	221

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1  SELECT DISTINCT PLOCATION
2  FROM PROJECT;
```

Data Output Messages Notifications

	plocation character varying (10)
1	Malad
2	Kandivali

Department of Computer Engineering


RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1 SELECT PLOCATION FROM PROJECT
2 UNION
3 SELECT DLOCATION FROM DEPT_LOC;
```

Data Output Messages Notifications



	plocation character varying (10) 
1	Malad
2	Kandivali
3	Colaba
4	Churchgate

Department of Computer Engineering


RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1  SELECT PLOCATION FROM PROJECT
2  UNION ALL
3  SELECT DLOCATION FROM DEPT_LOC;
```

Data Output Messages Notifications



	plocation character varying (10) 
1	Kandivali
2	Malad
3	Malad
4	Churchgate
5	Churchgate
6	Colaba
7	Colaba

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

[Query](#) [Query History](#)

1

2

3

▼


SELECT PLOCATION FROM PROJECT

INTERSECT


SELECT DLOCATION FROM DEPT_LOC;

[Data Output](#) [Messages](#) [Notifications](#)


≡+





▼




▼










SQL

plocation

character varying (10) 


Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query Query History

```
1  SELECT PLOCATION FROM PROJECT
2  EXCEPT
3  SELECT DLOCATION FROM DEPT_LOC;
```

Data Output Messages Notifications

	plocation character varying (10) 
1	Malad
2	Kandivali

Query

Query History

1

2

3

SELECT

ENAME,

ESSN

FROM

EMP

WHERE

ESSN = ANY

(SELECT

ESSN

FROM

WORKS_ON

WHERE

HOURS >

20);

Data Output

Messages

Notifications

≡+

▼

▼

SQL

	ename character varying (30)	essn [PK] character (9)
1	Aaditya Singh	1
2	Aditey Kshirsagar	2

Query

Query History

1

▼

SELECT

ename

2

FROM

emp

3

WHERE

essn

>

ALL

(SELECT

essn

FROM

emp

WHERE

dno

=

1

);

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	ename character varying (30) 🔒
1	Aaditya Singh
2	Aditey Kshirsagar
3	Vedika Padliya
4	Aditi Sharma

Query

Query History

1

2

3

SELECT

E.ENAME,

W.PNUMBER,

W.HOURS

FROM

EMP E

INNER JOIN

WORKS_ON W ON

E.ESSN = W.ESSN;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	ename character varying (30) 🔒	pnumber integer 🔒	hours integer 🔒
1	Aditey Kshirsagar	222	[null]
2	Aditey Kshirsagar	222	40
3	Vedika Padliya	221	10
4	Aaditya Singh	223	25

[Query](#) [Query History](#)

```
1  SELECT E.ENAME, W.PNUMBER, W.HOURS
2  FROM EMP E
3  LEFT JOIN WORKS_ON W ON E.ESSN = W.ESSN;
```

[Data Output](#) [Messages](#) [Notifications](#)

	ename character varying (30)	pnumber integer	hours integer
1	Aditey Kshirsagar	222	[null]
2	Aditey Kshirsagar	222	40
3	Vedika Padliya	221	10
4	Aaditya Singh	223	25
5	Aditi Sharma	[null]	[null]
6	Anirudh Navuduri	[null]	[null]

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Query

Query History

1

2

3

SELECT

E.ENAME,

W.PNUMBER,

W.HOURS

FROM

EMP E

RIGHT JOIN

WORKS_ON W

ON

E.ESSN = W.ESSN;

Data Output

Messages

Notifications

≡+

▼

▼

SQL

	ename character varying (30)	pnumber integer	hours integer
1	Aditey Kshirsagar	222	[null]
2	Aditey Kshirsagar	222	40
3	Vedika Padliya	221	10
4	Aaditya Singh	223	25

Query Query History

```
1  ✓ SELECT E.ENAME, W.PNUMBER, W.HOURS
2  FROM EMP E
3  FULL JOIN WORKS_ON W ON E.ESSN = W.ESSN;
```

Data Output Messages Notifications

	ename character varying (30)	pnumber integer	hours integer
1	Aditey Kshirsagar	222	[null]
2	Aditey Kshirsagar	222	40
3	Vedika Padliya	221	10
4	Aaditya Singh	223	25
5	Aditi Sharma	[null]	[null]
6	Anirudh Navuduri	[null]	[null]

Department of Computer Engineering

RDBMS –Sem-IV- Jan –April 2025

Conclusion:

I gained hands-on experience with various DML operations such as SELECT, INSERT, UPDATE and DELETE which allowed me to manipulate data within relational database tables effectively. I also worked with nested queries using conditions like AND, OR, IN, NOT IN, EXISTS, BETWEEN, LIKE, and DISTINCT, helping me refine query results efficiently. Additionally, I explored aggregate functions and set operations to perform complex data retrieval tasks. I also learned about JOIN operations, including INNER JOIN and FULL OUTER JOIN, to fetch related data from multiple tables. The use of GROUP BY and HAVING clauses further helped me group and filter aggregated results. Overall, this experiment significantly enhanced my understanding of SQL query processing, data manipulation, and optimization techniques, providing me with a strong foundation for working with relational databases.

Post lab queries:

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

```
INSERT INTO EMP (ID, NAME, AGE, ADDRESS, SALARY) VALUES (1, 'Aditya', 32, 'Ahmedabad', 2000.00),  
INSERT INTO EMP (ID, NAME, AGE, ADDRESS, SALARY) VALUES (2, 'Aditey', 28, 'Mumbai', 2500.00),  
INSERT INTO EMP (ID, NAME, AGE, ADDRESS, SALARY) VALUES (3, 'Ambuj', 35, 'Pune', 3000.00);
```

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

JOIN (INNER JOIN): Returns only the matching records between two tables based on the specified condition.

FULL OUTER JOIN: Returns all records from both tables, filling non-matching entries with NULL values.