K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment / assignment / tutorial No. 4

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

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 2: Convert entity-relationship diagrams into relational tables, populate a relational database and formulate SQL queries on the data Use SQL for creation and query the database.

.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 productnames 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);

Implementation details

- Simple question based on your application, queries and screen shots for each type:

Queries:

```
CAST_MOVIE:
SELECT MAX(no_of_movies) from cast_movie
select * from cast_movie

insert into cast_movie
values(2323, 'Johnny Depp', 52, 30)

insert into cast_movie
values(3211, 'Rowan Atkinson', 52, 24)

insert into cast_movie
values(4123, 'Shah Rukh Khan', 52, 45)

delete from cast_movie
where cast_id=4123

insert into cast_movie
values(4545, 'Robert Downey Jr.', 45, 15)

insert into cast_movie
values(2123, 'Emma Stone', 26, 10)

update cast_movie
set no_of_movies=20
where cast_id=3211

select * from cast_movie where age>all(select age from
cast_movie where age<40)

select * from cast_movie where age between 40 and 55

THEATRE:
select * from theatre

insert into theatre
values(65, 'INOX', 'Dadar');

insert into theatre
values(71, 'Carnival','Kanjurmarg')
```

```
insert into theatre
values(404, 'PVR', 'Ulhasnagar')

select * from cast_movie where name like '%Khan'

CUSTOMER:
insert into customer (email_id, name, age)
values('g.bhagwanani@somaiya.edu','Gaurav Bhagwanani',19)
insert into customer (email_id, name, age)
values('arghyadeep.d@somaiya.edu','Arghyadeep Das’,19)
insert into customer (email_id, name, age)
values('kaustubh.damania@somaiya.edu','Kaustubh
Damania',19)
insert into customer (email_id, name, age)
values('gaurang.a@somaiya.edu','Gaurang Athavale',19)
select name from customer intersect select name from
cast_movie
select name from customer except select name from
cast_movie

select * from customer
select movie_name from movie union all select location
from theatre
select name from customer intersect select name from
cast_movie
select name from customer except select name from
cast_movie
REVIEW:
select * from review

insert into review
values(2323, 'Very funny movie, loved it.','Acting was
superb, dialogues were good.')

insert into review
values(2323, 'Absolutely fantabulous.','Rowan Atkinson
played an amazing role as the main character.')

insert into review
values(3232, 'Iron Man is just too cool!','The storyline
was very well done. Cinematography was also amazing.')

insert into review
values(1544, 'IT was very scary to watch.','Pennywise did
an excellent job.')
```

```
insert into review
values(1234, 'Worth a watch!','Portrayal of Zuckerberg was
perfect.')

SELECT DISTINCT * from review

select * from review where exists(select movie_id from
movie where movie.movie_id=review.mov_id)


MOVIE:
select * from movie

insert into movie
values(2323, 'Johnny English', '2012-05-02', 30, 'Comedy')

insert into movie
values(3232, 'Avengers', '2012-06-02', 30, 'Action')

insert into movie
values(1234, 'The Social Network', '2009-02-12', 30,
'Biography')

insert into movie
values(1544, 'IT', '2018-12-12', 5.0, 'Horror')

insert into movie
values(1155, 'Death Race', '2008-12-12', 4.5, 'Thriller')

update movie
set rating=4
where movie_id=2323

update movie
set rating=4.7
where movie_id=3232

update movie
set rating=3.7
where movie_id=1234

SELECT count(movie_name) from movie
where genre='Comedy'
```

```
SELECT avg(rating) from movie
where genre='Biography'

select genre from movie group by genre order by genre

select * from movie where movie_id NOT IN (3281,4011)

select genre from movie group by genre having
count(movie_id)>1 order by genre
select movie_name from movie union all select location
from theatre
```

Screenshots:

Previous queries: select movie_name from movie union all select location fror | Delete | Delete All

```
select * from cast_movie where age>all(select age from cast_movie where age<40)
```

Output pane

Data Output | Explain | Messages | History

	cast_id integer	name character varying(30)	age integer	no_of_movies integer
1	1	Shah Rukh Khan	53	150
2	4	Aamir Khan	52	120
3	2	Salmaan Khan	53	120
4	8	Dwayne The Rock Johnson	45	20

Previous queries: select movie_name from movie union all select location fror | Delete | Delete All

```
select * from cast_movie where age between 20 and 45
```

Output pane

Data Output | Explain | Messages | History

	cast_id integer	name character varying(30)	age integer	no_of_movies integer
1	5	Emma Watson	25	10
2	6	Deepika Padukone	32	50
3	7	Anushka Sharma	28	30
4	8	Dwayne The Rock Johnson	45	20

```
select * from cast_movie where name like '%Khan'
```

Output pane

Data Output | Explain | Messages | History

	cast_id integer	name character varying(30)	age integer	no_of_movies integer
1	1	Shah Rukh Khan	53	150
2	4	Aamir Khan	52	120
3	2	Salmaan Khan	53	120

```
select name from customer intersect select name from cast_movie
```

Output pane

Data Output | Explain | Messages | History

	name character varying(30)
1	Aamir Khan

```
select name from customer except select name from cast_movie
```

Output pane

Data Output | Explain | Messages | History

	name character varying(30)
1	Gaurang Athavale
2	Gaurav Bhagwanani
3	Kaustubh Damania
4	Sushant Gharal
5	Arghyadeep Das

```
select movie_name from movie union all select location from theatre
```

Output pane

Data Output | Explain | Messages | History

	movie_name character varying
1	2012
2	Mr. Bean-The Painting
3	Welcome Back
4	Singham Returns
5	The Man Who Knew Infinity
6	MSD-An Untold Story
7	Annabelle
8	Stree
9	New York
10	Dadar
11	Kharghar
12	Vashi
13	Kanjurmarg
14	Ulhasnagar
15	New York

```
SELECT DISTINCT * from review
```

Output pane

Data Output | Explain | Messages | History

	mov_id integer	cust_review character varying(200)	analyst_review character varying(200)
1	9882	Very scary	Good job. Expecting a sequel to it.
2	3281	John did a fantastic role in the movie.	Justice is not done to the prequel.
3	9882	I almost freaked out.	Perfect film to watch at 1 AM.
4	1892	Amazing thriller experience. Loved it.	Excellent acting. Had me at the edge of my seat.
5	1892	Absolutely loved it.	Excellent acting. The storyline is amazing
6	3281	John did a fantastic role in the movie.	Excellent acting. The storyline is amazing

```
select * from review where exists(select movie_id from movie where movie.movie_id=review.mov_id)
```

Output pane

Data Output | Explain | Messages | History

	mov_id integer	cust_review character varying(200)	analyst_review character varying(200)
1	1892	Amazing thriller experience. Loved it.	Excellent acting. Had me at the edge of my seat.
2	1892	Absolutely loved it.	Excellent acting. The storyline is amazing
3	3281	John did a fantastic role in the movie.	Justice is not done to the prequel.
4	3281	John did a fantastic role in the movie.	Justice is not done to the prequel.
5	3281	John did a fantastic role in the movie.	Excellent acting. The storyline is amazing
6	9882	I almost freaked out.	Perfect film to watch at 1 AM.
7	9882	Very scary	Good job. Expecting a sequel to it.

```
SELECT avg(rating) from movie
where genre='Biography'
```

Output pane

Data Output | Explain | Messages | Hist

	avg double precision
1	4.8

```
SELECT count(movie_name) from movie
where genre='Comedy'
```

Output pane

Data Output | Explain | Messages | History

	genre character varying(15)
1	Biography
2	Comedy
3	Horror

```
select * from movie where movie_id NOT IN (3281,4011)
```

Output pane

Data Output | Explain | Messages | History

	movie_id integer	movie_name character varying(60)	release_date date	rating double precision	genre character varying(15)
1	1892	2012	2008-04-02	3.8	Thriller
2	2091	Mr. Bean-The Painting	2012-03-08	4.1	Comedy
3	5017	The Man Who Knew Infinity	2017-09-06	4.8	Biography
4	5482	MSD-An Untold Story	2017-10-20	4.8	Biography
5	9882	Annabelle	2016-10-20	4.2	Horror
6	2122	Stree	2018-11-20	3.2	Horror
7	1000	New York	2009-09-10	3.5	Adventure

```
select genre from movie group by genre having count(movie_id)>1 order by genre
```

Output pane

Data Output | Explain | Messages | History

	genre character varying(15)
1	Biography
2	Comedy
3	Horror

Conclusion: The database was successfully updated with data and updated with new data and various DML commands were performed.