

**Experiment No.: 05**

**Title:** To implement aggregate functions with order

by, group by, like and having clauses.

**Batch: B-1** **Roll No.:** **16010422234 Name: Chandana Ramesh Galgali**

**Experiment No: 05**

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

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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 9 queries using ‘order by’, ‘group by’, ‘like’ and ‘having’ clauses.

(3 with normal aggregate functions, 3 with clauses and aggregate functions and 3 with like operators)

SELECT \* FROM Flights ORDER BY Departure\_Time DESC;

SELECT \* FROM Passenger WHERE Name LIKE 'John%';

SELECT Destination, COUNT(\*) AS Total\_Flights FROM Flights GROUP BY Destination;

SELECT \* FROM Tickets WHERE Seat\_Type LIKE 'Economy%' ORDER BY Price ASC;

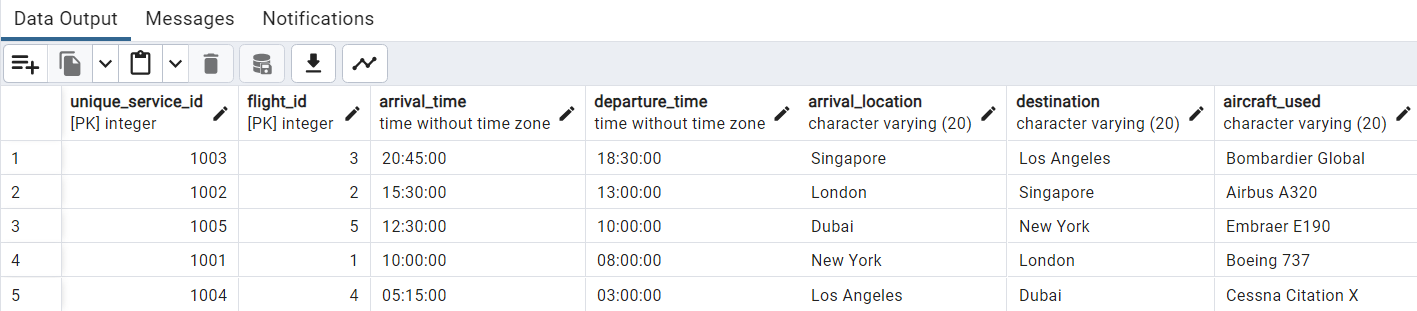
SELECT Airline\_Name, MAX(Fleet\_Size) AS Max\_Fleet\_Size FROM Airline GROUP BY Airline\_Name;

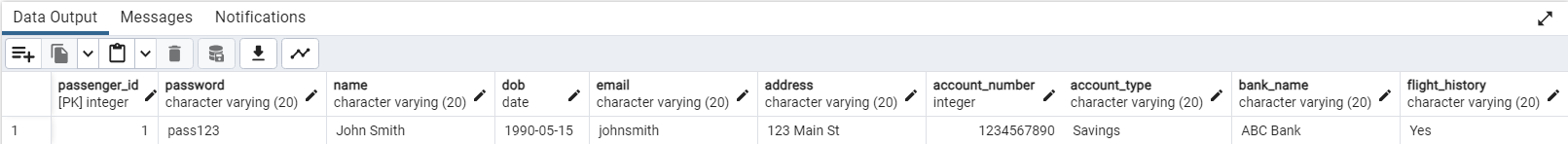
SELECT Engine\_Design, COUNT(\*) FROM Aircraft GROUP BY Engine\_Design;

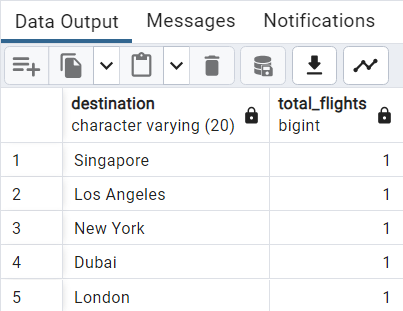
SELECT Airline\_Name, Country\_Of\_Origin FROM Airline WHERE Country\_Of\_Origin LIKE '%United%';

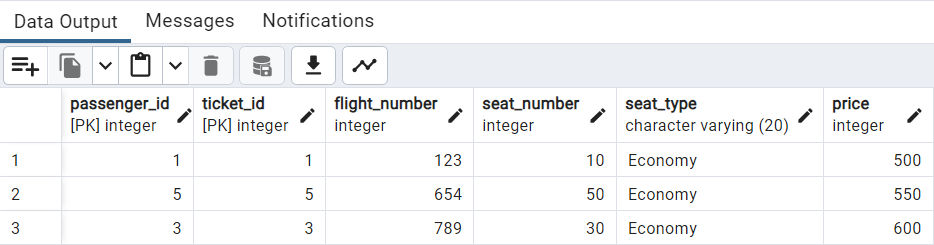
SELECT Aircraft\_ID, Airline\_Name FROM Operated\_By WHERE Airline\_Name LIKE '%Airlines';

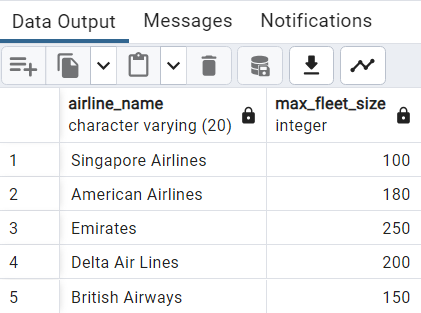
SELECT \* FROM Aircraft\_Model WHERE Aircraft\_Model LIKE 'C%C%';

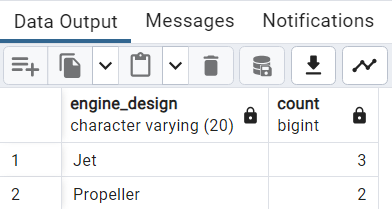


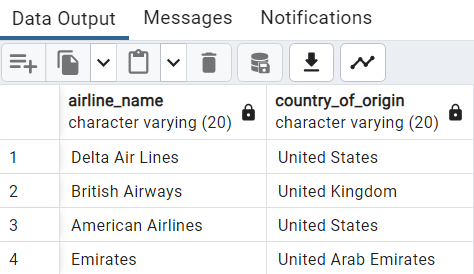


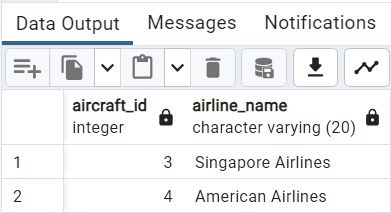


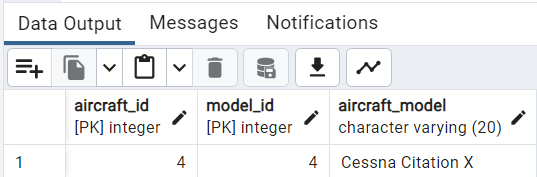












**Example:**

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;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Outcomes: Apply data models to real world scenario.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Questions:**

**Q1 Can you apply like operator on integer values? Explain with an example.**

**Ans:** No, the "like" operator is typically used for pattern matching in string values, not for integer values. The "like" operator is used in SQL queries to compare a string value against a pattern using wildcard characters such as "%".

For example, let's say we have a table called "Employees" with a column called "Name" that contains employee names. If we want to find all employees whose names start with "J", we can use the "like" operator as follows:

SELECT \* FROM Employees WHERE Name LIKE 'J%';

This query will return all rows from the "Employees" table where the "Name" column starts with the letter "J". However, if you try to use the "like" operator on an integer value, it will result in an error.

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

**Ans:** Aggregate functions, such as SUM, COUNT, AVG, MAX, and MIN, are commonly used in conjunction with the ORDER BY, GROUP BY, and HAVING clauses in SQL queries to perform calculations on groups of data and to control the output of the query.

The ORDER BY clause is used to sort the result set based on one or more columns. When used with aggregate functions, it allows you to sort the aggregated values in ascending or descending order.

The GROUP BY clause is used to group rows based on one or more columns. It is typically used with aggregate functions to calculate values for each group separately. The result set will then contain one row per group.

The HAVING clause is used to filter the result set based on conditions applied to the aggregated values. It allows you to specify conditions that must be met by the groups in order to be included in the output.

The order of these clauses in a single query is important. Generally, the order is as follows: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY. This order ensures that the query is executed correctly and produces the desired results. Changing the order of these clauses can lead to incorrect or unexpected results.

For example, if you change the order of the GROUP BY and HAVING clauses, the query may not produce the intended results because the HAVING clause operates on the grouped data. Similarly, changing the order of the ORDER BY and GROUP BY clauses may result in sorting the data before it is grouped, which can lead to incorrect sorting.

Therefore, it is recommended to follow the standard order of these clauses in SQL queries to ensure accurate and expected results.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Conclusion:**

The experiment successfully demonstrated the implementation of aggregate functions with the ORDER BY, GROUP BY, LIKE, and HAVING clauses. These clauses, when used together, provide powerful capabilities for performing calculations, filtering, and sorting in SQL queries.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.