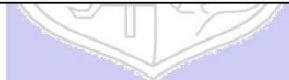




Experiment No.: 05

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



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;`

Outcomes:

Altering the Table to make it have 10 Rows

-- Update existing rows with city names and distances

UPDATE routes SET

origin = 'Delhi',

destination = 'Mumbai',

distance = 1400

WHERE route_id = 1;

UPDATE routes SET

origin = 'Kolkata',

destination = 'Chennai',

distance = 1650

WHERE route_id = 2;

UPDATE routes SET

origin = 'Bengaluru',

destination = 'Hyderabad',

distance = 600

WHERE route_id = 3;

UPDATE routes SET

origin = 'Pune',

destination = 'Goa',

distance = 450

WHERE route_id = 4;

UPDATE routes SET

origin = 'Jaipur',

destination = 'Agra',

distance = 240

WHERE route_id = 5;

-- Insert additional rows with city names and distances

INSERT INTO routes (route_id, origin, destination, distance) VALUES

(6, 'Ahmedabad', 'Surat', 270),

(7, 'Lucknow', 'Kanpur', 90),

(8, 'Chandigarh', 'Shimla', 115),

(9, 'Indore', 'Bhopal', 190),

(10, 'Nagpur', 'Aurangabad', 250);

The first screenshot shows a SQL query in pgAdmin 4 that updates the 'routes' table. The query sets the origin, destination, and distance for routes 1 through 10. The second screenshot shows the same query with additional rows inserted into the 'routes' table.

Query 1 (First Screenshot):

```

1 select * from routes
2
3 -- Update existing rows with city names and distances
4 UPDATE routes SET
5   origin = 'Delhi',
6   destination = 'Mumbai',
7   distance = 1400
8 WHERE route_id = 1;
9
10 UPDATE routes SET
11   origin = 'Kolkata',
12   destination = 'Chennai',
13   distance = 1650
14 WHERE route_id = 2;
15
16 UPDATE routes SET
17   origin = 'Bengaluru',
18   destination = 'Hyderabad',
19   distance = 600
20 WHERE route_id = 3;
21
22 UPDATE routes SET
23   origin = 'Pune',
24   destination = 'Goa',
25   distance = 450
26 WHERE route_id = 4;

```

Data Output (First Screenshot):

route_id	origin	destination	distance
1	Delhi	Mumbai	1400
2	Kolkata	Chennai	1650
3	Bengaluru	Hyderabad	600
4	Pune	Goa	450
5	Jaipur	Agra	240
6	Ahmedabad	Surat	270
7	Lucknow	Kanpur	90
8	Chandigarh	Shimla	115
9	Indore	Bhopal	190
10	Nagpur	Aurangabad	250

Query 2 (Second Screenshot):

```

17 UPDATE routes SET
18   origin = 'Bengaluru',
19   destination = 'Hyderabad',
20   distance = 600
21 WHERE route_id = 3;
22
23 UPDATE routes SET
24   origin = 'Pune',
25   destination = 'Goa',
26   distance = 450
27 WHERE route_id = 4;
28
29 UPDATE routes SET
30   origin = 'Jaipur',
31   destination = 'Agra',
32   distance = 240
33 WHERE route_id = 5;
34
35 -- Insert additional rows with city names and distances
36 INSERT INTO routes (route_id, origin, destination, distance) VALUES
37 (6, 'Ahmedabad', 'Surat', 270),
38 (7, 'Lucknow', 'Kanpur', 90),
39 (8, 'Chandigarh', 'Shimla', 115),
40 (9, 'Indore', 'Bhopal', 190),
41 (10, 'Nagpur', 'Aurangabad', 250);

```

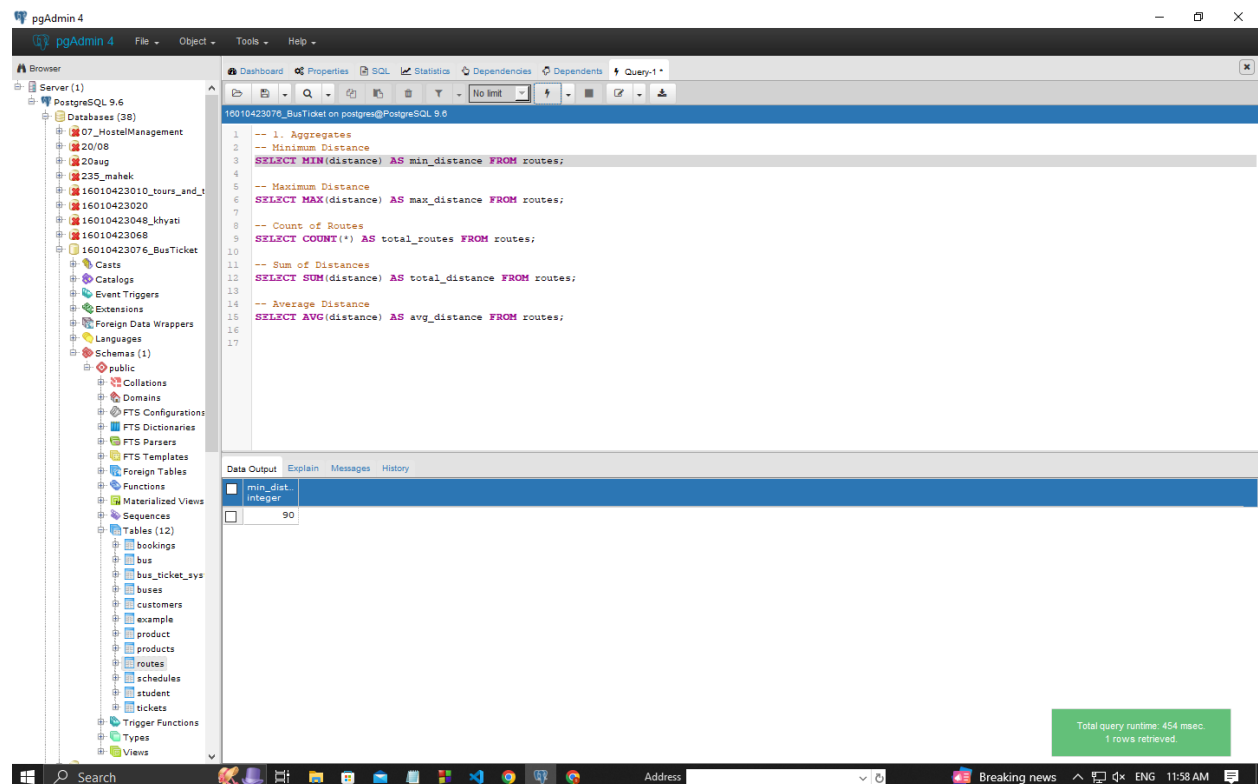
Data Output (Second Screenshot):

route_id	origin	destination	distance
1	Delhi	Mumbai	1400
2	Kolkata	Chennai	1650
3	Bengaluru	Hyderabad	600
4	Pune	Goa	450
5	Jaipur	Agra	240
6	Ahmedabad	Surat	270
7	Lucknow	Kanpur	90
8	Chandigarh	Shimla	115
9	Indore	Bhopal	190
10	Nagpur	Aurangabad	250

1. Simple Aggregates

-- Minimum Distance

SELECT MIN(distance) AS min_distance FROM routes;



The screenshot shows the pgAdmin 4 interface. On the left, the 'Server (1)' tree is expanded to show the 'public' schema, which contains a table named 'routes'. The main pane displays a SQL query with several aggregate functions:

```

1 -- 1. Aggregates
2 -- Minimum Distance
3 SELECT MIN(distance) AS min_distance FROM routes;
4
5 -- Maximum Distance
6 SELECT MAX(distance) AS max_distance FROM routes;
7
8 -- Count of Routes
9 SELECT COUNT(*) AS total_routes FROM routes;
10
11 -- Sum of Distances
12 SELECT SUM(distance) AS total_distance FROM routes;
13
14 -- Average Distance
15 SELECT AVG(distance) AS avg_distance FROM routes;
16
17

```

The 'Data Output' tab is selected, showing the results of the first query. The result is a single row with the value 90 for the 'min_dist' column, which is of type 'integer'.

min_dist
90

A green status bar at the bottom right indicates: 'Total query runtime: 454 msec. 1 rows retrieved.'

-- Maximum Distance

SELECT MAX(distance) AS max_distance FROM routes;

The screenshot shows the pgAdmin 4 interface. On the left, the 'Server (1)' tree is expanded to show the 'public' schema, which contains a table named 'routes'. The main pane displays a SQL query with several aggregate functions:

```

1 -- 1. Aggregates
2 -- Minimum Distance
3 SELECT MIN(distance) AS min_distance FROM routes;
4
5 -- Maximum Distance
6 SELECT MAX(distance) AS max_distance FROM routes;
7
8 -- Count of Routes
9 SELECT COUNT(*) AS total_routes FROM routes;
10
11 -- Sum of Distances
12 SELECT SUM(distance) AS total_distance FROM routes;
13
14 -- Average Distance
15 SELECT AVG(distance) AS avg_distance FROM routes;
16
17

```

The 'Data Output' tab shows the result of the third query, 'Count of Routes':

total_routes
1650

A green status bar at the bottom right indicates: 'Total query runtime: 517 msec. 1 rows retrieved.'

-- Count of Routes

SELECT COUNT(*) AS total_routes FROM routes;

The screenshot shows the pgAdmin 4 interface. On the left, the 'Server (1)' tree is expanded to show the 'public' schema, which contains a table named 'routes'. The main pane displays a SQL query with several aggregate functions:

```

1 -- 1. Aggregates
2 -- Minimum Distance
3 SELECT MIN(distance) AS min_distance FROM routes;
4
5 -- Maximum Distance
6 SELECT MAX(distance) AS max_distance FROM routes;
7
8 -- Count of Routes
9 SELECT COUNT(*) AS total_routes FROM routes;
10
11 -- Sum of Distances
12 SELECT SUM(distance) AS total_distance FROM routes;
13
14 -- Average Distance
15 SELECT AVG(distance) AS avg_distance FROM routes;
16
17

```

The 'Data Output' tab shows the result of the third query, 'Count of Routes':

total_routes
10

A green status bar at the bottom right indicates: 'Total query runtime: 400 msec. 1 rows retrieved.'

-- Sum of Distances

SELECT SUM(distance) AS total_distance FROM routes;

The screenshot shows the pgAdmin 4 interface. On the left, the 'Server (1)' tree is expanded to show the 'public' schema, which contains a table named 'routes'. The main pane displays a SQL query in the 'Query Tool' tab:

```

1 -- 1. Aggregates
2 -- Minimum Distance
3 SELECT MIN(distance) AS min_distance FROM routes;
4
5 -- Maximum Distance
6 SELECT MAX(distance) AS max_distance FROM routes;
7
8 -- Count of Routes
9 SELECT COUNT(*) AS total_routes FROM routes;
10
11 -- Sum of Distances
12 SELECT SUM(distance) AS total_distance FROM routes;
13
14 -- Average Distance
15 SELECT AVG(distance) AS avg_distance FROM routes;
16
17

```

The 'Data Output' tab shows the results of the query. The first row, labeled 'total_distance', has a value of 5255. A green status bar at the bottom right indicates 'Total query runtime: 517 msec. 1 rows retrieved.'

-- Average Distance

SELECT AVG(distance) AS avg_distance FROM routes;

The screenshot shows the pgAdmin 4 interface. On the left, the 'Server (1)' tree is expanded to show the 'public' schema, which contains a table named 'routes'. The main pane displays a SQL query with several aggregate functions:

```

1  -- 1. Aggregates
2  -- Minimum Distance
3  SELECT MIN(distance) AS min_distance FROM routes;
4
5  -- Maximum Distance
6  SELECT MAX(distance) AS max_distance FROM routes;
7
8  -- Count of Routes
9  SELECT COUNT(*) AS total_routes FROM routes;
10
11 -- Sum of Distances
12 SELECT SUM(distance) AS total_distance FROM routes;
13
14 -- Average Distance
15 SELECT AVG(distance) AS avg_distance FROM routes;
16
17

```

The 'Data Output' pane shows the result of the query, which is a single row with the value 525.500001 for the 'avg_dist' column.

avg_dist
525.500001

A green status bar at the bottom right indicates: 'Total query runtime: 305 msec. 1 rows retrieved.'

2. Aggregate with Clauses (ORDER BY, GROUP BY, HAVING)

-- Sum of Distances, Ordered by Distance

```
SELECT SUM(distance) AS total_distance FROM routes
```

```
ORDER BY total_distance DESC;
```

The screenshot shows the pgAdmin 4 interface with a PostgreSQL 9.6 database. The left sidebar displays the database structure, including tables like 'routes'. The main pane shows a SQL query with the following content:

```

12 SELECT SUM(distance) AS total_distance FROM routes;
13
14 -- Average Distance
15 SELECT AVG(distance) AS avg_distance FROM routes;
16
17
18 -- 2. Aggregate with Clauses (ORDER BY, GROUP BY, HAVING)
19
20 -- Sum of Distances, Ordered by Distance
21 SELECT SUM(distance) AS total_distance FROM routes
22 ORDER BY total_distance DESC;
23
24 -- Count of Routes, Grouped by Origin City
25 SELECT origin, COUNT(*) AS route_count FROM routes
26 GROUP BY origin
27 ORDER BY route_count DESC;
28
29 -- Having Clause - Cities with More than 1 Route
30 SELECT origin, COUNT(*) AS route_count FROM routes
31 GROUP BY origin
32 HAVING COUNT(*) > 1;
33
34
35 -- 3. Queries Using LIKE Operator
36

```

The 'Data Output' pane shows the result of the query:

total_distance	bigint
5255	

-- Count of Routes, Grouped by Origin City

SELECT origin, COUNT(*) AS route_count FROM routes

GROUP BY origin

ORDER BY route_count DESC;

The screenshot shows the pgAdmin 4 interface with a PostgreSQL 9.6 database. The left sidebar displays the database structure, including schemas, tables, and views. The main window shows a SQL query editor with the following queries:

```

12 SELECT SUM(distance) AS total_distance FROM routes;
13
14 -- Average Distance
15 SELECT AVG(distance) AS avg_distance FROM routes;
16
17
18 -- 2. Aggregate with Clauses (ORDER BY, GROUP BY, HAVING)
19
20 -- Sum of Distances, Ordered by Distance
21 SELECT SUM(distance) AS total_distance FROM routes
22 ORDER BY total_distance DESC;
23
24 -- Count of Routes, Grouped by Origin City
25 SELECT origin, COUNT(*) AS route_count FROM routes
26 GROUP BY origin
27 ORDER BY route_count DESC;
28
29 -- Having Clause - Cities with More than 1 Route
30 SELECT origin, COUNT(*) AS route_count FROM routes
31 GROUP BY origin
32 HAVING COUNT(*) > 1;
33
34
35 -- 3. Queries Using LIKE Operator
36

```

The Data Output tab shows the results of the query: `SELECT origin, COUNT(*) AS route_count FROM routes GROUP BY origin ORDER BY route_count DESC;`

origin	route_count
Lucknow	1
Indore	1
Pune	1
Nagpur	1
Ahmedabad	1
Bengaluru	1
Kolkata	1
Chandigarh	1
Jaipur	1
Delhi	1

Total query runtime: 340 msec.
10 rows retrieved.

-- Having Clause - Cities with More than 1 Route

SELECT origin, COUNT(*) AS route_count FROM routes

GROUP BY origin

HAVING COUNT(*) > 1;

The screenshot shows the pgAdmin 4 interface with a PostgreSQL 9.6 database. The left sidebar displays the database structure, including schemas, tables, and views. The main pane shows a SQL query with the following content:

```

16
17
18
19 -- 2. Aggregate with Clauses (ORDER BY, GROUP BY, HAVING)
20
21 -- Sum of Distances, Ordered by Distance
22 SELECT SUM(distance) AS total_distance FROM routes
23 ORDER BY total_distance DESC;
24
25 -- Count of Routes, Grouped by Origin City
26 SELECT origin, COUNT(*) AS route_count FROM routes
27 GROUP BY origin
28 ORDER BY route_count DESC;
29
30 -- Having Clause - Cities with More than 1 Route
31 SELECT origin, COUNT(*) AS route_count FROM routes
32 GROUP BY origin
33 HAVING COUNT(*) > 1;
34
35
36 -- 3. Queries Using LIKE Operator
37
38 -- Find Routes Originating from Cities Starting with 'D'
39 SELECT * FROM routes WHERE origin LIKE 'D%';
40

```

The Data Output pane shows the following columns:

origin	route_count
character varying (100)	bigint

Total query runtime: 623 msec.
0 rows retrieved.

3. Queries Using LIKE Operator

-- Find Routes Originating from Cities Starting with 'D'

```
SELECT * FROM routes WHERE origin LIKE 'D%';
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Server (1)' tree is expanded to show the 'public' schema, with the 'routes' table selected. The main pane displays a SQL query in the 'Query-1' editor:

```

34 SELECT origin, COUNT(*) AS route_count FROM routes
35 GROUP BY origin
36 HAVING COUNT(*) > 1;
37
38
39
40
41 -- 3. Queries Using LIKE Operator
42
43 -- Find Routes Originating from Cities Starting with 'D'
44 SELECT * FROM routes WHERE origin LIKE 'D%';
45
46 -- Find Routes Ending in Cities Starting with 'M'
47 SELECT * FROM routes WHERE destination LIKE 'M%';
48
49 -- Find Routes Where Origin Contains 'h'
50 SELECT * FROM routes WHERE origin LIKE '%h%';
51
52 -- Find Routes Where Destination Ends with 'a'
53 SELECT * FROM routes WHERE destination LIKE '%a';
54
55 -- Find Routes Where Distance is a Three-Digit Number
56 SELECT * FROM routes WHERE CAST(distance AS TEXT) LIKE '___';
57
58

```

Below the query editor, the 'Data Output' tab shows the results of the query. The table has 4 columns: route_id, origin, destination, and distance. The results are as follows:

route_id	origin	destination	distance
1	Delhi	Mumbai	1400

A green box at the bottom right indicates: 'Total query runtime: 330 msec. 1 rows retrieved.'

-- Find Routes Ending in Cities Starting with 'M'

SELECT * FROM routes WHERE destination LIKE 'M%';

pgAdmin 4

Server (1)

PostgreSQL 9.6

Databases (28)

- 07_HotelManagement
- 20/08
- 20aug
- 235_mahak
- 16010423010_tours_and_t
- 16010423020
- 16010423048_khyati
- 16010423068
- 16010423076_BusTicket
- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas (1)
 - public
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Sequences
 - Tables (12)
 - bookings
 - bus
 - bus_ticket_sys
 - buses
 - customers
 - example
 - product
 - products
 - routes
 - schedules
 - student
 - tickets
 - Trigger Functions
 - Types
 - Views

16010423076_BusTicket on postgres@PostgreSQL 9.6

```

34 SELECT origin, COUNT(*) AS route_count FROM routes
35 GROUP BY origin
36 HAVING COUNT(*) > 1;
37
38
39
40
41 -- 3. Queries Using LIKE Operator
42
43 -- Find Routes Originating from Cities Starting with 'D'
44 SELECT * FROM routes WHERE origin LIKE 'D%';
45
46 -- Find Routes Ending in Cities Starting with 'M'
47 SELECT * FROM routes WHERE destination LIKE '%M';
48
49 -- Find Routes Where Origin Contains 'h'
50 SELECT * FROM routes WHERE origin LIKE '%h%';
51
52 -- Find Routes Where Destination Ends with 'a'
53 SELECT * FROM routes WHERE destination LIKE '%a';
54
55 -- Find Routes Where Distance is a Three-Digit Number
56 SELECT * FROM routes WHERE CAST(distance AS TEXT) LIKE '___';
57
58

```

Data Output Explain Messages History

route_id	origin	destination	distance
integer	Character varying (100)	Character varying (100)	integer
1	Delhi	Mumbai	1400

Total query runtime: 344 msec.
1 rows retrieved.

Search

Address

30°C Mostly cloudy

ENG 12:03 PM

-- Find Routes Where Origin Contains 'h'

SELECT * FROM routes WHERE origin LIKE '%h%';

pgAdmin 4

Server (1)

PostgreSQL 9.6

Databases (38)

07_HostelManagement

20/08

20aug

235_mahak

16010423010_tours_and_t

16010423020

16010423048_khyati

16010423068

16010423076_BusTicket

Casts

Catalogs

Event Triggers

Extensions

Foreign Data Wrappers

Languages

Schemas (1)

public

Collations

Domains

FTS Configurations

FTS Dictionaries

FTS Parsers

FTS Templates

Foreign Tables

Functions

Materialized Views

Sequences

Tables (12)

bookings

bus

bus_ticket_sys

buses

customers

example

product

products

routes

schedules

student

tickets

Trigger Functions

Types

Views

Dashboard Properties SQL Statistics Dependencies Dependents Query-1

16010423076_BusTicket on postgres@PostgreSQL 9.6

```

34 SELECT origin, COUNT(*) AS route_count FROM routes
35 GROUP BY origin
36 HAVING COUNT(*) > 1;
37
38
39
40
41 -- 3. Queries Using LIKE Operator
42
43 -- Find Routes Originating from Cities Starting with 'D'
44 SELECT * FROM routes WHERE origin LIKE 'D%';
45
46 -- Find Routes Ending in Cities Starting with 'M'
47 SELECT * FROM routes WHERE destination LIKE 'M%';
48
49 -- Find Routes Where Origin Contains 'h'
50 SELECT * FROM routes WHERE origin LIKE '%h%';
51
52 -- Find Routes Where Destination Ends with 'a'
53 SELECT * FROM routes WHERE destination LIKE '%a';
54
55 -- Find Routes Where Distance is a Three-Digit Number
56 SELECT * FROM routes WHERE CAST(distance AS TEXT) LIKE '___';
57
58

```

Data Output Explain Messages History

route_id	origin	destination	distance
1	Delhi	Mumbai	1400
6	Ahmedabad	Surat	270
8	Chandigarh	Shimla	115

Total query runtime: 351 msec.
3 rows retrieved.

-- Find Routes Where Destination Ends with 'a'

SELECT * FROM routes WHERE destination LIKE '%a';

pgAdmin 4

Server (1)

PostgreSQL 9.6

Databases (38)

07_HostelManagement

20/08

20aug

235_mahak

16010423010_tours_and_t

16010423020

16010423048_khyati

16010423068

16010423076_BusTicket

Casts

Catalogs

Event Triggers

Extensions

Foreign Data Wrappers

Languages

Schemas (1)

public

Collations

Domains

FTS Configurations

FTS Dictionaries

FTS Parsers

FTS Templates

Foreign Tables

Functions

Materialized Views

Sequences

Tables (12)

bookings

bus

bus_ticket_sys

buses

customers

example

product

products

routes

schedules

student

tickets

Trigger Functions

Types

Views

Dashboard Properties SQL Statistics Dependencies Dependents Query-1

16010423076_BusTicket on postgres@PostgreSQL 9.6

```

34 SELECT origin, COUNT(*) AS route_count FROM routes
35 GROUP BY origin
36 HAVING COUNT(*) > 1;
37
38
39
40
41 -- 3. Queries Using LIKE Operator
42
43 -- Find Routes Originating from Cities Starting with 'D'
44 SELECT * FROM routes WHERE origin LIKE 'D%';
45
46 -- Find Routes Ending in Cities Starting with 'M'
47 SELECT * FROM routes WHERE destination LIKE 'M%';
48
49 -- Find Routes Where Origin Contains 'h'
50 SELECT * FROM routes WHERE origin LIKE '%h%';
51
52 -- Find Routes Where Destination Ends with 'a'
53 SELECT * FROM routes WHERE destination LIKE '%a';
54
55 -- Find Routes Where Distance is a Three-Digit Number
56 SELECT * FROM routes WHERE CAST(distance AS TEXT) LIKE '___';
57
58

```

Data Output Explain Messages History

route_id	origin	destination	distance
4	Pune	Goa	450
5	Jaipur	Agra	240
8	Chandigarh	Shimla	115

Total query runtime: 336 msec.
3 rows retrieved.

-- Find Routes Where Distance is a Three-Digit Number

SELECT * FROM routes WHERE CAST(distance AS TEXT) LIKE '___';

The screenshot shows the pgAdmin 4 interface. The left pane displays the database structure for 'PostgreSQL 9.6', including databases like '07_HotelManagement', '20/08', '20aug', '235_mahak', and '16010423010_tours_and_t'. The right pane shows a SQL query window with the following content:

```

16010423076_BusTicket on postgres@PostgreSQL 9.6
34 SELECT origin, COUNT(*) AS route_count FROM routes
35 GROUP BY origin
36 HAVING COUNT(*) > 1;
37
38
39
40
41 -- 3. Queries Using LIKE Operator
42
43 -- Find Routes Originating from Cities Starting with 'D'
44 SELECT * FROM routes WHERE origin LIKE 'D%';
45
46 -- Find Routes Ending in Cities Starting with 'H'
47 SELECT * FROM routes WHERE destination LIKE 'H%';
48
49 -- Find Routes Where Origin Contains 'h'
50 SELECT * FROM routes WHERE origin LIKE '%h%';
51
52 -- Find Routes Where Destination Ends with 'a'
53 SELECT * FROM routes WHERE destination LIKE '%a';
54
55 -- Find Routes Where Distance is a Three-Digit Number
56 SELECT * FROM routes WHERE distance BETWEEN 100 AND 999;
57
58

```

Below the query window, the 'Data Output' tab shows the results of the query. The table has four columns: 'route_id', 'origin', 'destination', and 'distance'. The data is as follows:

route_id	origin	destination	distance
3	Bengaluru	Hyderabad	600
4	Pune	Goa	450
5	Jaipur	Agra	240
6	Ahmedabad	Surat	270
8	Chandigarh	Shimla	115
9	Indore	Bhopal	190
10	Nagpur	Aurangabad	250

A green status bar at the bottom right indicates: 'Total query runtime: 511 msec. 7 rows retrieved.'

Questions:

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

In PostgreSQL (and SQL in general), the LIKE operator is primarily used for pattern matching on string values. It's not directly applicable to integer values because integers are not inherently text-based and do not have patterns in the same sense as strings do.

However, you can convert integers to strings and then apply the LIKE operator. This is done using the CAST or ::text operation to convert the integer to a text type. Here's an example:

-- Sample table

```

CREATE TABLE numbers (
  id SERIAL PRIMARY KEY,
  value INT
);

```

-- Insert some values

```
INSERT INTO numbers (value) VALUES (123), (456), (789);
```

-- Query using LIKE operator on integer values

```
SELECT * FROM numbers
WHERE value::text LIKE '12%';
```

In this example, the integer values are cast to text using value::text, and then the LIKE operator is used to find values where the text representation of the number starts with '12'.

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 (like SUM(), COUNT(), AVG(), etc.) are typically used in conjunction with GROUP BY, ORDER BY, and HAVING clauses in SQL queries for several reasons:

- **GROUP BY:** This clause groups rows that have the same values into summary rows, like "total sales per region". Aggregate functions are then applied to these groups.
- **HAVING:** This clause filters groups based on a condition, similar to WHERE, but it operates on aggregated data. For example, you might want to find regions where the total sales exceed a certain amount.
- **ORDER BY:** This clause sorts the result set. After aggregation, you might want to order the results based on the aggregated values, like sorting by total sales in descending order.

-- Sample table

```
CREATE TABLE sales (
    region TEXT,
    amount NUMERIC
);
```

-- Insert some values

```
INSERT INTO sales (region, amount) VALUES
('North', 100),
('North', 200),
('South', 150),
('South', 50),
('East', 300),
('West', 250);
```

-- Query using GROUP BY, HAVING, and ORDER BY

```
SELECT region, SUM(amount) AS total_sales
FROM sales
GROUP BY region
HAVING SUM(amount) > 100
ORDER BY total_sales DESC;
```

In this example:

- GROUP BY is used to aggregate sales by region.
- HAVING filters out regions where total sales are less than or equal to 100.
- ORDER BY sorts the results by total_sales in descending order.

Clause Order:

In a single SQL query, the order of these clauses is fixed and cannot be changed:

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

The order is important because SQL processes the clauses in this sequence. For example, GROUP BY must come before HAVING because HAVING is meant to filter groups created by GROUP BY.

Conclusion:

I learned how to update and insert rows in SQL tables, use simple aggregate functions like SUM and COUNT, and apply conditions using GROUP BY, HAVING, and ORDER BY. I also understood how to use the LIKE operator on strings and convert integers to text for pattern matching. Overall, it enhanced my understanding of querying and manipulating data in SQL.

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, Silberchatz, Sudarshan, "Database System Concepts", 6th Edition, McGraw – Hill.