# SQL Part 2 Assignment

**Assignment 1**

**Name: Akshada Baad**

**Batch - CPPE**

**Assignment 1: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.**

let's start by creating the customer's table, inserting some sample data, and then writing the

required SELECT queries.

Step 1: Create the customer's Table:

First, create the table:

```
CREATE TABLE customers (

customer_id INT PRIMARY KEY,

customer_name VARCHAR(100),

email VARCHAR(100),

phone VARCHAR(20),

address VARCHAR(255),

city VARCHAR(100),

postal_code VARCHAR(20),

country VARCHAR(100)

);
```

Step 2: Insert Sample Values into the customer's Table:

Now, insert some sample data into the table:

```
INSERT INTO customers (customer_id, customer_name, email, phone, address, city,

postal_code, country)

VALUES
```

(1, 'John Doe', 'john.doe@example.com', '555-1234', '123 Elm St', 'New York', '10001',

'USA'),

(2, 'Jane Smith', 'jane.smith@example.com', '555-5678', '456 Oak St', 'Los Angeles', '90001',

'USA'),

(3, 'Alice Johnson', 'alice.johnson@example.com', '555-8765', '789 Pine St', 'New York',

'10002', 'USA'),

(4, 'Bob Brown', 'bob.brown@example.com', '555-4321', '321 Maple St', 'Chicago', '60601',

'USA');

Step 3: Query to Retrieve All Columns from the Customer Table:

To retrieve all columns from the customer table:

SELECT * FROM customers;

Step 4: Query to Retrieve Customer Name and Email for Customers in a Specific City:

Assuming the specific city is 'New York', the modified query to retrieve only the

customer name and email address for customers in 'New York' is:

SELECT customer_name, email

FROM customers

WHERE city = 'New York';

## Assignment 2

**Assignment 2: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.**

Let's go through the process of creating the orders and customers tables, inserting values,

and then crafting the required queries using INNER JOIN and LEFT JOIN.

Step 1: Create the customer's Table:

```
CREATE TABLE customers (
customer_id INT PRIMARY KEY,
customer_name VARCHAR(100),
email VARCHAR(100),
phone VARCHAR(20),
address VARCHAR(255),
city VARCHAR(100),
region VARCHAR(100), -- (Added region column)
postal_code VARCHAR(20),
country VARCHAR(100)
);
```

Step 2: Create the orders Table:

```
CREATE TABLE orders (
order_id INT PRIMARY KEY,
order_date DATE,
customer_id INT,
amount DECIMAL(10, 2),
FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Step 3: Insert Sample Values into the customer's Table:

```
INSERT INTO customers (customer_id, customer_name, email, phone, address, city, region,
postal_code, country)
VALUES
(1, 'John Doe', 'john.doe@example.com', '555-1234', '123 Elm St', 'New York', 'East', '10001',
```

'USA'),

(2, 'Jane Smith', 'jane.smith@example.com', '555-5678', '456 Oak St', 'Los Angeles', 'West',

'90001', 'USA'),

(3, 'Alice Johnson', 'alice.johnson@example.com', '555-8765', '789 Pine St', 'New York',

'East', '10002', 'USA'),

(4, 'Bob Brown', 'bob.brown@example.com', '555-4321', '321 Maple St', 'Chicago', 'Midwest',

'60601', 'USA');

Step 4: Insert Sample Values into the orders Table:

```
INSERT INTO orders (order_id, order_date, customer_id, amount)
VALUES
(1, '2023-05-01', 1, 150.00),
(2, '2023-05-03', 2, 200.00),
(3, '2023-05-05', 3, 300.00);
```

Step 5: Query Using INNER JOIN to Combine orders and customers Tables for

Customers in a Specified Region:

Assume the specified region is 'East'. The query using INNER JOIN is:

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.amount
FROM customers c
INNER JOIN o.orders ON c.customer_id = o.customer_id
WHERE c.region = 'East';
```

Step 6: Query Using LEFT JOIN to Display All Customers Including Those Without

Orders :

The query using LEFT JOIN is:

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.amount
```

```
FROM customers c

LEFT JOIN orders o ON c.customer_id = o.customer_id;
```

## Assignment 3

**Assignment 3: Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.**

) Let's proceed with creating the customers and orders tables, inserting the values, and then crafting the required queries using a subquery and a UNION.

Step 1: Create the customer's Table

```
CREATE TABLE customers (

customer_id INT PRIMARY KEY,

customer_name VARCHAR(100),

email VARCHAR(100),

phone VARCHAR(20),

address VARCHAR(255),

city VARCHAR(100),

region VARCHAR(100),

postal_code VARCHAR(20),

country VARCHAR(100)

);
```

Step 2: Create the orders Table

```
CREATE TABLE orders (

order_id INT PRIMARY KEY,

order_date DATE,
```

```
customer_id INT,
amount DECIMAL(10, 2),
FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Step 3: Insert Sample Values into the customer's Table

```
INSERT INTO customers (customer_id, customer_name, email, phone, address, city, region,
postal_code, country)
VALUES
(1, 'John Doe', 'john.doe@example.com', '555-1234', '123 Elm St', 'New York', 'East', '10001',
'USA'),
(2, 'Jane Smith', 'jane.smith@example.com', '555-5678', '456 Oak St', 'Los Angeles', 'West',
'90001', 'USA'),
(3, 'Alice Johnson', 'alice.johnson@example.com', '555-8765', '789 Pine St', 'New York',
'East', '10002', 'USA'),
(4, 'Bob Brown', 'bob.brown@example.com', '555-4321', '321 Maple St', 'Chicago', 'Midwest',
'60601', 'USA');
```

Step 4: Insert Sample Values into the orders Table

```
INSERT INTO orders (order_id, order_date, customer_id, amount)
VALUES
(1, '2023-05-01', 1, 150.00),
(2, '2023-05-03', 2, 200.00),
(3, '2023-05-05', 3, 300.00),
(4, '2023-05-07', 1, 350.00),
(5, '2023-05-09', 4, 400.00);
```

Step 5: Query Using a Subquery to Find Customers Who Have Placed Orders Above the Average Order Value

First, we find the average order value:

```
SELECT AVG(amount) AS avg_order_value FROM orders;
```

Next, we use this average order value in a subquery to find customers who have placed orders above this value:

```
SELECT customer_id, customer_name
FROM customers
WHERE customer_id IN (
SELECT DISTINCT customer_id
FROM orders
WHERE amount > (SELECT AVG(amount) FROM orders)
);
```

Step 6: UNION Query to Combine Two SELECT Statements with the Same Number of Columns

For the UNION query, we need two SELECT statements with the same number of columns and compatible data types. Here's an example combining customers from 'East' and 'West' regions:

```
SELECT customer_id, customer_name, region
FROM customers
WHERE region = 'East'
UNION
SELECT customer_id, customer_name, region
FROM customers
```

WHERE region = 'West';

## Assignment 4

**Assignment 4: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.**

Let's break down the process step by step.

Step-1: Create Tables:

We'll start by creating the necessary tables, orders, and products. These tables will store information about orders and products respectively.

```
CREATE TABLE orders (
order_id INT AUTO_INCREMENT PRIMARY KEY,
customer_id INT,
order_date DATE,
total_amount DECIMAL(10,2)
);
CREATE TABLE products (
product_id INT AUTO_INCREMENT PRIMARY KEY,
product_name VARCHAR(100),
price DECIMAL(10,2),
quantity INT
);
```

The orders table has columns for order_id (unique identifier for each order), customer_id (to associate orders with customers), order_date (date of the order), and total_amount (total cost of the order).

The products table has columns for product_id (unique identifier for each product), product_name, price, and quantity (available quantity of the product).

Insert Data:

Step-2: Now, let's insert some sample data into the tables. This will allow us to demonstrate the transaction.

```
INSERT INTO products (product_name, price, quantity)
VALUES ('Product A', 10.99, 100),
('Product B', 15.99, 50),
('Product C', 20.49, 75);
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (101, '2024-05-20', 50.00),
(102, '2024-05-21', 100.00);
```

We insert some sample products into the products table with their respective prices and quantities.

We insert sample orders into the orders table, associating them with customers and providing order dates and total amounts.

Step-3:Transaction:

Now, let's perform the transaction as specified:

Begin a transaction

```
BEGIN;
```

Insert a new record into the 'orders' table

```
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES (103, '2024-05-22', 75.00);
```

Commit the transaction

COMMIT;

Update the 'products' table

INSERT INTO price_changes (product_id, old_price, new_price, change_date)

VALUES (1, 10.99, 12.99, NOW());

Rollback the transaction

ROLLBACK;

Explanation:

● We begin a transaction using BEGIN.

● We insert a new order into the orders table.

● We commit the transaction using COMMIT, which saves the changes made so far.

● We then update the quantity of a product in the products table.

● Finally, we roll back the transaction using ROLLBACK, which undoes all the changes made within the transaction. This means that the insertion of the new order is undone, leaving the database in the state it was before the transaction began.

## Assignment 5

**Assignment 5: Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, roll back to the second SAVEPOINT, and COMMIT the overall transaction.**

let's create the orders table and then perform a series of INSERTs into it within a transaction, setting a SAVEPOINT after each, and finally rolling back to the second SAVEPOINT before committing the overall transaction.

First, we'll create the orders table:

CREATE TABLE orders (

order_id INT AUTO_INCREMENT PRIMARY KEY,

customer_id INT,

order_date DATE,

total_amount DECIMAL(10,2)

);

Now, let's insert some sample data into the orders table:

INSERT INTO orders (customer_id, order_date, total_amount)

VALUES (101, '2024-05-20', 50.00),

(102, '2024-05-21', 100.00),

(103, '2024-05-22', 75.00);

Now, let's perform the series of INSERTs within a transaction, setting SAVEPOINTs after each one, and then rolling back to the second SAVEPOINT before committing the overall transaction:

Begin a transaction

BEGIN;

Insert the first record into the 'orders' table

INSERT INTO orders (customer_id, order_date, total_amount)

VALUES (104, '2024-05-23', 80.00);

Set a SAVEPOINT after the first INSERT

SAVEPOINT savepoint1;

Insert the second record into the 'orders' table

INSERT INTO orders (customer_id, order_date, total_amount)

VALUES (105, '2024-05-24', 90.00);

Set a SAVEPOINT after the second INSERT

SAVEPOINT savepoint2;

Insert the third record into the 'orders' table

INSERT INTO orders (customer_id, order_date, total_amount)

VALUES (106, '2024-05-25', 120.00);

Rollback to the second SAVEPOINT

ROLLBACK TO SAVEPOINT savepoint2;

Commit the overall transaction

COMMIT;

Explanation:

● We begin a transaction using BEGIN.

● We perform a series of INSERT operations in the orders table.

● After each INSERT, we set a SAVEPOINT using SAVEPOINT.

● We then roll back to the second SAVEPOINT using ROLLBACK TO SAVEPOINT, effectively undoing the INSERT of the third record.

● Finally, we commit the overall transaction using COMMIT, which applies all changes made up to that point (i.e., the first and second INSERTs).

## Assignment 6

**Assignment 6: Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.**

Report on the Use of Transaction Logs for Data Recovery

Introduction:

Transaction logs are essential components of database management systems (DBMS) that record all transactions executed within a database. These logs serve as a crucial tool for data recovery in the event of unexpected shutdowns, system failures, or other disruptions.

This report aims to explore the significance of transaction logs in data recovery and present a hypothetical scenario demonstrating their instrumental role.

The Importance of Transaction Logs for Data Recovery:

Transaction logs function as a chronological record of database modifications, capturing both committed and uncommitted transactions. Transaction logs play a vital role in restoring the database to a consistent state in a system failure or unexpected shutdown. DBMS can recover lost or corrupted data by replaying the recorded transactions, ensuring data integrity and minimizing downtime.

Scenario:

Consider a multinational retail corporation operating a centralized inventory management system to track its vast array of products across numerous stores worldwide. The corporation relies heavily on its database to manage inventory, process transactions, and generate reports for decision-making.

One day, during peak business hours, the database server experienced a sudden power outage due to a fault in the electrical supply system. As a result, the database abruptly shuts down, leaving transactions unfinished and potentially jeopardizing critical data integrity.

The system administrator initiates the data recovery process upon rebooting the database server. Fortunately, due to the presence of robust transaction logging mechanisms, the administrator can leverage the transaction logs to restore the database to its last consistent state before the unexpected shutdown.

The transaction logs contain a comprehensive record of all transactions executed up to the moment of the outage, including details such as transaction IDs, timestamps, and the changes made to the database. Using this information, the DBMS systematically replays the transactions, applying committed changes and rolling back any incomplete transactions to

maintain data consistency.

Through the effective utilization of transaction logs, the retail corporation successfully restores its inventory database, ensuring that no transactions are lost, and data integrity is preserved. The swift recovery minimizes disruption to operations, allowing the corporation to resume normal business activities promptly.