Experiment No.: 05

Title: To implement aggregate functions with order by, group by, like and having clause.

Batch:B1 **Roll No.:16010420133**
Experiment No: 05

Aim: To implement aggregate functions with order by, group by, like and having clause.

Resources needed: PostgreSQL PgAdmin4

Theory:
The ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns.

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

The GROUP BY clause is used in collaboration with the SELECT statement to group together those rows in a table that have identical data. This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups.
The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

```
SELECT column-list
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2....columnN
ORDER BY column1, column2....columnN
```

The LIKE operator is used to match text values against a pattern using wildcards. If the search expression can be matched to the pattern expression, the LIKE operator will return true, which is 1. There are two wildcards used in conjunction with the LIKE operator:

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one, or multiple numbers or characters. The underscore represents a single number or character. These symbols can be used in combinations.
If either of these two signs is not used in conjunction with the LIKE clause, then the LIKE acts like the equals operator.

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'
or
SELECT FROM table_name
```

```
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name

WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name

WHERE column LIKE '_XXXX'

or

SELECT FROM table_name

WHERE column LIKE '_XXXX_'
```

Here are examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY::text LIKE '200%'	Finds any values that start with 200
WHERE SALARY::text LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY::text LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY::text LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY::text LIKE '%2'	Finds any values that end with 2
WHERE SALARY::text LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY::text LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

The HAVING clause allows us to pick out particular rows where the function's result meets

some condition.
The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

Results: (Queries printout with output)

1. Write 13 queries using ‘order by’, ‘group by’, ‘like’ and ‘having’ clause.
   5 with normal aggregate fun,3 with clauses and aggregate function and 5 with like operator

Example:

```
1.   SELECT * FROM COMPANY ORDER BY NAME, SALARY ASC;
2.   SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME;
3.   SELECT * FROM COMPANY WHERE AGE::text LIKE '2%';
4.   SELECT * FROM COMPANY WHERE ADDRESS  LIKE '%-%';
5.   SELECT NAME FROM COMPANY GROUP BY name HAVING count(name) > 1;
```

Aggregate Function:

CREATE TABLE building (
BName text PRIMARY KEY,
Photos text NOT NULL,
no_of_flats integer CHECK (no_of_flats > 200)
);
INSERT into building values('Galaxy2', 'Link2', 216);
INSERT into building values('Galaxy3', 'Link3', 316);
INSERT into building values('Galaxy1', 'Link1', 1016);
Select count(*) as no_of_flats from building;

Output:

	no_of_flats bigint
1	3

Select sum(no_of_flats) as total_flats from building;

Output:

	total_flats bigint
1	1548

Select avg(no_of_flats) as avg_flats from building;

Output:

	avg_flats numeric
1	516.0000000000000000

Select min(no_of_flats) as min_flats from building;

Output:

	min_flats integer
1	216

Select max(no_of_flats) as max_flats from building;

Output:

	max_flats integer
1	1016

Clause & Aggregate Function:

Select BName from building Group by BName Having sum(no_of_flats) > 600;

Output:

	bname [PK] text
1	Galaxy1

Select BName from building Group by BName Having no_of_flats >300;

Output:

	bname [PK] text
1	Galaxy1
2	Galaxy3

Select BName from building Group by BName Having no_of_flats > 100;

Output:

	bname [PK] text
1	Galaxy1
2	Galaxy3
3	Galaxy2

Like operator:

Select * from building where BName like 'G%';

Output:

	bname [PK] text	photos text	no_of_flats integer
1	Galaxy2	Link2	216
2	Galaxy3	Link3	316
3	Galaxy1	Link1	1016

Select * from building where photos like '%1';

Output:

	bname [PK] text	photos text	no_of_flats integer
1	Galaxy1	Link1	1016

Select * from building where bname like 'g%';

Output:

	bname [PK] text	photos text	no_of_flats integer

Select * from building where photos like '_i%';

Output:

	bname [PK] text	photos text	no_of_flats integer
1	Galaxy2	Link2	216
2	Galaxy3	Link3	316
3	Galaxy1	Link1	1016

Select * from building where photos like '%n%';

Output:

	bname [PK] text	photos text	no_of_flats integer
1	Galaxy2	Link2	216
2	Galaxy3	Link3	316
3	Galaxy1	Link1	1016

Outcomes: : Illustrate the concept of security, Query processing, indexing and Normalization for Relational databases

---

Questions:

Q1 Can you apply like operator on integer value? explain with example how?

We cannot use like operator on integer values. First we would need to cast it into char or varchar after which we can use it.

Eg. CAST(phone AS VARCHAR(9)) LIKE '%0203'

Q2 Why aggregate functions are more used with order by, group by and having clauses? Can we change order of these clauses when used in single query

Ans)

Aggregate functions are used with group by, having etc as these functions are used to handle groups of data, which are aggregate ether than each individual data. They should always be used in order and the order cannot be changed, as having puts a condition on group by and order by puts a condition on everything selected and we can change the order but we might not get the desired output.

Conclusion:

We implemented aggregate functions with order by, group by, like and having clause.

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

Signature of faculty in-charge with date

References:

Books:

1. Elmasri and Navathe, "Fundamentals of Database Systems", 6th Edition, Pearson Education
2. Korth, Slberchatz,Sudarshan, :"Database System Concepts", 6th Edition, McGraw – Hill.

WebSite:

1. http://www.tutorialspoint.com/postgresql/