Tasks 1: Database Design:

1. Create the database named "HMBank"

```
CREATE DATABASE HMBank;

USE HMBank;
```

2. Define the schema for the Customers, Accounts, and Transactions tables based on the provided schema.

```
CREATE TABLE Customers (
    ->      customer_id INT PRIMARY KEY NOT NULL,
    ->      first_name TEXT,
    ->      last_name TEXT,
    ->      DOB DATE,
    ->      email TEXT,
    ->      phone_number VARCHAR(20)
    -> );


CREATE TABLE Accounts (
    ->      account_id INT PRIMARY KEY NOT NULL,
    ->      customer_id INT,
    ->      account_type TEXT,
    ->      balance DECIMAL(10, 2),
    ->      FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
    -> );

CREATE TABLE Transactions (
    ->      transaction_id INT PRIMARY KEY NOT NULL,
```

```
-> account_id INT,
-> transaction_type TEXT,
-> amount DECIMAL(10, 2),
-> transaction_date TIMESTAMP,
-> FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
-> );
```

3. Create an ERD (Entity Relationship Diagram) for the database.
4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

   DONE BELOW.
5. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.

   • Customers

   CREATE TABLE Customers (

```
-> customer_id INT PRIMARY KEY NOT NULL,
-> first_name TEXT,
-> last_name TEXT,
-> DOB DATE,
-> email TEXT,
-> phone_number VARCHAR(20)
-> );
```

   • Accounts

   CREATE TABLE Accounts (

```
-> account_id INT PRIMARY KEY NOT NULL,
-> customer_id INT,
-> account_type TEXT,
-> balance DECIMAL(10, 2),
```

-> FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ->);

• Transactions

CREATE TABLE Transactions (

-> transaction_id INT PRIMARY KEY NOT NULL,

-> account_id INT,

-> transaction_type TEXT,

-> amount DECIMAL(10, 2),

-> transaction_date TIMESTAMP,

-> FOREIGN KEY (account_id) REFERENCES Accounts(account_id)

->);

Tasks 2: Select, Where, Between, AND, LIKE:

1. Insert at least 10 sample records into each of the following tables.

   • Customers

   INSERT INTO Customers (customer_id, first_name, last_name, DOB, email, phone_number)

   -> VALUES

   -> (1, 'John', 'Doe', '1990-01-15', 'john.doe@email.com', '123-456-7890'),

   -> (2, 'Jane', 'Smith', '1985-03-20', 'jane.smith@email.com', '987-654-3210'),

   -> (3, 'Alice', 'Johnson', '1988-05-22', 'alice.johnson@email.com', '555-123-4567'),

   -> (4, 'Bob', 'Miller', '1995-12-10', 'bob.miller@email.com', '777-987-6543'),

   -> (5, 'Eva', 'Garcia', '1982-08-18', 'eva.garcia@email.com', '111-222-3333'),

   -> (6, 'Michael', 'Clark', '1976-03-05', 'michael.clark@email.com', '999-888-7777'),

   -> (7, 'Sophia', 'Chen', '1992-11-30', 'sophia.chen@email.com', '444-555-6666'),

   -> (8, 'Daniel', 'Brown', '1980-09-12', 'daniel.brown@email.com', '666-777-8888'),

   -> (9, 'Olivia', 'Lee', '1998-04-25', 'olivia.lee@email.com', '333-111-9999'),

   -> (10, 'David', 'Nguyen', '1987-07-15', 'david.nguyen@email.com', '222-444-5555');

   • Accounts

   INSERT INTO Accounts (account_id, customer_id, account_type, balance)

   -> VALUES

   -> (1, 1, 'savings', 5000.00),

   -> (2, 1, 'current', 1000.00),

   -> (3, 2, 'savings', 8000.00),

   -> (4, 2, 'current', 2500.00),

   -> (5, 3, 'savings', 12000.00),

-> (6, 4, 'zero_balance', 0.00),

-> (7, 5, 'current', 5000.00),

-> (8, 6, 'savings', 8000.00),

-> (9, 7, 'current', 3000.00),

-> (10, 8, 'savings', 6000.00);

• Transactions

INSERT INTO Transactions (transaction_id, account_id, transaction_type, amount, transaction_date)

-> VALUES

-> (1, 1, 'deposit', 1000.00, '2024-01-15 10:30:00'),

-> (2, 2, 'withdrawal', 500.00, '2024-01-16 12:45:00'),

-> (3, 3, 'deposit', 200.00, '2024-01-17 14:15:00'),

-> (4, 4, 'deposit', 2000.00, '2024-01-18 09:30:00'),

-> (5, 5, 'withdrawal', 800.00, '2024-01-19 11:15:00'),

-> (6, 6, 'deposit', 3000.00, '2024-01-20 14:45:00'),

-> (7, 7, 'withdrawal', 1200.00, '2024-01-21 16:30:00'),

-> (8, 8, 'deposit', 1500.00, '2024-01-22 10:00:00'),

-> (9, 9, 'withdrawal', 700.00, '2024-01-23 12:45:00'),

-> (10, 10, 'deposit', 1800.00, '2024-01-24 15:15:00');

2. Write SQL queries for the following tasks:

1. Write a SQL query to retrieve the name, account type and email of all customers.

SELECT

-> c.first_name || ' ' || c.last_name AS customer_name,

-> a.account_type,

```
->      c.email
-> FROM
->      Customers c
-> JOIN
->      Accounts a ON c.customer_id = a.customer_id;
```

2. Write a SQL query to list all transaction corresponding customer.

```
mysql> SELECT
    ->      t.transaction_id,
    ->      c.first_name || ' ' || c.last_name AS customer_name,
    ->      a.account_type,
    ->      t.transaction_type,
    ->      t.amount,
    ->      t.transaction_date
    -> FROM
    ->      Transactions t
    -> JOIN
    ->      Accounts a ON t.account_id = a.account_id
    -> JOIN
    ->      Customers c ON a.customer_id = c.customer_id;
```

3. Write a SQL query to increase the balance of a specific account by a certain amount.

```
UPDATE Accounts
    -> SET balance = balance + 500.00
    -> WHERE account_id = 101;
```

4. Write a SQL query to Combine first and last names of customers as a full_name.

```
SELECT
    ->      customer_id,
    ->      first_name || ' ' || last_name AS full_name
    -> FROM
    ->      Customers;
```

5. Write a SQL query to remove accounts with a balance of zero where the account type is savings.

```
DELETE FROM Accounts
-> WHERE balance = 0.00 AND account_type = 'savings';
```

6. Write a SQL query to Find customers living in a specific city.

```
SELECT
    ->      customer_id,
    ->      first_name,
    ->      last_name,
    ->      city
    -> FROM
    ->      Customers
    -> WHERE
    ->      city = 'JAMMU';
```

7. Write a SQL query to Get the account balance for a specific account.

```
> SELECT balance
    -> FROM Accounts
```

```
    -> WHERE account_id = YourSpecificAccountID;
ERROR 1054 (42S22): Unknown column 'YourSpecificAccountID' in 'where clause'
mysql> SELECT balance
    -> FROM Accounts
    -> WHERE account_id = 1;
```

8. Write a SQL query to List all current accounts with a balance greater than $1,000.

```
SELECT
    ->      account_id,
    ->      customer_id,
    ->      balance
    -> FROM
    ->      Accounts
    -> WHERE
    ->      account_type = 'current' AND balance > 1000.00;
```

9. Write a SQL query to Retrieve all transactions for a specific account.

```
SELECT
    ->      t.transaction_id,
    ->      t.account_id,
    ->      t.transaction_type,
    ->      t.amount,
    ->      t.transaction_date
    -> FROM
    ->      Transactions t
    -> JOIN
```

```
    ->      Accounts a ON t.account_id = a.account_id
-> WHERE
    ->      a.account_id = 2;
```

10. Write a SQL query to Calculate the interest accrued on savings accounts based on a given interest rate.

```
SELECT
    ->      account_id,
    ->      balance,
    ->      interest_rate,
    ->      balance * (interest_rate / 100) AS accrued_interest
-> FROM
    ->      Accounts
-> WHERE
    ->      account_type = 'savings';
```

11. Write a SQL query to Identify accounts where the balance is less than a specified overdraft limit.

```
SELECT
    ->      account_id,
    ->      customer_id,
    ->      balance
-> FROM
    ->      Accounts
```

```
-> WHERE
->      balance < 10000;
```

12. Write a SQL query to Find customers not living in a specific city.

```
SELECT
->      customer_id,
->      first_name,
->      last_name,
->      city
-> FROM
->      Customers c
-> WHERE
->      NOT EXISTS (
->          SELECT 1
->          FROM Customers c2
->          WHERE c2.customer_id = c.customer_id AND c2.city = 'CHENNAIL'
->      );
```

Tasks 3: Aggregate functions, Having, Order By, GroupBy and Joins:

1. Write a SQL query to Find the average account balance for all customers.

```
SELECT AVG(balance) AS average_balance
    -> FROM Accounts;
```

2. Write a SQL query to Retrieve the top 10 highest account balances.

```
SELECT
    ->      account_id,
    ->      customer_id,
    ->      balance
    -> FROM
    ->      Accounts
    -> ORDER BY
    ->      balance DESC
    -> LIMIT 10;
```

3. Write a SQL query to Calculate Total Deposits for All Customers in specific date.

```
SELECT
    ->      customer_id,
    ->      SUM(amount) AS total_deposits
    -> FROM
    ->      Transactions
    -> WHERE
    ->      transaction_type = 'deposit'
    ->      AND DATE(transaction_date) = 2024-01-15
```

```
    -> GROUP BY
    ->      customer_id;
```

4. Write a SQL query to Find the Oldest and Newest Customers.

OLDEST

```
SELECT
    ->      customer_id,
    ->      first_name,
    ->      last_name,
    ->      DOB
    -> FROM
    ->      Customers
    -> ORDER BY
    ->      DOB ASC
    -> LIMIT 1;
```

NEWEST

```
SELECT
    ->      customer_id,
    ->      first_name,
    ->      last_name,
    ->      DOB
    -> FROM
    ->      Customers
    -> ORDER BY
    ->      DOB DESC
    -> LIMIT 1;
```

5. Write a SQL query to Retrieve transaction details along with the account type

```
SELECT
    ->      t.transaction_id,
    ->      t.account_id,
    ->      a.account_type,
    ->      t.transaction_type,
    ->      t.amount,
    ->      t.transaction_date
    -> FROM
    ->      Transactions t
    -> JOIN
    ->      Accounts a ON t.account_id = a.account_id;
```

6. Write a SQL query to Get a list of customers along with their account details.

```
SELECT
    ->      c.customer_id,
    ->      c.first_name,
    ->      c.last_name,
    ->      c.DOB,
    ->      c.email,
    ->      c.phone_number,
    ->      a.account_id,
    ->      a.account_type,
    ->      a.balance,
    ->      t.transaction_id,
    ->      t.transaction_type,
    ->      t.amount,
```

```
    ->      t.transaction_date
    -> FROM
    ->      Customers c
    -> JOIN
    ->      Accounts a ON c.customer_id = a.customer_id
    -> LEFT JOIN
    ->      Transactions t ON a.account_id = t.account_id;
```

7. Write a SQL query to Retrieve transaction details along with customer information for a specific account.

1054 (42S22): Unknown column 'YourSpecificAccountID' in 'where clause'

```
mysql> SELECT
    ->      c.customer_id,
    ->      c.first_name,
    ->      c.last_name,
    ->      c.DOB,
    ->      c.email,
    ->      c.phone_number,
    ->      a.account_id,
    ->      a.account_type,
    ->      a.balance,
    ->      t.transaction_id,
    ->      t.transaction_type,
    ->      t.amount,
    ->      t.transaction_date
    -> FROM
    ->      Customers c
```

```
    -> JOIN
    ->      Accounts a ON c.customer_id = a.customer_id
    -> LEFT JOIN
    ->      Transactions t ON a.account_id = t.account_id
    -> WHERE
    ->      a.account_id = 9;
```

8. Write a SQL query to Identify customers who have more than one account.

```
mysql> SELECT
    ->      c.customer_id,
    ->      c.first_name,
    ->      c.last_name,
    ->      COUNT(a.account_id) AS account_count
    -> FROM
    ->      Customers c
    -> JOIN
    ->      Accounts a ON c.customer_id = a.customer_id
    -> GROUP BY
    ->      c.customer_id, c.first_name, c.last_name
    -> HAVING
    ->      COUNT(a.account_id) > 1;
```

9. Write a SQL query to Calculate the difference in transaction amounts between deposits and withdrawals.

```
SELECT
    -> account_id,
    -> SUM(CASE WHEN transaction_type = 'deposit' THEN amount ELSE 0 END)
AS total_deposits,
    -> SUM(CASE WHEN transaction_type = 'withdrawal' THEN amount ELSE 0
END) AS total_withdrawals,
    -> SUM(CASE WHEN transaction_type = 'deposit' THEN amount ELSE -amount
END) AS difference
    -> FROM
    -> Transactions
    -> GROUP BY
    -> account_id;
```

10. Write a SQL query to Calculate the average daily balance for each account over a specified period.

```
> SELECT
    -> account_id,
    -> AVG(balance) AS average_daily_balance
    -> FROM
    -> (
    ->     SELECT
    ->         account_id,
    ->         balance,
    ->         transaction_date,
    ->         LAG(balance, 1, balance) OVER (PARTITION BY account_id ORDER
BY transaction_date) AS prev_balance
    ->     FROM
    ->         Transactions
```

```
->                 WHERE
->                         transaction_date BETWEEN ' 2024-01-15' AND ' 2024-01-24'
->         ) AS daily_balances
-> GROUP BY
->         account_id;
```

11. Calculate the total balance for each account type

```
SELECT
->         account_type,
->         SUM(balance) AS total_balance
-> FROM
->         Accounts
-> GROUP BY
->         account_type;
```

12. Identify accounts with the highest number of transactions order by descending order.

```
SELECT
->         a.account_id,
->         COUNT(t.transaction_id) AS transaction_count
-> FROM
->         Accounts a
-> JOIN
->         Transactions t ON a.account_id = t.account_id
-> GROUP BY
->         a.account_id
-> ORDER BY
->         transaction_count DESC;
```

13. List customers with high aggregate account balances, along with their account types.

```
SELECT
    ->      c.customer_id,
    ->      c.first_name,
    ->      c.last_name,
    ->      a.account_type,
    ->      SUM(a.balance) AS total_balance
    -> FROM
    ->      Customers c
    -> JOIN
    ->      Accounts a ON c.customer_id = a.customer_id
    -> GROUP BY
    ->      c.customer_id, c.first_name, c.last_name, a.account_type
    -> ORDER BY
    ->      total_balance DESC;
```

14. Identify and list duplicate transactions based on transaction amount, date, and account.

```
SELECT
    ->      transaction_id,
    ->      account_id,
    ->      transaction_type,
    ->      amount,
    ->      transaction_date
```

```
-> FROM
->     Transactions
-> WHERE
->     (amount, transaction_date, account_id) IN (
->         SELECT
->             amount,
->             transaction_date,
->             account_id
->         FROM
->             Transactions
->         GROUP BY
->             amount, transaction_date, account_id
->         HAVING
->             COUNT(transaction_id) > 1
->     )
-> ORDER BY
->     amount, transaction_date, account_id;
```

Tasks 4: Subquery and its type:

1. Retrieve the customer(s) with the highest account balance.

```
SELECT C.customer_id, C.first_name, C.last_name, C.DOB, C.email, C.phone_number,
A.account_id, A.account_type, A.balance
    -> FROM Customers C
    -> JOIN Accounts A ON C.customer_id = A.customer_id
    -> WHERE A.balance = (SELECT MAX(balance) FROM Accounts);
```

2. Calculate the average account balance for customers who have more than one account.

```
SELECT C.customer_id, AVG(A.balance) AS average_balance
    -> FROM Customers C
    -> JOIN Accounts A ON C.customer_id = A.customer_id
    -> GROUP BY C.customer_id
    -> HAVING COUNT(A.account_id) > 1;
```

3. Retrieve accounts with transactions whose amounts exceed the average transaction amount.

```
SELECT A.account_id, A.customer_id, A.account_type, A.balance, T.transaction_id,
T.transaction_type, T.amount, T.transaction_date

    -> FROM Accounts A
    -> JOIN Transactions T ON A.account_id = T.account_id
    -> WHERE T.amount > (SELECT AVG(amount) FROM Transactions);
```

4. Identify customers who have no recorded transactions.

```
SELECT C.customer_id, C.first_name, C.last_name, C.DOB, C.email, C.phone_number
    -> FROM Customers C
    -> LEFT JOIN Accounts A ON C.customer_id = A.customer_id
    -> LEFT JOIN Transactions T ON A.account_id = T.account_id
    -> WHERE T.transaction_id IS NULL;
```

5. Calculate the total balance of accounts with no recorded transactions.

```
  SELECT SUM(A.balance) AS total_balance
    -> FROM Accounts A
    -> LEFT JOIN Transactions T ON A.account_id = T.account_id
    -> WHERE T.transaction_id IS NULL;
```

6. Retrieve transactions for accounts with the lowest balance.

```
SELECT T.transaction_id, T.account_id, T.transaction_type, T.amount, T.transaction_date
    -> FROM Transactions T
    -> JOIN Accounts A ON T.account_id = A.account_id
    -> WHERE A.balance = (SELECT MIN(balance) FROM Accounts);
```

7. Identify customers who have accounts of multiple types.

```
SELECT C.customer_id, C.first_name, C.last_name, C.DOB, C.email, C.phone_number
    -> FROM Customers C
    -> JOIN Accounts A ON C.customer_id = A.customer_id
```

-> GROUP BY C.customer_id

-> HAVING COUNT(DISTINCT A.account_type) > 1;

8. Calculate the percentage of each account type out of the total number of accounts.

SELECT

-> account_type,

-> COUNT(*) AS total_accounts,

-> (COUNT(*) * 100.0 / (SELECT COUNT(*) FROM Accounts)) AS percentage

-> FROM Accounts

-> GROUP BY account_type;

9. Retrieve all transactions for a customer with a given customer_id.

SELECT T.transaction_id, T.account_id, T.transaction_type, T.amount, T.transaction_date

-> FROM Transactions T

-> JOIN Accounts A ON T.account_id = A.account_id

-> WHERE A.customer_id = 10;

10. Calculate the total balance for each account type, including a subquery within the SELECT clause.

SELECT

-> account_type,

-> (SELECT SUM(balance) FROM Accounts A2 WHERE A2.account_type = A.account_type) AS total_balance

```
-> FROM Accounts A
-> GROUP BY account_type;
```