Batch: B2 Roll No.: 110 and 109

Experiment / assignment / tutorial No. 4

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 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:

select * from items

insert into items(itemname,effect,itemno) values ('Potion','heal',1);

/*insert into items(itemname,effect) values ('pokeball','catch pokemon');*/ /*must throw error as primary key cannot be null*/

/*insert into items(itemname,effect,itemno) values ('Pokeball','catch pokemon',1);*/
/*must throw error as primary key cannot be duplicate*/

insert into items(itemname,effect,itemno) values ('Pokeball','catch pokemon',2);
update items set itemname='Great Ball' where (itemno=2);
select itemname,effect from items;

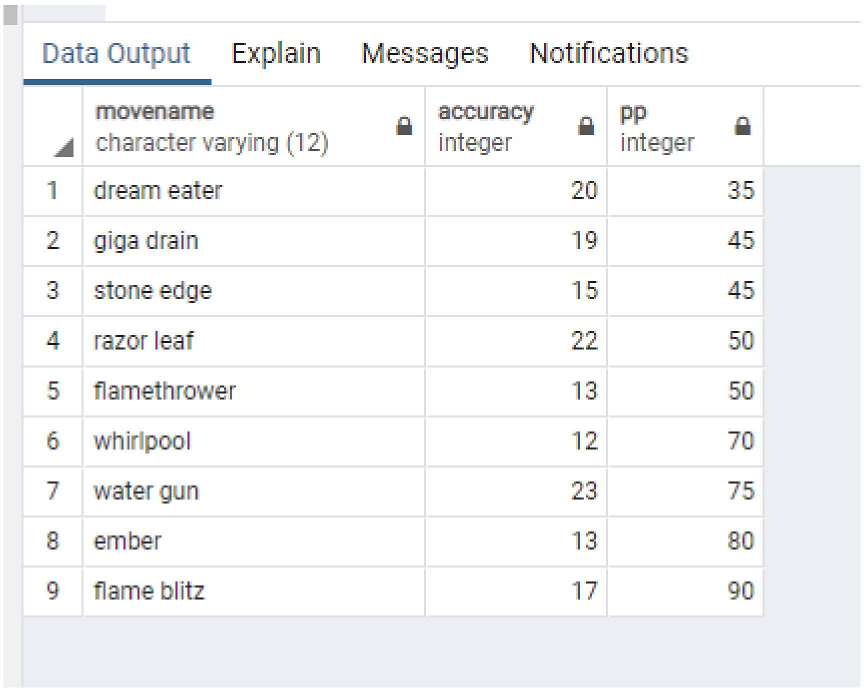
insert into items(itemname,effect,itemno) values ('Poke Ball','catch pokemon',3);
insert into items(itemname,effect,itemno) values ('Master Ball','catch pokemon',4);
select itemname,effect from items;

/*lets update all balls*/
update items set effect='catch wild pokemon' where (itemname like '% Ball');
select itemname,effect from items;

/*lets update for a few balls*/

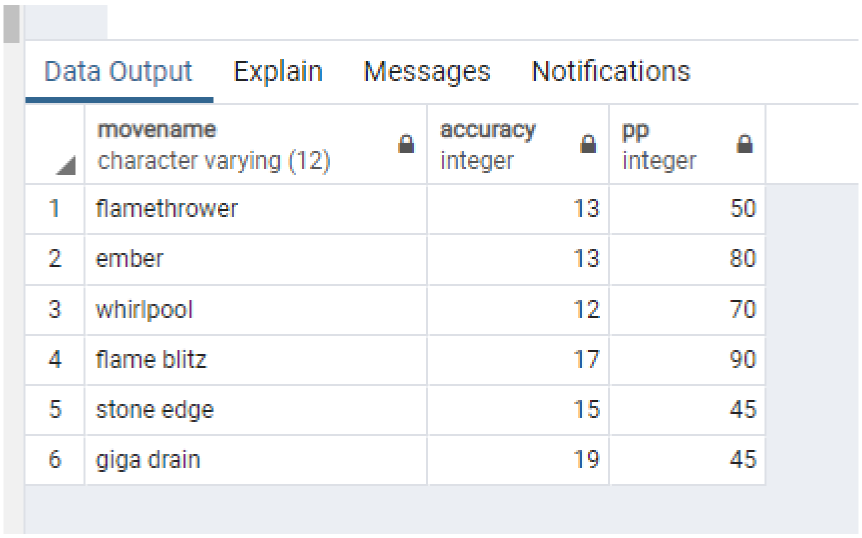
```
update items set effect='catch strong wild pokemon' where (itemname in ('Great
Ball','Master Ball'));

select itemname,effect from items;

insert into items(itemname,effect,itemno) values ('Citrus berry','heal',5);
/*lets see all distinct effects*/
select distinct effect from items

/*lets add some types*/
select * from pokemontype
insert into pokemontype(typeid,typename) values (1,'fire');
insert into pokemontype(typeid,typename) values (2,'water');
insert into pokemontype(typeid,typename) values (3,'grass');
insert into pokemontype(typeid,typename) values (4,'psychic');
insert into pokemontype(typeid,typename) values (5,'rock');
select * from pokemontype

/*lets work on moves*/
alter table pokemonmove add column pp integer;
select * from pokemonmove
insert into pokemonmove (movename,moveid,accuracy,mtype) values
('flamethrower',1,13,1);

/*opps we forgot to add power points to our table*/
update pokemonmove set pp=50 where (moveid=1);

/*lets add more pokemon moves*/
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values ('water
gun',2,23,2,75);
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values
('ember',3,13,1,80);
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values
('whirlpool',4,12,2,70);
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values ('flame
blitz',5,17,1,90);
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values ('razor
leaf',6,22,3,50);
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values ('dream
eater',7,20,4,35);
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values ('stone
edge',8,15,5,45);
insert into pokemonmove (movename,moveid,accuracy,mtype,pp) values ('giga
drain',9,19,3,45);
```

```
/*lets see all strong moves*/
select movename,accuracy,pp from pokemonmove order by pp ;
select movename,accuracy,pp from pokemonmove order by pp*accuracy ; /*more
insightful*/

/*lets see moves whose accuracy  greater than 15*/
select movename, accuracy,pp from pokemonmove where accuracy>15;

/*lets see moves whose accuracy  greater than 10 but less than 20*/
select movename, accuracy,pp from pokemonmove where accuracy>10 and accuracy<20;

/*another way to do the same thing*/
select movename, accuracy,pp from pokemonmove where accuracy between 10 and 20;

/*lets see maximum pp*/
select max(pp) from pokemonmove;

/*lets see minimum pp*/
select min(pp) from pokemonmove;

/*lets see maximum accuracy and pp ratio*/
select max(pp)*max(accuracy) from pokemonmove; /*oops this is  wrong*/
select max(pp*accuracy) from pokemonmove; /*correct one*/

/*lets see move with maximum pp*/
/*select movename from pokemonmove where pp=max(pp); *//*does not work as
aggregate functoins are not allowed in where*/

/*lets see average pp*/
select avg(pp) from pokemonmove;

/*lets see the types which have moves above 70 pp*/
select typename from pokemontype where exists (select * from pokemonmove where
pp>70 and typeid=mtype); /*used to look at two columns together. First selects all
columns as per where condition and then checks for exists. The exists clause checks for all
elements in move table which follow the condition and returns true. here the second
condition which maps the two tables is important*/

/*avg of pp of moves from every type*/
select avg(pp) from pokemonmove group by mtype;
```



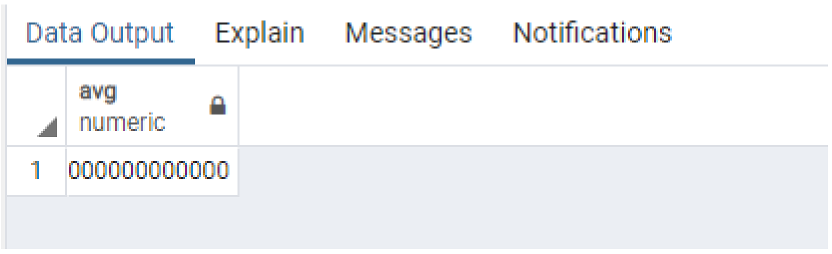
Data Output | Explain | Messages | Notifications

	movename character varying (12)	accuracy integer	pp integer
1	dream eater	20	35
2	giga drain	19	45
3	stone edge	15	45
4	razor leaf	22	50
5	flamethrower	13	50
6	whirlpool	12	70
7	water gun	23	75
8	ember	13	80
9	flame blitz	17	90



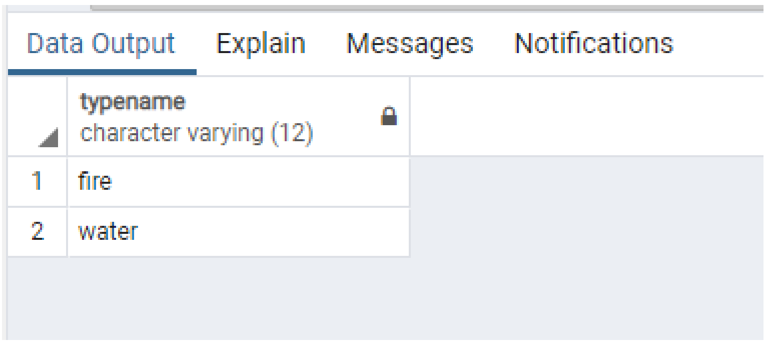
Data Output | Explain | Messages | Notifications

	movename character varying (12)	accuracy integer	pp integer
1	flamethrower	13	50
2	ember	13	80
3	whirlpool	12	70
4	flame blitz	17	90
5	stone edge	15	45
6	giga drain	19	45



Data Output | Explain | Messages | Notifications

	avg numeric
1	000000000000



	typename character varying (12)
1	fire
2	water

Conclusion:

In conclusion, performing Data Manipulation Language (DML) operations and executing nested queries with various clauses is essential in managing and manipulating data in relational databases. These operations allow for the insertion, deletion, and modification of data, as well as the retrieval of specific information based on certain conditions.

Post Lab Questions

1. **In SQL, which of the following is not a data Manipulation Language Commands?**

   a) Delete
   b) Truncate
   c) Update
   d) Create

2. **Write SQL query for following statements:**

   a. Retrieve all student who his grade has not been awarded
   SELECT * FROM students WHERE grade IS NULL;

   b. Find the names of all instructors in the Computer Science department
   SELECT name FROM instructors WHERE department = 'Computer Science';

   c. Find the names of all student whose name starts with 'S'.
   SELECT name FROM students WHERE name LIKE 'S%';

d. Find the names of instructors with salary amounts between $90,000 and $100,000.

SELECT name FROM instructors WHERE salary BETWEEN 90000 AND 100000;

e. Find all student sorted by their department name , if there are two student have the same department name , then sort them by total credit in ascending order, then by their “student name” alias.

```
SELECT name, department, total_credit

FROM students

ORDER BY department, total_credit ASC, name ASC;
```