Deccan Education Society's (DES)
Pune University, Pune
School of Engineering and Technology
Department of Computer Engineering and Technology
Program: B. Tech in Computer Science and Engineering

Academic Year: 2024-25	**Year:** Second Year	**Semester:** II
PRN No.: 1012412079	**Name: Ratnajeet Patil**	
Subject: Database Management System		
Assignment No.: 4		
Date:		

Lab Assignment: 04

Title: Advanced SQL Queries: Design and execute following SQL queries on suitable applications.
Joins: Retrieve data from multiple related tables using INNER JOIN, LEFT JOIN, and RIGHT JOIN.
Aggregation: Use aggregate functions like COUNT, AVG, SUM, MIN, MAX in combination with GROUP BY and HAVING clauses.

Theory:

What is JOIN in SQL?

A JOIN in SQL is used to combine rows from two or more tables based on a related column between them. It allows retrieving data from multiple tables in a single query.

Different Types of JOINs

1. INNER JOIN – Returns only the matching rows between the two tables.

```
1. SELECT A.*, B.*
2. FROM TableA A
3. INNER JOIN TableB B
4. ON A.common_column = B.common_column;
5.
```

2. LEFT JOIN (LEFT OUTER JOIN) – Returns all rows from the left table and the matched rows from the right table. If no match is found, NULLs are returned.

```
1. SELECT A.*, B.*
2. FROM TableA A
3. LEFT JOIN TableB B
4. ON A.common_column = B.common_column;
5.
```

3. RIGHT JOIN (RIGHT OUTER JOIN) – Returns all rows from the right table and the matched rows from the left table.

```
1. SELECT A.*, B.*
2. FROM TableA A
3. RIGHT JOIN TableB B
4. ON A.common_column = B.common_column;
5.
```

4. FULL JOIN (FULL OUTER JOIN) – Returns all rows when there is a match in either table. If there's no match, NULLs are returned.

```
1. SELECT A.*, B.*
2. FROM TableA A
3. FULL JOIN TableB B
4. ON A.common_column = B.common_column;
5.
```

5. CROSS JOIN – Returns the Cartesian product of both tables (every row in TableA is combined with every row in TableB).

```
1. SELECT A.*, B.*
2. FROM TableA A
3. CROSS JOIN TableB B;
4.
```

6. SELF JOIN – A table joins itself by using aliases.

```
1. SELECT A.*, B.*
2. FROM TableA A
3. INNER JOIN TableA B
4. ON A.column_name = B.column_name;
5.
```

Aggregate Functions

Aggregate functions perform calculations on a set of values and return a single value.

Function	Description
COUNT()	Returns the number of rows
SUM()	Returns the total sum of a column
AVG()	Returns the average value of a column
MAX()	Returns the highest value in a column
MIN()	Returns the lowest value in a column

Group By

The GROUP BY clause is used with aggregate functions to group rows that have the same values in specified columns.

Example:

```
1. SELECT department, COUNT(*) AS total_employees
2. FROM employees
3. GROUP BY department;
4.
```

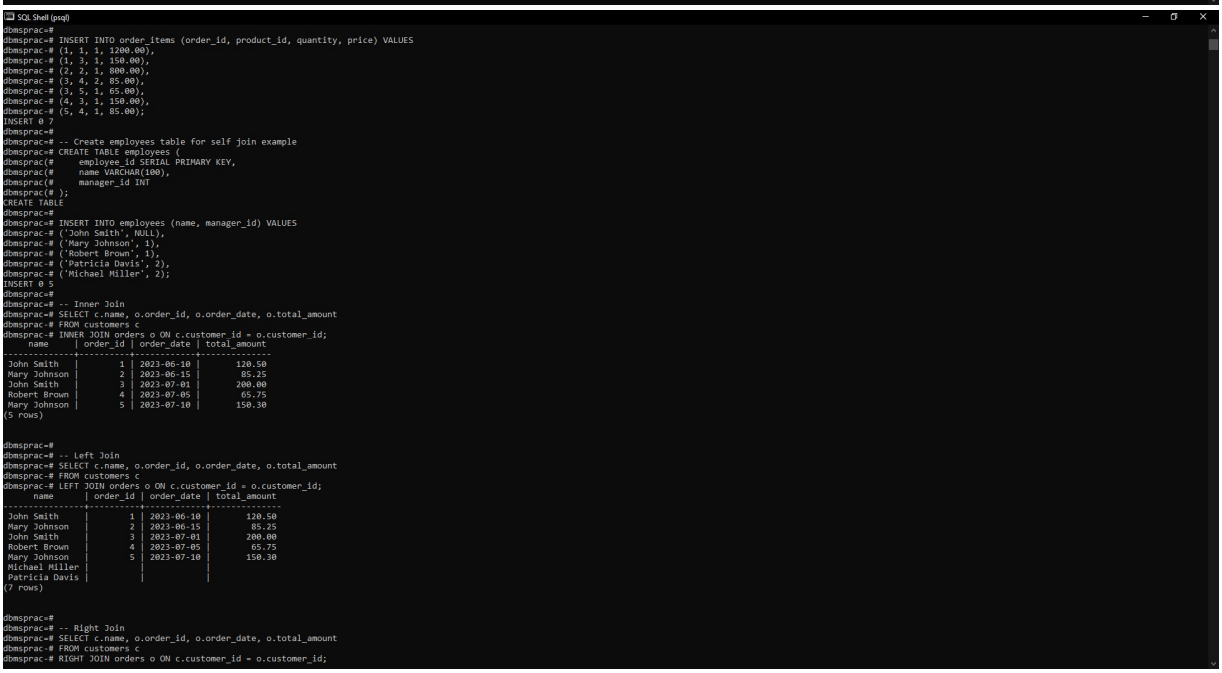
Having

The HAVING clause is used to filter groups created by GROUP BY based on aggregate function results (since WHERE cannot be used with aggregate functions).

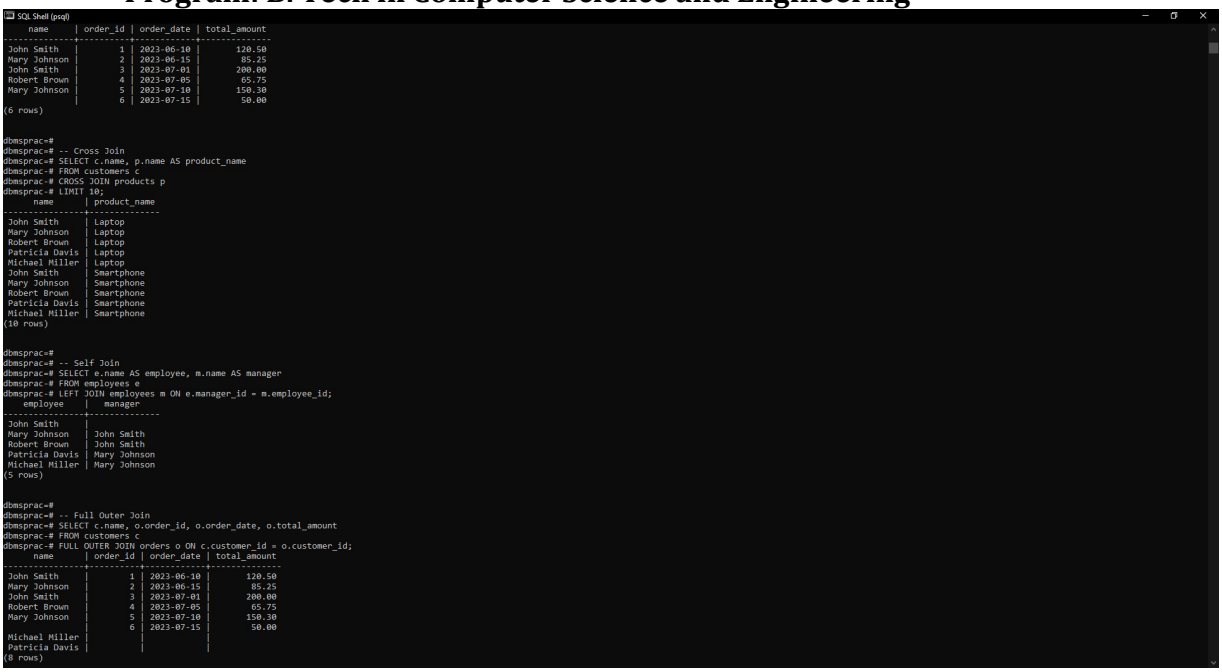
Show Query Execution Screenshots for:

- Inner join
- Left join
- Right join
- Cross join
- Self join
- Full Outer join
- All Aggregate Functions
- Group By
- Having
- Set Operators (Union, Intersect and Difference)
- Use of Date, Time, Mathematical and String Functions

```
SQL Shell (psql)
dbmsprac=# -- Create sample tables
dbmsprac=# CREATE TABLE customers (
dbmsprac(#     customer_id SERIAL PRIMARY KEY,
dbmsprac(#     name VARCHAR(100),
dbmsprac(#     email VARCHAR(100),
dbmsprac(#     city VARCHAR(50),
dbmsprac(#     join_date DATE
dbmsprac(# );
CREATE TABLE
dbmsprac=#
dbmsprac=# CREATE TABLE orders (
dbmsprac(#     order_id SERIAL PRIMARY KEY,
dbmsprac(#     customer_id INT,
dbmsprac(#     order_date DATE,
dbmsprac(#     total_amount DECIMAL(10,2)
dbmsprac(# );
CREATE TABLE
dbmsprac=#
dbmsprac=# CREATE TABLE products (
dbmsprac(#     product_id SERIAL PRIMARY KEY,
dbmsprac(#     name VARCHAR(100),
dbmsprac(#     category VARCHAR(50),
dbmsprac(#     price DECIMAL(10,2)
dbmsprac(# );
CREATE TABLE
dbmsprac=#
dbmsprac=# CREATE TABLE order_items (
dbmsprac(#     item_id SERIAL PRIMARY KEY,
dbmsprac(#     order_id INT,
dbmsprac(#     product_id INT,
dbmsprac(#     quantity INT,
dbmsprac(#     price DECIMAL(10,2)
dbmsprac(# );
CREATE TABLE
dbmsprac=#
dbmsprac=# -- Insert sample data
dbmsprac=# INSERT INTO customers (name, email, city, join_date) VALUES
dbmsprac-# ('John Smith', 'john@example.com', 'New York', '2023-01-15'),
dbmsprac-# ('Mary Johnson', 'mary@example.com', 'Los Angeles', '2023-02-10'),
dbmsprac-# ('Robert Brown', 'robert@example.com', 'Chicago', '2023-03-05'),
dbmsprac-# ('Patricia Davis', 'patricia@example.com', 'Houston', '2023-04-20'),
dbmsprac-# ('Michael Miller', 'michael@example.com', 'Phoenix', '2023-05-12');
INSERT 0 5
dbmsprac=#
dbmsprac=# INSERT INTO orders (customer_id, order_date, total_amount) VALUES
dbmsprac-# (1, '2023-06-10', 120.50),
dbmsprac-# (2, '2023-06-15', 85.25),
dbmsprac-# (1, '2023-07-01', 200.00),
dbmsprac-# (3, '2023-07-05', 65.75),
dbmsprac-# (2, '2023-07-10', 150.30),
dbmsprac-# (NULL, '2023-07-15', 50.00);
INSERT 0 6
dbmsprac=#
dbmsprac=# INSERT INTO products (name, category, price) VALUES
dbmsprac-# ('Laptop', 'Electronics', 1200.00),
dbmsprac-# ('Smartphone', 'Electronics', 800.00),
dbmsprac-# ('Headphones', 'Electronics', 150.00),
dbmsprac-# ('Coffee Maker', 'Kitchen', 85.00),
dbmsprac-# ('Blender', 'Kitchen', 65.00);
INSERT 0 5
dbmsprac=#
dbmsprac=# INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
dbmsprac-# (1, 1, 1, 1200.00),
```



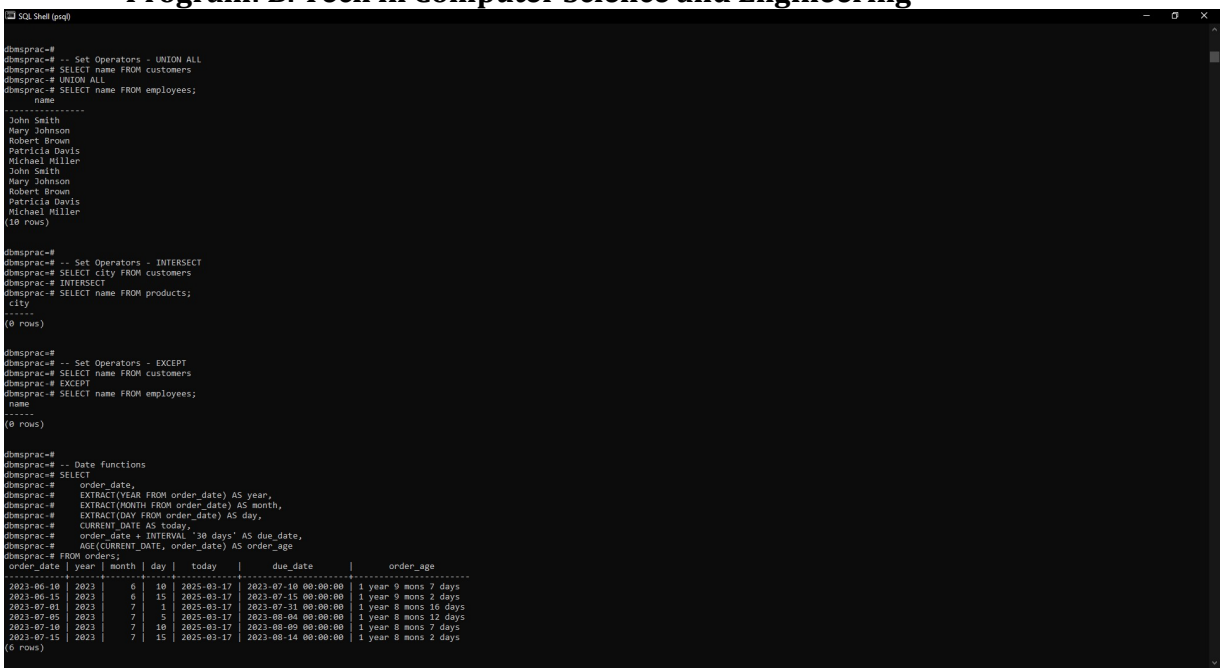
```
SQL Shell (psql)
dbmsprac=#
dbmsprac=# INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
dbmsprac-# (1, 1, 1, 1200.00),
dbmsprac-# (1, 3, 1, 150.00),
dbmsprac-# (2, 2, 1, 800.00),
dbmsprac-# (3, 4, 2, 85.00),
dbmsprac-# (3, 5, 1, 65.00),
dbmsprac-# (4, 3, 1, 150.00),
dbmsprac-# (5, 4, 1, 85.00);
INSERT 0 7
dbmsprac=#
dbmsprac=# -- Create employees table for self join example
dbmsprac=# CREATE TABLE employees (
dbmsprac(#     employee_id SERIAL PRIMARY KEY,
dbmsprac(#     name VARCHAR(100),
dbmsprac(#     manager_id INT
dbmsprac(# );
CREATE TABLE
dbmsprac=#
dbmsprac=# INSERT INTO employees (name, manager_id) VALUES
dbmsprac-# ('John Smith', NULL),
dbmsprac-# ('Mary Johnson', 1),
dbmsprac-# ('Robert Brown', 1),
dbmsprac-# ('Patricia Davis', 2),
dbmsprac-# ('Michael Miller', 2);
INSERT 0 5
dbmsprac=#
dbmsprac=# -- Inner Join
dbmsprac=# SELECT c.name, o.order_id, o.order_date, o.total_amount
dbmsprac-# FROM customers c
dbmsprac-# INNER JOIN orders o ON c.customer_id = o.customer_id;
     name     | order_id | order_date | total_amount
--------------+----------+------------+--------------
 John Smith   |        1 | 2023-06-10 |       120.50
 Mary Johnson |        2 | 2023-06-15 |        85.25
 John Smith   |        3 | 2023-07-01 |       200.00
 Robert Brown |        4 | 2023-07-05 |        65.75
 Mary Johnson |        5 | 2023-07-10 |       150.30
(5 rows)


dbmsprac=#
dbmsprac=# -- Left Join
dbmsprac=# SELECT c.name, o.order_id, o.order_date, o.total_amount
dbmsprac-# FROM customers c
dbmsprac-# LEFT JOIN orders o ON c.customer_id = o.customer_id;
     name     | order_id | order_date | total_amount
--------------+----------+------------+--------------
 John Smith   |        1 | 2023-06-10 |       120.50
 Mary Johnson |        2 | 2023-06-15 |        85.25
 John Smith   |        3 | 2023-07-01 |       200.00
 Robert Brown |        4 | 2023-07-05 |        65.75
 Mary Johnson |        5 | 2023-07-10 |       150.30
 Michael Miller |          |            |
 Patricia Davis |          |            |
(7 rows)


dbmsprac=#
dbmsprac=# -- Right Join
dbmsprac=# SELECT c.name, o.order_id, o.order_date, o.total_amount
dbmsprac-# FROM customers c
dbmsprac-# RIGHT JOIN orders o ON c.customer_id = o.customer_id;
```



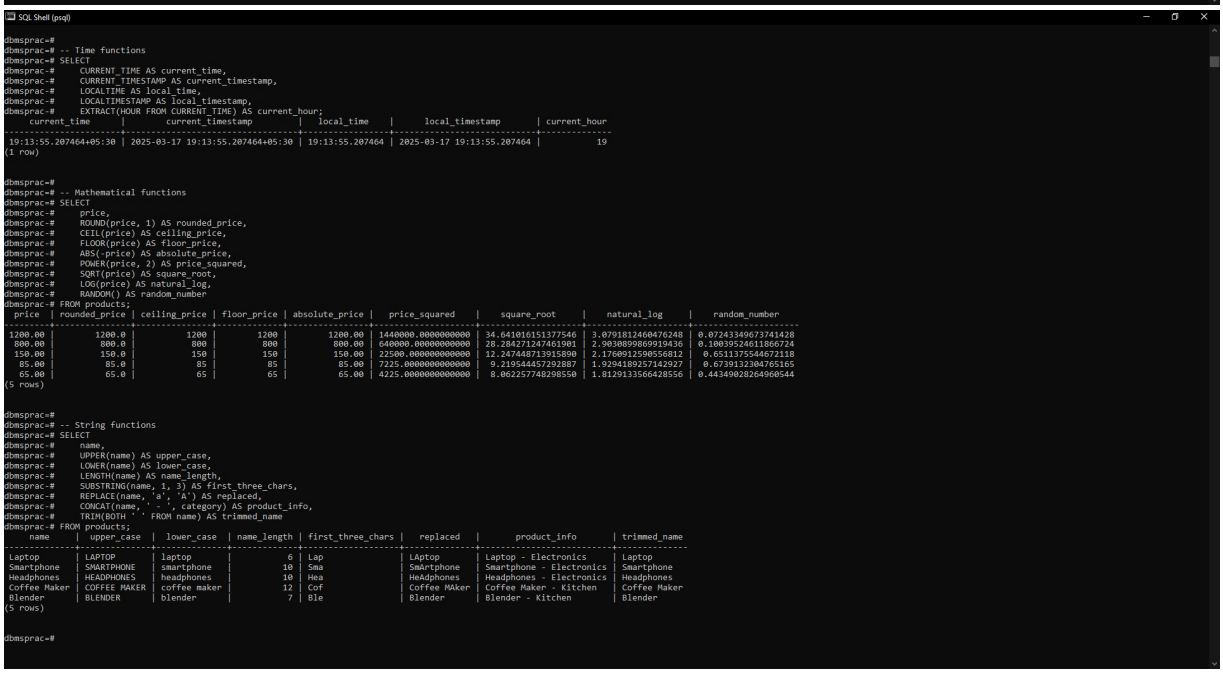
```
SQL Shell (psql)
     name      | order_id | order_date | total_amount
---------------+----------+------------+--------------
 John Smith    |        1 | 2023-06-10 |       120.50
 Mary Johnson  |        2 | 2023-06-15 |        85.25
 John Smith    |        3 | 2023-07-01 |       200.00
 Robert Brown  |        4 | 2023-07-05 |        65.75
 Mary Johnson  |        5 | 2023-07-10 |       150.30
               |        6 | 2023-07-15 |        50.00
(6 rows)


dbmsprac=#
dbmsprac=# -- Cross Join
dbmsprac=# SELECT c.name, p.name AS product_name
dbmsprac-# FROM customers c
dbmsprac-# CROSS JOIN products p
dbmsprac-# LIMIT 10;
     name      | product_name
---------------+--------------
 John Smith    | Laptop
 Mary Johnson  | Laptop
 Robert Brown  | Laptop
 Patricia Davis | Laptop
 Michael Miller | Laptop
 John Smith    | Smartphone
 Mary Johnson  | Smartphone
 Robert Brown  | Smartphone
 Patricia Davis | Smartphone
 Michael Miller | Smartphone
(10 rows)


dbmsprac=#
dbmsprac=# -- Self Join
dbmsprac=# SELECT e.name AS employee, m.name AS manager
dbmsprac-# FROM employees e
dbmsprac-# LEFT JOIN employees m ON e.manager_id = m.employee_id;
    employee    |   manager
----------------+--------------
 John Smith     |
 Mary Johnson   | John Smith
 Robert Brown   | John Smith
 Patricia Davis | Mary Johnson
 Michael Miller | Mary Johnson
(5 rows)


dbmsprac=#
dbmsprac=# -- Full Outer Join
dbmsprac=# SELECT c.name, o.order_id, o.order_date, o.total_amount
dbmsprac-# FROM customers c
dbmsprac-# FULL OUTER JOIN orders o ON c.customer_id = o.customer_id;
     name      | order_id | order_date | total_amount
---------------+----------+------------+--------------
 John Smith    |        1 | 2023-06-10 |       120.50
 Mary Johnson  |        2 | 2023-06-15 |        85.25
 John Smith    |        3 | 2023-07-01 |       200.00
 Robert Brown  |        4 | 2023-07-05 |        65.75
 Mary Johnson  |        5 | 2023-07-10 |       150.30
               |        6 | 2023-07-15 |        50.00
 Michael Miller |          |            |
 Patricia Davis |          |            |
(8 rows)
```

```
SQL Shell (psql)


dbmsprac=#
dbmsprac=# -- All Aggregate Functions
dbmsprac=# SELECT
dbmsprac-#     COUNT(*) AS total_orders,
dbmsprac-#     SUM(total_amount) AS total_sales,
dbmsprac-#     AVG(total_amount) AS average_order_value,
dbmsprac-#     MIN(total_amount) AS min_order_value,
dbmsprac-#     MAX(total_amount) AS max_order_value,
dbmsprac-#     STDDEV(total_amount) AS standard_deviation,
dbmsprac-#     VARIANCE(total_amount) AS variance
dbmsprac-# FROM orders;
 total_orders | total_sales | average_order_value  | min_order_value | max_order_value | standard_deviation  |       variance
--------------+-------------+----------------------+-----------------+-----------------+---------------------+-----------------------
            6 |      671.80 | 111.9666666666666667 |           50.00 |          200.00 | 56.5362862829410353 | 3196.3516666666666667
(1 row)


dbmsprac=#
dbmsprac=# -- Group By
dbmsprac=# SELECT customer_id, COUNT(*) AS order_count, SUM(total_amount) AS total_spent
dbmsprac-# FROM orders
dbmsprac-# WHERE customer_id IS NOT NULL
dbmsprac-# GROUP BY customer_id;
 customer_id | order_count | total_spent
-------------+-------------+-------------
           3 |           1 |       65.75
           2 |           2 |      235.55
           1 |           2 |      320.50
(3 rows)


dbmsprac=#
dbmsprac=# -- Having
dbmsprac=# SELECT customer_id, COUNT(*) AS order_count, SUM(total_amount) AS total_spent
dbmsprac-# FROM orders
dbmsprac-# WHERE customer_id IS NOT NULL
dbmsprac-# GROUP BY customer_id
dbmsprac-# HAVING COUNT(*) > 1;
 customer_id | order_count | total_spent
-------------+-------------+-------------
           2 |           2 |      235.55
           1 |           2 |      320.50
(2 rows)


dbmsprac=#
dbmsprac=# -- Set Operators - UNION
dbmsprac=# SELECT name FROM customers
dbmsprac-# UNION
dbmsprac-# SELECT name FROM employees;
     name
---------------
 Michael Miller
 Mary Johnson
 John Smith
 Robert Brown
 Patricia Davis
(5 rows)


dbmsprac=#
```

Deccan Education Society's (DES)
Pune University, Pune
School of Engineering and Technology
Department of Computer Engineering and Technology
Program: B. Tech in Computer Science and Engineering



```
SQL Shell (psql)

dbmsprac=#
dbmsprac=# -- Set Operators - UNION ALL
dbmsprac=# SELECT name FROM customers
dbmsprac-# UNION ALL
dbmsprac-# SELECT name FROM employees;
      name
----------------
 John Smith
 Mary Johnson
 Robert Brown
 Patricia Davis
 Michael Miller
 John Smith
 Mary Johnson
 Robert Brown
 Patricia Davis
 Michael Miller
(10 rows)


dbmsprac=#
dbmsprac=# -- Set Operators - INTERSECT
dbmsprac=# SELECT city FROM customers
dbmsprac-# INTERSECT
dbmsprac-# SELECT name FROM products;
 city
------
(0 rows)

dbmsprac=#
dbmsprac=# -- Set Operators - EXCEPT
dbmsprac=# SELECT name FROM customers
dbmsprac-# EXCEPT
dbmsprac-# SELECT name FROM employees;
 name
------
(0 rows)


dbmsprac=#
dbmsprac=# -- Date functions
dbmsprac=# SELECT
dbmsprac-#     order_date,
dbmsprac-#     EXTRACT(YEAR FROM order_date) AS year,
dbmsprac-#     EXTRACT(MONTH FROM order_date) AS month,
dbmsprac-#     EXTRACT(DAY FROM order_date) AS day,
dbmsprac-#     CURRENT_DATE AS today,
dbmsprac-#     order_date + INTERVAL '30 days' AS due_date,
dbmsprac-#     AGE(CURRENT_DATE, order_date) AS order_age
dbmsprac-# FROM orders;
 order_date | year | month | day |   today    |      due_date       |       order_age
------------+------+-------+-----+------------+---------------------+------------------------
 2023-06-10 | 2023 |     6 |  10 | 2025-03-17 | 2023-07-10 00:00:00 | 1 year 9 mons 7 days
 2023-06-15 | 2023 |     6 |  15 | 2025-03-17 | 2023-07-15 00:00:00 | 1 year 9 mons 2 days
 2023-07-01 | 2023 |     7 |   1 | 2025-03-17 | 2023-07-31 00:00:00 | 1 year 8 mons 16 days
 2023-07-05 | 2023 |     7 |   5 | 2025-03-17 | 2023-08-04 00:00:00 | 1 year 8 mons 12 days
 2023-07-10 | 2023 |     7 |  10 | 2025-03-17 | 2023-08-09 00:00:00 | 1 year 8 mons 7 days
 2023-07-15 | 2023 |     7 |  15 | 2025-03-17 | 2023-08-14 00:00:00 | 1 year 8 mons 2 days
(6 rows)
```



```
SQL Shell (psql)

dbmsprac=#
dbmsprac=# -- Time functions
dbmsprac=# SELECT
dbmsprac-#     CURRENT_TIME AS current_time,
dbmsprac-#     CURRENT_TIMESTAMP AS current_timestamp,
dbmsprac-#     LOCALTIME AS local_time,
dbmsprac-#     LOCALTIMESTAMP AS local_timestamp,
dbmsprac-#     EXTRACT(HOUR FROM CURRENT_TIME) AS current_hour;
    current_time     |       current_timestamp       |  local_time  |     local_timestamp     | current_hour
---------------------+-------------------------------+--------------+-------------------------+--------------
 19:13:55.207464+05:30 | 2025-03-17 19:13:55.207464+05:30 | 19:13:55.207464 | 2025-03-17 19:13:55.207464 |           19
(1 row)


dbmsprac=#
dbmsprac=# -- Mathematical functions
dbmsprac=# SELECT
dbmsprac-#     price,
dbmsprac-#     ROUND(price, 1) AS rounded_price,
dbmsprac-#     CEIL(price) AS ceiling_price,
dbmsprac-#     FLOOR(price) AS floor_price,
dbmsprac-#     ABS(-price) AS absolute_price,
dbmsprac-#     POWER(price, 2) AS price_squared,
dbmsprac-#     SQRT(price) AS square_root,
dbmsprac-#     LOG(price) AS natural_log,
dbmsprac-#     RANDOM() AS random_number
dbmsprac-# FROM products;
  price  | rounded_price | ceiling_price | floor_price | absolute_price |   price_squared    |    square_root     |    natural_log     |    random_number
---------+---------------+---------------+-------------+----------------+--------------------+--------------------+--------------------+---------------------
 1200.00 |        1200.0 |          1200 |        1200 |        1200.00 | 1440000.0000000000 | 34.641016151377546 |  7.0724334967374428 | 0.07243349673741428
  800.00 |         800.0 |           800 |         800 |         800.00 |  640000.0000000000 | 28.284271247461901 |  2.9030899869919436 | 0.10039524611866724
  150.00 |         150.0 |           150 |         150 |         150.00 |   22500.0000000000 | 12.247448713915890 |  2.1760912590556812 |  0.6511375544672118
   85.00 |          85.0 |            85 |          85 |          85.00 |    7225.0000000000 |  9.2195444572928887 |  1.9294189257142927 |  0.6739132304765165
   65.00 |          65.0 |            65 |          65 |          65.00 |    4225.0000000000 |  8.0622577482985550 |  1.8129133566428556 | 0.44349028264960544
(5 rows)


dbmsprac=#
dbmsprac=# -- String functions
dbmsprac=# SELECT
dbmsprac-#     name,
dbmsprac-#     UPPER(name) AS upper_case,
dbmsprac-#     LOWER(name) AS lower_case,
dbmsprac-#     LENGTH(name) AS name_length,
dbmsprac-#     SUBSTRING(name, 1, 3) AS first_three_chars,
dbmsprac-#     REPLACE(name, 'a', 'A') AS replaced,
dbmsprac-#     CONCAT(name, ' - ', category) AS product_info,
dbmsprac-#     TRIM(BOTH ' ' FROM name) AS trimmed_name
dbmsprac-# FROM products;
     name     |  upper_case  |  lower_case  | name_length | first_three_chars |   replaced   |        product_info        | trimmed_name
--------------+--------------+--------------+-------------+-------------------+--------------+----------------------------+--------------
 Laptop       | LAPTOP       | laptop       |           6 | Lap               | LAptop       | Laptop - Electronics       | Laptop
 Smartphone   | SMARTPHONE   | smartphone   |          10 | Sma               | SmArtphone   | Smartphone - Electronics   | Smartphone
 Headphones   | HEADPHONES   | headphones   |          10 | Hea               | HeAdphones   | Headphones - Electronics   | Headphones
 Coffee Maker | COFFEE MAKER | coffee maker |          12 | Cof               | Coffee MAker | Coffee Maker - Kitchen     | Coffee Maker
 Blender      | BLENDER      | blender      |           7 | Ble               | Blender      | Blender - Kitchen          | Blender
(5 rows)


dbmsprac=#
```

FAQs:

1. **Can aggregate functions be used without the GROUP BY clause?**
   Yes, aggregate functions can be used without GROUP BY. When used without GROUP BY, they apply to the entire result set and return a single row. For example:

```
1. SELECT COUNT(*), AVG(total_amount), SUM(total_amount) FROM orders;
```

2. What is the difference between COUNT(*) and COUNT(column_name)?

COUNT(*) counts all rows in the table, including rows with NULL values
COUNT(column_name) counts only non-NULL values in the specified column For example, if a column has NULL values, COUNT(column_name) will return a smaller number than COUNT(*).

3. Can we use multiple aggregate functions in a single query? Yes, you can use multiple aggregate functions in a single query. For example:

```
1. SELECT COUNT(*) AS total_orders,
2.    SUM(total_amount) AS total_sales,
3.  AVG(total_amount) AS average_order
4. FROM orders;
5.
```

4. Can we use WHERE and HAVING together in a query with aggregate functions? Yes, you can use both WHERE and HAVING together. The key difference is:

WHERE filters rows before they're grouped

HAVING filters groups after they're formed For example:

```
1. SELECT customer_id, COUNT(*) AS order_count
2. FROM orders
3. WHERE order_date > '2023-01-01'
4. GROUP BY customer_id
5. HAVING COUNT(*) > 5;
6.
```

5. What is the difference between INNER JOIN and LEFT JOIN?

INNER JOIN: Returns only rows that have matching values in both tables

LEFT JOIN: Returns all rows from the left table and matching rows from the right table. If no match is found, NULL values are returned for right table columns

6. What is the difference between ON and USING in joins?

ON: Allows you to specify any join condition (with any columns)

```
1. -- Using ON
2. SELECT * FROM orders o JOIN customers c ON o.customer_id = c.customer_id;
3.
4.
```

USING: Simplifies the syntax when joining tables on columns with the same name For example

```
-- 1. Using USING (works only when column names are identical)
2. SELECT * FROM orders JOIN customers USING (customer_id);
3.
```

7. Can we use aggregate functions with joins?

Yes, you can use aggregate functions with joins. This is often used to aggregate data from related tables. For example:

```
1. SELECT c.name, COUNT(o.order_id) AS order_count, SUM(o.total_amount) AS total_spent
2. FROM customers c
3. LEFT JOIN orders o ON c.customer_id = o.customer_id
4. GROUP BY c.name;
5.
```

This query joins customers and orders tables, then aggregates the order data for each customer.