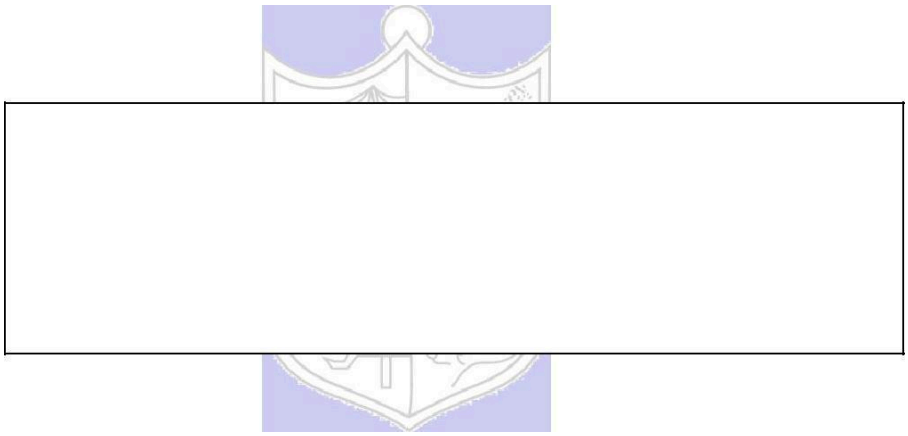


Experiment No.: 05

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

Batch:A3 **Roll No.:16010423099** **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.

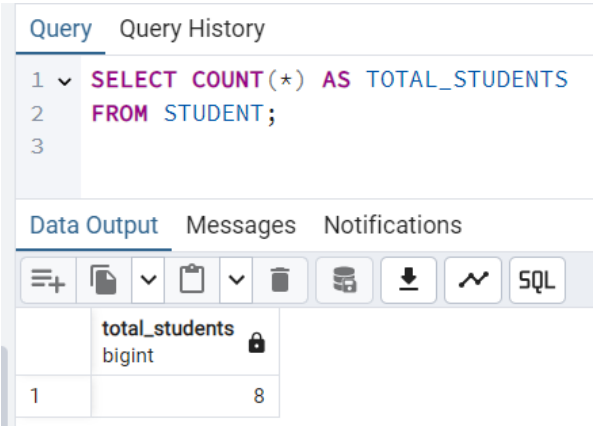
```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

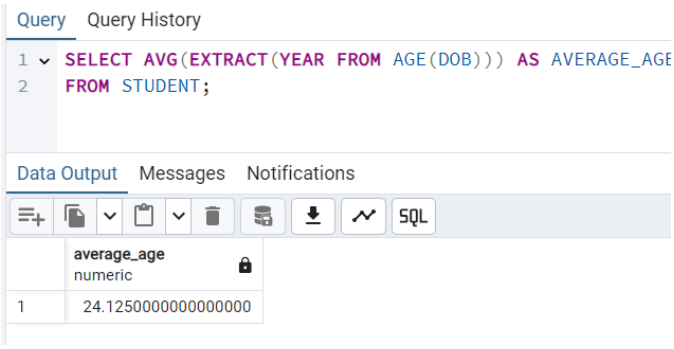
Example:

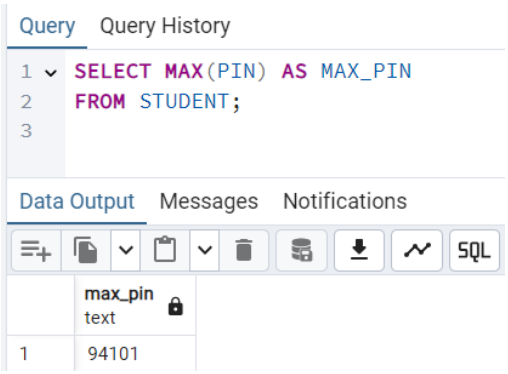
```
1.  SELECT * FROM COMPANY ORDER BY NAME, SALARY ASC;
2.  SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME;
3.  SELECT * FROM COMPANY WHERE AGE::text LIKE '2%';
4.  SELECT * FROM COMPANY WHERE ADDRESS  LIKE '%-%';
5.  SELECT NAME FROM COMPANY GROUP BY name HAVING count(name) > 1;
```

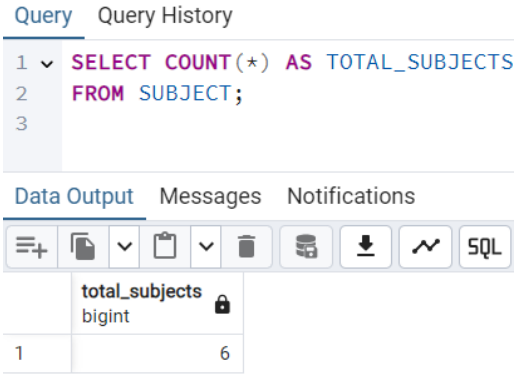
Results: (Queries printout with output)

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









Query Query History

```
1 SELECT MIN(COURSE_ID) AS MIN_COURSE_ID
2 FROM COURSE;
3
```

Data Output Messages Notifications

	min_course_id integer
1	1

Query Query History

```
1 SELECT COURSE_ID, COUNT(STUDENT_ID) AS STUDENT_COUNT
2 FROM STUDENT
3 GROUP BY COURSE_ID
4 HAVING COUNT(STUDENT_ID) > 1;
```

Data Output Messages Notifications

	course_id integer	student_count bigint
1	2	2
2	3	2
3	5	2

Query Query History

```
1 SELECT COURSE_ID, AVG(EXTRACT(YEAR FROM AGE(DOB))) AS AVG_AGE
2 FROM STUDENT
3 GROUP BY COURSE_ID
4 HAVING AVG(EXTRACT(YEAR FROM AGE(DOB))) > 20;
```

Data Output Messages Notifications

	course_id integer	avg_age numeric
1	6	25.0000000000000000
2	2	25.5000000000000000
3	3	23.0000000000000000
4	1	24.0000000000000000
5	5	23.5000000000000000

Query Query History

```
1 SELECT CITY, COUNT(STUDENT_ID) AS STUDENT_COUNT
2 FROM STUDENT
3 GROUP BY CITY
4 HAVING COUNT(STUDENT_ID) > 1;
```

Data Output Messages Notifications

	city character varying (50)	student_count bigint
1	New York	2

Query Query History

```
1 SELECT STUDENT_NAME
2 FROM STUDENT
3 WHERE STUDENT_NAME LIKE 'A%';
4
```

Data Output Messages Notifications

	student_name character varying (100)
1	Alice
2	Alice

Query Query History

```
1 SELECT SUBJECT_NAME
2 FROM SUBJECT
3 WHERE SUBJECT_NAME LIKE '%Chemistry%';
4
```

Data Output Messages Notifications

	subject_name character varying (100)
1	Organic Chemistry

Query Query History

```
1 SELECT STUDENT_NAME, CITY
2 FROM STUDENT
3 WHERE CITY LIKE '%San%';
4
```

Data Output Messages Notifications

	student_name character varying (100)	city character varying (50)
1	Charlie	San Francisco
2	Frank	San Diego

Query Query History

```
1 SELECT LECTURER_NAME
2 FROM LECTURER
3 WHERE LECTURER_NAME LIKE '%Dr.%';
4
```

Data Output Messages Notifications

	lecturer_name character varying (100)
1	Dr. Smith
2	Dr. Jones
3	Dr. Lee
4	Dr. Adams
5	Dr. Sharini
6	Dr. Johnson

Query Query History

```
1 SELECT COURSE_NAME
2 FROM COURSE
3 WHERE COURSE_NAME LIKE '%ics';
4
```

Data Output Messages Notifications

	course_name character varying (100)
1	Mathematics
2	Physics

Outcomes:

CO2: Apply data models to real world scenario.

Questions:

Q1 Can you apply like operator on integer value? explain with example how?
Yes, you can apply the LIKE operator on integer values by converting them to strings using CAST or CONVERT. For example, to find student IDs starting with '1', you can use:
SELECT * FROM students WHERE CAST(ID AS VARCHAR) LIKE '1%';

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
Aggregate functions are used with ORDER BY, GROUP BY, and HAVING clauses to summarize and filter data efficiently. The GROUP BY clause groups records, HAVING filters these groups, and ORDER BY sorts the results. The order of these clauses is fixed in SQL and cannot be changed.

Conclusion:

Successfully executed 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.