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.

Customers table creation

```
mysql> create table customers(
    -> name  varchar(10),
    -> email varchar(20),
    -> city varchar(10));
Query OK, 0 rows affected (0.05 sec)
```

Insertion of values

```
mysql> insert into customers values("karthik","kartik@gmail.com", "Bangalore");
Query OK, 1 row affected (0.02 sec)

mysql> insert into customers values("kumar","kumar@gmail.com", "Hyd");
Query OK, 1 row affected (0.05 sec)

mysql> insert into customers values("ananya","ananya@gmail.com", "Mumbai");
Query OK, 1 row affected (0.01 sec)
```

Projection of only the customer name and email address in a specific city

```
mysql> select name,email
    -> from customers
    -> where city="bangalore";
+---------+------------------+
| name    | email            |
+---------+------------------+
| karthik | kartik@gmail.com |
+---------+------------------+
1 row in set (0.00 sec)

mysql>
```

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.

Creation of tables ‘orders’ and ‘customers’:

```
mysql> CREATE TABLE customers (
    ->     customer_id INT PRIMARY KEY,
    ->     customer_name VARCHAR(100),
    ->     region VARCHAR(50),
    ->     email_address VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE orders (
    ->     order_id INT PRIMARY KEY,
    ->     customer_id INT,
    ->     order_date DATE,
    ->     total_amount DECIMAL(10, 2),
    ->     FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql>
```

Insertion of tuples:

```
mysql> INSERT INTO customers (customer_id, customer_name, region, email_address)
    -> VALUES
    -> (1, 'John Doe', 'North America', 'john.doe@example.com'),
    -> (2, 'Jane Smith', 'Europe', 'jane.smith@example.com'),
    -> (3, 'Alice Johnson', 'Asia', 'alice.johnson@example.com'),
    -> (4, 'Bob Johnson', 'North America', 'bob.johnson@example.com'),
    -> (5, 'Emily Brown', 'Europe', 'emily.brown@example.com');
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
    -> VALUES
    -> (101, 1, '2024-05-25', 100.00),
    -> (102, 1, '2024-05-26', 150.00),
    -> (103, 2, '2024-05-27', 200.00),
    -> (104, 4, '2024-05-27', 250.00),
    -> (105, 5, '2024-05-28', 300.00);
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Insertion of tuples without order

```
mysql> INSERT INTO customers (customer_id, customer_name, region, email_address)
    -> VALUES
    -> (10, 'Michael Johnson', 'North America', 'michael.johnson@example.com');
Query OK, 1 row affected (0.02 sec)
```

Query to display all customers including those without orders:

```
mysql> SELECT c.customer_id, c.customer_name, c.region, c.email_address, o.order_id, o.order_date, o.total_amount
    -> FROM customers c
    -> LEFT JOIN orders o ON c.customer_id = o.customer_id
    -> WHERE c.region = 'North America' AND o.customer_id IS NULL;
+-------------+-----------------+---------------+---------------------------+----------+------------+--------------+
| customer_id | customer_name   | region        | email_address             | order_id | order_date | total_amount |
+-------------+-----------------+---------------+---------------------------+----------+------------+--------------+
|          10 | Michael Johnson | North America | michael.johnson@example.com |     NULL | NULL       |         NULL |
+-------------+-----------------+---------------+---------------------------+----------+------------+--------------+
1 row in set (0.00 sec)
```

Query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region:

```
mysql> SELECT c.customer_id, c.customer_name, c.region, c.email_address, o.order_id, o.order_date, o.total_amount
    -> FROM customers c
    -> INNER JOIN orders o ON c.customer_id = o.customer_id AND c.region = 'North America'
    -> ;
+-------------+---------------+---------------+-----------------------+----------+------------+--------------+
| customer_id | customer_name | region        | email_address         | order_id | order_date | total_amount |
+-------------+---------------+---------------+-----------------------+----------+------------+--------------+
|           1 | John Doe      | North America | john.doe@example.com  |      101 | 2024-05-25 |       100.00 |
|           1 | John Doe      | North America | john.doe@example.com  |      102 | 2024-05-26 |       150.00 |
|           4 | Bob Johnson   | North America | bob.johnson@example.com |      104 | 2024-05-27 |       250.00 |
+-------------+---------------+---------------+-----------------------+----------+------------+--------------+
3 rows in set (0.00 sec)
```

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.

1) Subquery to find customers who have placed orders above the average order value

```
mysql> SELECT *
    -> FROM customers
    -> WHERE customer_id IN (
    ->     SELECT customer_id
    ->     FROM orders
    ->     GROUP BY customer_id
    ->     HAVING AVG(total_amount) > (SELECT AVG(total_amount) FROM orders)
    -> );
+-------------+---------------+---------------+-----------------------+
| customer_id | customer_name | region        | email_address         |
+-------------+---------------+---------------+-----------------------+
|           4 | Bob Johnson   | North America | bob.johnson@example.com |
|           5 | Emily Brown   | Europe        | emily.brown@example.com |
+-------------+---------------+---------------+-----------------------+
2 rows in set (0.01 sec)
```

2) UNION query to combine two SELECT statements

```
mysql> select order_id,total_amount from orders UNION select customer_name,region from customers;
+-----------------+---------------+
| order_id        | total_amount  |
+-----------------+---------------+
| 101             | 100.00        |
| 102             | 150.00        |
| 103             | 200.00        |
| 104             | 250.00        |
| 105             | 300.00        |
| John Doe        | North America |
| Jane Smith      | Europe        |
| Alice Johnson   | Asia          |
| Bob Johnson     | North America |
| Emily Brown     | Europe        |
| Michael Johnson | North America |
+-----------------+---------------+
11 rows in set (0.00 sec)
```

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.

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
    -> VALUES (106, 3, '2024-05-29', 180.00);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.03 sec)

mysql> UPDATE orders
    -> SET total_amount = total_amount + 10
    -> WHERE order_id = 106;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM orders WHERE order_id = 106;
+----------+-------------+------------+--------------+
| order_id | customer_id | order_date | total_amount |
+----------+-------------+------------+--------------+
|      106 |           3 | 2024-05-29 |       190.00 |
+----------+-------------+------------+--------------+
1 row in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM orders WHERE order_id = 106;
+----------+-------------+------------+--------------+
| order_id | customer_id | order_date | total_amount |
+----------+-------------+------------+--------------+
|      106 |           3 | 2024-05-29 |       190.00 |
+----------+-------------+------------+--------------+
1 row in set (0.00 sec)

mysql> UPDATE orders
    -> SET total_amount = 180
    -> WHERE order_id = 106;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM orders WHERE order_id = 106;
+----------+-------------+------------+--------------+
| order_id | customer_id | order_date | total_amount |
+----------+-------------+------------+--------------+
|      106 |           3 | 2024-05-29 |       180.00 |
+----------+-------------+------------+--------------+
1 row in set (0.00 sec)
```

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

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
    -> VALUES (107, 4, '2024-05-30', 200.00);
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT savepoint1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
    -> VALUES (108, 5, '2024-05-31', 250.00);
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT savepoint2;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders (order_id, customer_id, order_date, total_amount)
    -> VALUES (105, 3, '2024-05-29', 300.00);
Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK TO SAVEPOINT savepoint2;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from orders;
+----------+-------------+------------+--------------+
| order_id | customer_id | order_date | total_amount |
+----------+-------------+------------+--------------+
|      107 |           4 | 2024-05-30 |       200.00 |
|      108 |           5 | 2024-05-31 |       250.00 |
+----------+-------------+------------+--------------+
2 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.03 sec)

mysql>
```

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.

Transaction logs play a crucial role in ensuring data integrity and facilitating recovery in database management systems. They record all changes made to the database, providing a detailed trail of transactions that can be used for recovery purposes in case of system failures or data corruption.

Transaction Logs and Their Importance:

- Every SQL Server database maintains a transaction log that records all transactions and modifications made by each transaction.
- The transaction log is a critical component of the database. In case of system failures, it plays a pivotal role in bringing the database back to a consistent state.

Operations Supported by the Transaction Log:

Individual Transaction Recovery:

If an application issues a ROLLBACK statement or if there's an error (e.g., loss of communication with a client), the log records are used to roll back incomplete transaction modifications.

Recovery When SQL Server Starts:

After a server failure, databases may be left with uncommitted modifications in memory. SQL Server runs recovery for each database during startup.

Modifications recorded in the log but not yet written to data files are rolled forward, and incomplete transactions are rolled back to preserve database integrity.

Rolling Forward to Point of Failure:

After hardware or disk failure, restore the database to the point of failure.

Restore the last full database backup, the last differential database backup, and subsequent transaction log backups.

The Database Engine reapplies modifications from the log to roll forward transactions.

Supporting High Availability and Disaster Recovery:

Transaction logs are essential for solutions like Always On availability groups, database mirroring, and log shipping.

Hypothetical Scenario: Imagine a library system where users borrow and return books. Suppose an unexpected shutdown occurs during a busy day:

Before Shutdown:

- Users borrow books, and the system records these transactions in the database.
- The transaction log captures these changes.

Shutdown Event:

- Suddenly, the server crashes due to a power outage.
- Uncommitted changes are still in memory, and some transactions are incomplete.

Recovery Process:

- During startup, SQL Server uses the transaction log to roll forward committed changes.
- It also identifies incomplete transactions and rolls them back.
- The database is brought to a consistent state.

Post-Recovery:

- Users can continue borrowing and returning books without data loss.