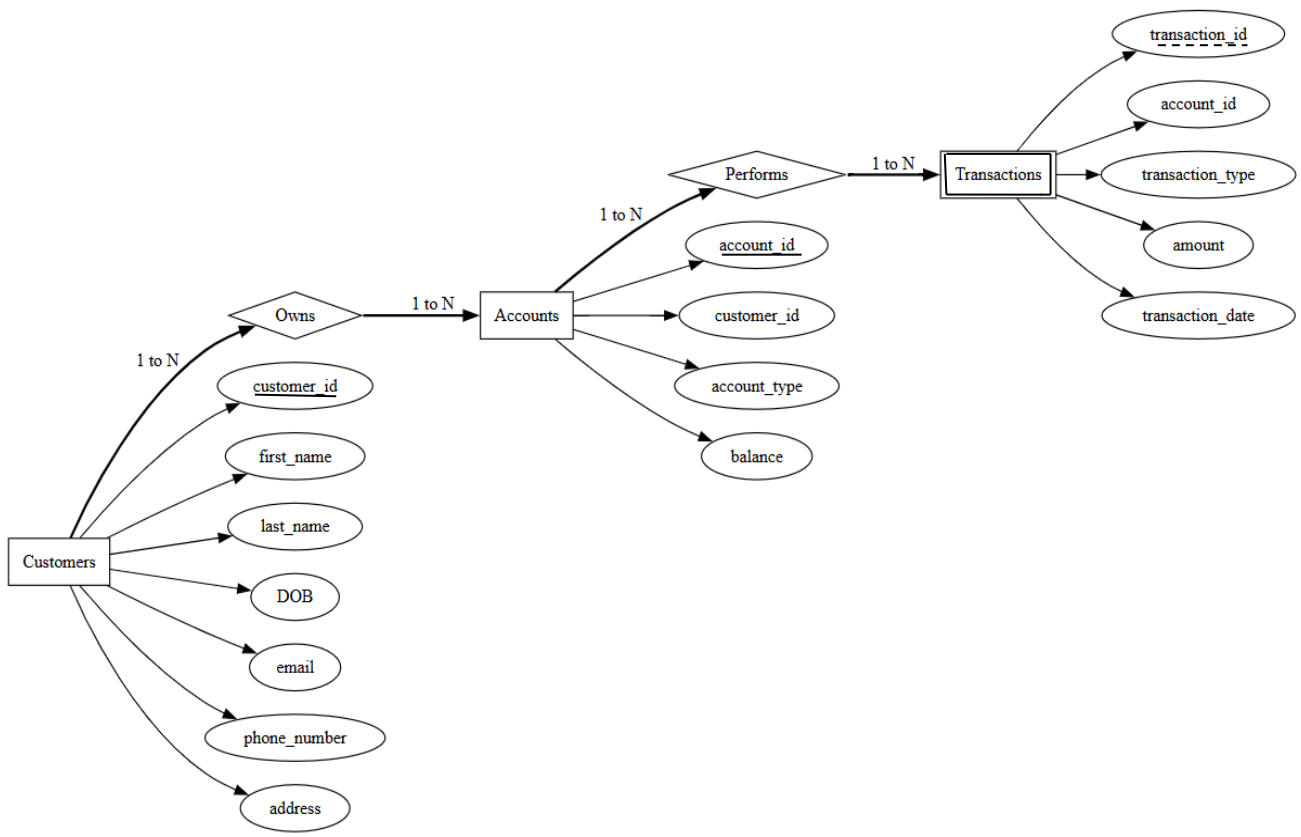
ASSIGNMENT-BANKING SYSTEM

ENTITY-RELATIONSHIP DIAGRAM:



TASK-1

1. Create the database named "HMBank"
2. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships. • Customers
   • Accounts
   • Transactions

```sql
CREATE DATABASE HMBank;
USE HMBank;
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    DOB DATE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(15) UNIQUE NOT NULL,
    address TEXT NOT NULL
);
CREATE TABLE Accounts (
    account_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    account_type ENUM('savings', 'current', 'zero_balance') NOT NULL,
    balance DECIMAL(15,2) NOT NULL CHECK (balance >= 0),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE
);
```

OUTPUT:

#	Time	Action	Message
2	17:43:31	USE HMBank	0 row(s) affected
3	17:44:21	CREATE TABLE Customers (customer_id INT PRIMARY KEY AUTO_INCREMENT, first_name VARCHA...	0 row(s) affected
4	17:47:34	CREATE TABLE Accounts (account_id INT PRIMARY KEY AUTO_INCREMENT, customer_id INT, ac...	0 row(s) affected

```sql
CREATE TABLE Transactions (
    transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    account_id INT,
    transaction_type ENUM('deposit', 'withdrawal', 'transfer') NOT NULL,
    amount DECIMAL(15,2) NOT NULL CHECK (amount > 0),
    transaction_date TIMESTAMP NOT NULL,
    FOREIGN KEY (account_id) REFERENCES Accounts(account_id) ON DELETE CASCADE
);
```

OUTPUT:

5	17:47:39	CREATE TABLE Transactions (transaction_id INT PRIMARY KEY AUTO_INCREMENT, account_id IN...	0 row(s) affected

TASK 2: STEP 1

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

   - Customers
   - Accounts
   - Transactions

CUSTOMERS TABLE:

```sql
INSERT INTO Customers (first_name, last_name, DOB, email, phone_number, address) VALUES
('Varshini', 'Prem', '1985-04-23', 'varsh.prem@gmail.com', '9876543210', '123 Main St'),
('Ram', 'Nehru', '1992-07-12', 'ram.nehru@gmail.com', '9123456789', '456 Elm St'),
('Deepak', 'Prasanna', '1978-01-30', 'deepak.prasanna@gmail.com', '9345678901', '789 Oak St'),
('Swathi', 'Paraman', '1990-09-15', 'swathi.paraman@gmail.com', '9876543221', '321 Maple St'),
('Daniel', 'Williams', '1987-05-10', 'daniel.williams@gmail.com', '9998887776', '654 Pine St'),
('Sophia', 'Lee', '1995-11-05', 'sophia.lee@gmail.com', '9234567890', '987 Birch St'),
('Nishanth', 'Raj', '1980-02-20', 'nishanth.raj@gmail.com', '9356789012', '555 Cedar St'),
('Oviya', 'Miller', '1993-06-25', 'oviya.miller@gmail.com', '9567890123', '876 Willow St'),
('William', 'Davis', '1982-08-18', 'william.davis@gmail.com', '9678901234', '135 Walnut St'),
('Isabella', 'Wilson', '1989-10-28', 'isabella.wilson@gmail.com', '9789012345', '753 Hickory St');
```

OUTPUT:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	first_name	last_name	DOB	email	phone_number	address
1	Varshini	Prem	1985-04-23	varsh.prem@gmail.com	9876543210	123 Main St
2	Ram	Nehru	1992-07-12	ram.nehru@gmail.com	9123456789	456 Elm St
3	Deepak	Prasanna	1978-01-30	deepak.prasanna@gmail.com	9345678901	789 Oak St
4	Swathi	Paraman	1990-09-15	swathi.paraman@gmail.com	9876543221	321 Maple St
5	Daniel	Williams	1987-05-10	daniel.williams@gmail.com	9998887776	654 Pine St
6	Sophia	Lee	1995-11-05	sophia.lee@gmail.com	9234567890	987 Birch St
7	Nishanth	Raj	1980-02-20	nishanth.raj@gmail.com	9356789012	555 Cedar St
8	Oviya	Miller	1993-06-25	oviya.miller@gmail.com	9567890123	876 Willow St
9	William	Davis	1982-08-18	william.davis@gmail.com	9678901234	135 Walnut St
10	Isabella	Wilson	1989-10-28	isabella.wilson@gmail.com	9789012345	753 Hickory St
NULL	NULL	NULL	NULL	NULL	NULL	NULL

ACCOUNTS TABLE:

```sql
INSERT INTO Accounts (customer_id, account_type, balance) VALUES
(1, 'savings', 5000.00),
(2, 'current', 12000.50),
(3, 'zero_balance', 0.00),
(4, 'savings', 8000.75),
(5, 'current', 1500.00),
(6, 'savings', 10000.00),
(7, 'current', 3000.00),
(8, 'savings', 20000.00),
(9, 'zero_balance', 0.00),
(10, 'savings', 5500.25);
Select * from accounts;
```

OUTPUT:

Result Grid | Filter Rows: | Edit:

account_id	customer_id	account_type	balance
1	1	savings	5000.00
2	2	current	12000.50
3	3	zero_balance	0.00
4	4	savings	8000.75
5	5	current	1500.00
6	6	savings	10000.00
7	7	current	3000.00
8	8	savings	20000.00
9	9	zero_balance	0.00
10	10	savings	5500.25
NULL	NULL	NULL	NULL

TRANSACTIONS TABLE:

```sql
INSERT INTO Transactions (account_id, transaction_type, amount, transaction_date) VALUES
(1, 'deposit', 1000.00, '2024-03-01 10:15:00'),
(2, 'withdrawal', 500.00, '2024-03-02 12:30:00'),
(3, 'deposit', 2000.00, '2024-04-03 14:45:00'),
(4, 'transfer', 1500.00, '2024-05-18 16:00:00'),
(5, 'withdrawal', 750.50, '2024-06-05 18:20:00'),
(6, 'deposit', 3000.75, '2024-07-21 20:10:00'),
(7, 'transfer', 500.00, '2024-08-22 22:05:00'),
(8, 'withdrawal', 1000.00, '2024-09-08 09:30:00'),
(9, 'deposit', 2500.00, '2024-03-09 11:45:00'),
(10, 'transfer', 400.25, '2024-03-10 13:55:00');
Select * from transactions;
```

OUTPUT:

Result Grid | Filter Rows: | Edit: | Export/Import:

transaction_id	account_id	transaction_type	amount	transaction_date
1	1	deposit	1000.00	2024-03-01 10:15:00
2	2	withdrawal	500.00	2024-03-02 12:30:00
3	3	deposit	2000.00	2024-04-03 14:45:00
4	4	transfer	1500.00	2024-05-18 16:00:00
5	5	withdrawal	750.50	2024-06-05 18:20:00
6	6	deposit	3000.75	2024-07-21 20:10:00
7	7	transfer	500.00	2024-08-22 22:05:00
8	8	withdrawal	1000.00	2024-09-08 09:30:00
9	9	deposit	2500.00	2024-03-09 11:45:00
10	10	transfer	400.25	2024-03-10 13:55:00
NULL	NULL	NULL	NULL	NULL

TASK 2- STEP 2

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

```
-- Task 2: Step 2
SELECT first_name, last_name, account_type, email
FROM Customers
JOIN Accounts ON Customers.customer_id = Accounts.customer_id;
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	account_type	email
Ram	Nehru	current	ram.nehru@gmail.com
Deepak	Prasanna	zero_balance	deepak.prasanna@gmail.com
Swathi	Paraman	savings	swathi.paraman@gmail.com
Daniel	Williams	current	daniel.williams@gmail.com
Sophia	Lee	savings	sophia.lee@gmail.com
Nishanth	Raj	current	nishanth.raj@gmail.com
Oviya	Miller	savings	oviya.miller@gmail.com
William	Davis	zero_balance	william.davis@gmail.com
Isabella	Wilson	savings	isabella.wilson@gmail.com

2. SQL query to list all transaction corresponding customer.

```
SELECT Customers.first_name, Customers.last_name, Transactions.transaction_type, Transactions.amount, Transactions.transaction_date
FROM Customers
JOIN Accounts ON Customers.customer_id = Accounts.customer_id
JOIN Transactions ON Accounts.account_id = Transactions.account_id;
```

OUTPUT:

Result Grid | Filter Rows: | Export: | Wrap Cell Co

first_name	last_name	transaction_type	amount	transaction_date
Varshini	Prem	deposit	1000.00	2024-03-01 10:15:00
Ram	Nehru	withdrawal	500.00	2024-03-02 12:30:00
Deepak	Prasanna	deposit	2000.00	2024-04-03 14:45:00
Swathi	Paraman	transfer	1500.00	2024-05-18 16:00:00
Daniel	Williams	withdrawal	750.50	2024-06-05 18:20:00
Sophia	Lee	deposit	3000.75	2024-07-21 20:10:00
Nishanth	Raj	transfer	500.00	2024-08-22 22:05:00
Oviya	Miller	withdrawal	1000.00	2024-09-08 09:30:00
William	Davis	deposit	2500.00	2024-03-09 11:45:00
Isabella	Wilson	transfer	400.25	2024-03-10 13:55:00

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

CODE AND OUTPUT:

```
UPDATE Accounts
SET balance = balance + 1000
WHERE account_id = 1;
Select * from accounts;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

account_id	customer_id	account_type	balance
1	1	savings	6000.00
2	2	current	12000.50
3	3	zero_balance	0.00
4	4	savings	8000.75
5	5	current	1500.00
6	6	savings	10000.00
7	7	current	3000.00
8	8	savings	20000.00
9	9	zero_balance	0.00
10	10	savings	5500.25

accounts 12

Output

Action Output

#	Time	Action	Message
31	18:35:43	UPDATE Accounts SET balance = balance + 1000 WHERE account_id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
32	18:35:43	Select * from accounts LIMIT 0, 1000	10 row(s) returned

Select * from accounts

4. SQL query to combine first and last names of customers as a full name.

CODE AND OUTPUT:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM Customers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

full_name
Varshini Prem
Ram Nehru
Deepak Prasanna
Swathi Paraman
Daniel Williams
Sophia Lee
Nishanth Raj
Oviya Miller
William Davis
Isabella Wilson

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

```sql
DELETE FROM Accounts
WHERE balance = 0 AND account_type = 'savings';
```

OUTPUT:

Result Grid | Filter Rows: | Edit:

account_id	customer_id	account_type	balance
1	1	savings	6000.00
2	2	current	12000.50
3	3	zero_balance	0.00
4	4	savings	8000.75
5	5	current	1500.00
6	6	savings	10000.00
7	7	current	3000.00
8	8	savings	20000.00
9	9	zero_balance	0.00
10	10	savings	5500.25
NULL	NULL	NULL	NULL

6. SQL query to find customers living in a specific city.

```sql
SELECT * FROM Customers WHERE address LIKE '%Elm%';
```

OUTPUT:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

customer_id	first_name	last_name	DOB	email	phone_number	address
2	Ram	Nehru	1992-07-12	ram.nehru@gmail.com	9123456789	456 Elm St
NULL	NULL	NULL	NULL	NULL	NULL	NULL

7. SQL query to get the account balance for a specific account.

```
88 •  SELECT balance FROM Accounts WHERE account_id = 2;
89
90
91
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

balance
12000.50

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

```
90 •  SELECT * FROM Accounts WHERE account_type = 'current' AND balance > 1000;
91
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

account_id	customer_id	account_type	balance
2	2	current	12000.50
5	5	current	1500.00
7	7	current	3000.00
NULL	NULL	NULL	NULL

9. SQL query to retrieve all transactions for a specific account.

Result Grid | Filter Rows: | Edit: | Export/Import:

transaction_id	account_id	transaction_type	amount	transaction_date
1	1	deposit	1000.00	2024-03-01 10:15:00
NULL	NULL	NULL	NULL	NULL

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

```
SELECT account_id, balance, balance * 0.04 AS interest FROM Accounts WHERE account_type = 'savings';
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	balance	interest
1	6000.00	240.0000
4	8000.75	320.0300
6	10000.00	400.0000
8	20000.00	800.0000
10	5500.25	220.0100

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

```
SELECT * FROM Accounts WHERE balance < -500;
```

Result Grid | Filter Rows: | Edit: | Export/

account_id	customer_id	account_type	balance
NULL	NULL	NULL	NULL

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

```
SELECT * FROM Customers WHERE address NOT LIKE '%New York%';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	first_name	last_name	DOB	email	phone_number	address
1	Varshini	Prem	1985-04-23	varsh.prem@gmail.com	9876543210	123 Main St
2	Ram	Nehru	1992-07-12	ram.nehru@gmail.com	9123456789	456 Elm St
3	Deepak	Prasanna	1978-01-30	deepak.prasanna@gmail.com	9345678901	789 Oak St
4	Swathi	Paraman	1990-09-15	swathi.paraman@gmail.com	9876543221	321 Maple St
5	Daniel	Williams	1987-05-10	daniel.williams@gmail.com	9998887776	654 Pine St
6	Sophia	Lee	1995-11-05	sophia.lee@gmail.com	9234567890	987 Birch St
7	Nishanth	Raj	1980-02-20	nishanth.raj@gmail.com	9356789012	555 Cedar St
8	Oviya	Miller	1993-06-25	oviya.miller@gmail.com	9567890123	876 Willow St
9	William	Davis	1982-08-18	william.davis@gmail.com	9678901234	135 Walnut St
10	Isabella	Wilson	1989-10-28	isabella.wilson@gmail.com	9789012345	753 Hickory St
NULL	NULL	NULL	NULL	NULL	NULL	NULL

TASK 3

1. SQL query to find the average account balance for all customers.

```sql
-- Task 3- 1
SELECT customer_id, AVG(balance) AS average_balance
FROM Accounts
GROUP BY customer_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

customer_id	average_balance
1	6000.000000
2	12000.500000
3	0.000000
4	8000.750000
5	1500.000000
6	10000.000000
7	3000.000000
8	20000.000000
9	0.000000
10	5500.250000

2. SQL query to retrieve the top 10 highest account balances.

```sql
-- Task 3-2
SELECT account_id, customer_id, balance
FROM Accounts
ORDER BY balance DESC
LIMIT 10;
```

Result Grid | Filter Rows: | Edit:

account_id	customer_id	balance
8	8	20000.00
2	2	12000.50
6	6	10000.00
4	4	8000.75
1	1	6000.00
10	10	5500.25
7	7	3000.00
5	5	1500.00
3	3	0.00
9	9	0.00
NULL	NULL	NULL

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

```sql
-- Task 3-3
SELECT A.customer_id, SUM(T.amount) AS total_deposits
FROM Transactions T
JOIN Accounts A ON T.account_id = A.account_id
WHERE T.transaction_type = 'deposit'
AND T.transaction_date = '2024-03-09 11:45:00'
GROUP BY A.customer_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

customer_id	total_deposits
9	2500.00

4**.** SQL query to find the Oldest and Newest Customers.

```sql
-- Task 3-4
SELECT c.first_name, c.last_name, c.DOB, t.transaction_date
FROM Customers c
JOIN Accounts a ON c.customer_id = a.customer_id
JOIN Transactions t ON a.account_id = t.account_id
ORDER BY t.transaction_date ASC
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	DOB	transaction_date
Varshini	Prem	1985-04-23	2024-03-01 10:15:00

```sql
SELECT c.first_name, c.last_name, c.DOB, t.transaction_date
FROM Customers c
JOIN Accounts a ON c.customer_id = a.customer_id
JOIN Transactions t ON a.account_id = t.account_id
ORDER BY t.transaction_date DESC
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	DOB	transaction_date
Oviya	Miller	1993-06-25	2024-09-08 09:30:00

5. SQL query to retrieve transaction details along with the account type.

```sql
-- Task 3-5
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;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	account_id	account_type	transaction_type	amount	transaction_date
1	1	savings	deposit	1000.00	2024-03-01 10:15:00
2	2	current	withdrawal	500.00	2024-03-02 12:30:00
3	3	zero_balance	deposit	2000.00	2024-04-03 14:45:00
4	4	savings	transfer	1500.00	2024-05-18 16:00:00
5	5	current	withdrawal	750.50	2024-06-05 18:20:00
6	6	savings	deposit	3000.75	2024-07-21 20:10:00
7	7	current	transfer	500.00	2024-08-22 22:05:00
8	8	savings	withdrawal	1000.00	2024-09-08 09:30:00
9	9	zero_balance	deposit	2500.00	2024-03-09 11:45:00
10	10	savings	transfer	400.25	2024-03-10 13:55:00

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

```sql
-- Task 3-6
SELECT C.customer_id, C.first_name, C.last_name, A.account_id, A.account_type, A.balance
FROM Customers C
JOIN Accounts A ON C.customer_id = A.customer_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	first_name	last_name	account_id	account_type	balance
1	Varshini	Prem	1	savings	6000.00
2	Ram	Nehru	2	current	12000.50
3	Deepak	Prasanna	3	zero_balance	0.00
4	Swathi	Paraman	4	savings	8000.75
5	Daniel	Williams	5	current	1500.00
6	Sophia	Lee	6	savings	10000.00
7	Nishanth	Raj	7	current	3000.00
8	Oviya	Miller	8	savings	20000.00
9	William	Davis	9	zero_balance	0.00
10	Isabella	Wilson	10	savings	5500.25

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

```sql
-- Task 3-7
SELECT T.transaction_id, T.transaction_type, T.amount, T.transaction_date, C.first_name, C.last_name, C.email
FROM Transactions T
JOIN Accounts A ON T.account_id = A.account_id
JOIN Customers C ON A.customer_id = C.customer_id
WHERE T.account_id = 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	transaction_type	amount	transaction_date	first_name	last_name	email
5	withdrawal	750.50	2024-06-05 18:20:00	Daniel	Williams	daniel.williams@gmail.com

8. SQL query to identify customers who have more than one account.

```sql
-- Tasl 3-8
SELECT customer_id, COUNT(account_id) AS account_count
FROM Accounts
GROUP BY customer_id
HAVING COUNT(account_id) > 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	account_count

9. SQL query to calculate the difference in transaction amounts between deposits and withdrawals.

```sql
-- Task 3-9
SELECT account_id,
       SUM(CASE WHEN transaction_type = 'deposit' THEN amount ELSE 0 END) -
       SUM(CASE WHEN transaction_type = 'withdrawal' THEN amount ELSE 0 END) AS balance_difference
FROM Transactions
GROUP BY account_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	balance_difference
1	2000.00
2	-500.00
3	1000.00
4	3000.00
5	-200.00
6	5000.00
7	-1500.00
8	7000.00
9	800.00
10	-1000.00

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

```sql
-- Task 3-10
SELECT account_id, AVG(balance) AS avg_daily_balance
FROM Accounts
WHERE account_id IN (SELECT DISTINCT account_id FROM Transactions WHERE transaction_date BETWEEN '2024-03-01' AND '2024-05-18')
GROUP BY account_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	avg_daily_balance
1	5000.000000
2	12000.500000
3	0.000000
9	0.000000
10	5500.250000

11. Calculate the total balance for each account type.

```
-- Task 3-11
SELECT account_type, SUM(balance) AS total_balance
FROM Accounts
GROUP BY account_type;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Co

account_type	total_balance
savings	49501.00
current	16500.50
zero_balance	0.00

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

```
-- Task 3-12
SELECT account_id, COUNT(transaction_id) AS transaction_count
FROM Transactions
GROUP BY account_id
ORDER BY transaction_count DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	transaction_count
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

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

```
-- Task 3-13
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, A.account_type
HAVING SUM(A.balance) > 5000;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	first_name	last_name	account_type	total_balance
1	Varshini	Prem	savings	6000.00
2	Ram	Nehru	current	12000.50
4	Swathi	Paraman	savings	8000.75
6	Sophia	Lee	savings	10000.00
8	Oviya	Miller	savings	20000.00
10	Isabella	Wilson	savings	5500.25

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

```
-- Task 3-14
SELECT account_id, amount, transaction_date, COUNT(*) AS duplicate_count,
       GROUP_CONCAT(transaction_id) AS duplicate_transaction_ids
FROM Transactions
GROUP BY account_id, amount, transaction_date
HAVING COUNT(*) > 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	amount	transaction_date	duplicate_count	duplicate_transaction_ids

TASK-4

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

```
-- Task 4-1
SELECT customer_id, first_name, last_name
FROM Customers
WHERE customer_id IN (SELECT customer_id FROM Accounts WHERE balance = (SELECT MAX(balance) FROM Accounts));
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	first_name	last_name
8	Oviya	Miller
NULL	NULL	NULL

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

```
-- Task 4-2
SELECT AVG(balance) AS avg_balance
FROM Accounts
WHERE customer_id IN (SELECT customer_id FROM Accounts GROUP BY customer_id HAVING COUNT(account_id) > 1);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

avg_balance
NULL

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

```
-- Task 4-3
SELECT * FROM Transactions
WHERE amount > (SELECT AVG(amount) FROM Transactions);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

transaction_id	account_id	transaction_type	amount	transaction_date
3	3	deposit	2000.00	2024-04-03 14:45:00
4	4	transfer	1500.00	2024-05-18 16:00:00
6	6	deposit	3000.75	2024-07-21 20:10:00
9	9	deposit	2500.00	2024-03-09 11:45:00
NULL	NULL	NULL	NULL	NULL

4.Identify customers who have no recorded transactions.

```
-- Task 4-4
SELECT * FROM Customers
WHERE customer_id NOT IN (SELECT DISTINCT customer_id FROM Accounts A JOIN Transactions T ON A.account_id = T.account_id);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	first_name	last_name	DOB	email	phone_number	address
NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

```
-- Task 4-5
SELECT SUM(balance) AS total_balance
FROM Accounts
WHERE account_id NOT IN (SELECT DISTINCT account_id FROM Transactions);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total_balance
NULL

6.Retrieve transactions for accounts with the lowest balance.

```
-- Task 4-6
SELECT * FROM Transactions
WHERE account_id IN (
    SELECT account_id
    FROM Accounts
    WHERE balance = (SELECT MIN(balance) FROM Accounts)
);
```

Result Grid | Filter Rows: | Edit: | Export/Imp

transaction_id	account_id	transaction_type	amount	transaction_date
3	3	deposit	2000.00	2024-04-03 14:45:00
9	9	deposit	2500.00	2024-03-09 11:45:00
NULL	NULL	NULL	NULL	NULL

7.Identify customers who have accounts of multiple types.

```
-- Task 4-7
SELECT customer_id
FROM Accounts
GROUP BY customer_id
HAVING COUNT(DISTINCT account_type) > 1;
```

Result Grid | Filter Rows: | Export:

customer_id

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

```
-- Task 4-8
SELECT
    account_type,
    COUNT(*) AS account_count,
    (COUNT(*) * 100.0 / (SELECT COUNT(*) FROM Accounts)) AS percentage
FROM Accounts
GROUP BY account_type;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_type	account_count	percentage
savings	5	50.00000
current	3	30.00000
zero_balance	2	20.00000

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

```
-- Task 4-9
SELECT T.*
FROM Transactions T
JOIN Accounts A ON T.account_id = A.account_id
JOIN Customers C ON A.customer_id = C.customer_id
WHERE C.customer_id =4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	account_id	transaction_type	amount	transaction_date
4	4	transfer	1500.00	2024-05-18 16:00:00

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

```
-- Task 4-10
SELECT
    account_type,
    (SELECT SUM(balance) FROM Accounts A2 WHERE A2.account_type = A1.account_type) AS total_balance
FROM Accounts A1
GROUP BY account_type;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_type	total_balance
savings	48501.00
current	16500.50
zero_balance	0.00

CONTROL STRUCTURE

TASK 1:

Conditional Statements

```
import java.util.Scanner;

public class LoanEligibility {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your credit score: ");
        int creditScore = sc.nextInt();

        System.out.print("Enter your annual income: ");
        double annualIncome = sc.nextDouble();

        if (creditScore > 700 && annualIncome >= 50000) {
            System.out.println("You are eligible for a loan.");
        } else {
            System.out.println("Sorry, you are not eligible for a loan.");
        }

        sc.close();
    }
}
```

TASK 2:

Nested Conditional Statements

```
import java.util.Scanner;

public class ATMSimulation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double balance = 5000;

        System.out.println("Welcome to ATM!");
        System.out.println("1. Check Balance\n2. Withdraw\n3. Deposit");
        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();

        if (choice == 1) {
            System.out.println("Balance: $" + balance);
        } else if (choice == 2) {
            System.out.print("Enter amount to withdraw: ");
            double amount = sc.nextDouble();
            if (amount <= balance) {
                if (amount % 100 == 0 || amount % 500 == 0) {
                    balance -= amount;
                    System.out.println("Withdrawal successful. Balance: $" +
```

```
balance);
                } else {
                    System.out.println("Amount must be multiple of 100 or
500.");
                }
            } else {
                System.out.println("Insufficient funds.");
            }
        } else if (choice == 3) {
            System.out.print("Enter amount to deposit: ");
            double amount = sc.nextDouble();
            balance += amount;
            System.out.println("Deposit successful. Balance: $" + balance);
        } else {
            System.out.println("Invalid choice.");
        }

        sc.close();
    }
}
```

TASK 3:

Loop Structures:

```
import java.util.Scanner;

public class CompoundInterest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of customers: ");
        int n = sc.nextInt();

        for (int i = 1; i <= n; i++) {
            System.out.println("\nbean.Customer " + i);
            System.out.print("Initial balance: ");
            double balance = sc.nextDouble();
            System.out.print("Interest rate (%): ");
            double rate = sc.nextDouble();
            System.out.print("Years: ");
            int years = sc.nextInt();

            double futureBalance = balance * Math.pow(1 + rate / 100, years);
            System.out.printf("Future Balance: %.2f\n", futureBalance);
        }

        sc.close();
    }
}
```

TASK 4:

Looping, Array and Data Validation

```
import java.util.Scanner;

public class AccountChecker {
    public static void main(String[] args) {
        String[] accountNumbers = {"1001", "1002", "1003"};
        double[] balances = {2500.50, 5600.00, 920.75};

        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.print("Enter account number: ");
            String acc = sc.next();
            boolean found = false;

            for (int i = 0; i < accountNumbers.length; i++) {
                if (acc.equals(accountNumbers[i])) {
                    System.out.println("Balance: $" + balances[i]);
                    found = true;
                    break;
                }
            }

            if (found) {
                break;
            } else {
                System.out.println("Invalid account number. Try again.");
            }
        }

        sc.close();
    }
}
```

TASK 5:

Password Validation

```
import java.util.Scanner;

public class PasswordValidator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter password: ");
        String pwd = sc.next();

        boolean hasUpper = false, hasDigit = false;

        if (pwd.length() >= 8) {
```

```
            for (char ch : pwd.toCharArray()) {
                if (Character.isUpperCase(ch)) hasUpper = true;
                if (Character.isDigit(ch)) hasDigit = true;
            }

            if (hasUpper && hasDigit) {
                System.out.println("Valid password.");
            } else {
                System.out.println("Must include uppercase and digit.");
            }
        } else {
            System.out.println("Password must be at least 8 characters.");
        }

        sc.close();
    }
}
```

TASK 6

Transaction History

```
import java.util.ArrayList;
import java.util.Scanner;

public class TransactionHistory {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<String> transactions = new ArrayList<>();

        while (true) {
            System.out.print("Enter transaction type (deposit/withdraw/exit): 
");
            String type = sc.next().toLowerCase();

            if (type.equals("exit")) break;

            if (type.equals("deposit") || type.equals("withdraw")) {
                System.out.print("Enter amount: ");
                double amt = sc.nextDouble();
                transactions.add(type + " of $" + amt);
            } else {
                System.out.println("Invalid transaction type.");
            }
        }

        System.out.println("\n Transaction History:");
        for (String t : transactions) {
            System.out.println("- " + t);
        }

        sc.close();
    }
}
```

OOPS, Collections and Exception Handling

BEAN PACKAGE:

Accounts.java:

```
package bean;
import exception.InvalidAccountException;
import exception.InsufficientFundException;
import exception.OverDraftLimitExceededException;
public class Accounts {
    private static long lastAccNo = 1000;

    private String accountNumber;
    private String accountType; // Savings or Current
    protected double accountBalance;
    private Customer customer; // Association: Account "has a" bean.Customer

    // Default Constructor
    public Accounts() {
        this.accountNumber = "";
        this.accountType = "";
        this.accountBalance = 0.0;
        this.customer = null;
    }

    // Parameterized Constructor (with customer)
    public Accounts(String accountNumber, String accountType, double
accountBalance, Customer customer) {
        this.accountNumber = accountNumber;
        this.accountType = accountType;
        this.accountBalance = accountBalance;
        this.customer = customer;
    }
    @Override
    public String toString() {
        return "Account Details:\n" +
                "Account Number: " + accountNumber + "\n" +
                "Name           : " + customer + "\n" +
                "Balance        : " + accountBalance;
    }
    private String generateAccountNumber() {
        return String.valueOf(++lastAccNo);
    }

    // Getters and Setters
    public String getAccountNumber() { return accountNumber; }
    public void setAccountNumber(String accountNumber) { this.accountNumber =
accountNumber; }

    public String getAccountType() { return accountType; }
    public void setAccountType(String accountType) { this.accountType =
accountType; }

    public double getAccountBalance() { return accountBalance; }
```

```java
    public void setAccountBalance(double accountBalance) {
this.accountBalance = accountBalance; }

    public Customer getCustomer() { return customer; }
    public void setCustomer(Customer customer) { this.customer = customer; }

    // Print Account Info
    public void printAccountInfo() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Account Type  : " + accountType);
        System.out.println("Balance       : $" + accountBalance);
        if (customer != null) {
            System.out.println("Name          : " + customer.getFirstName() +
" " + customer.getLastName());
            System.out.println("Email         : " +
customer.getEmailAddress());
            System.out.println("Phone         : " +
customer.getPhoneNumber());
            System.out.println("Address       : " + customer.getAddress());
        } else {
            System.out.println("No customer linked to this account.");
        }
    }


    // Deposit
    public void deposit(double amount) {
        if (amount > 0) {
            accountBalance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    // Overloaded withdraw methods
    public void withdraw(float amount) throws
InsufficientFundException,OverDraftLimitExceededException {
        withdraw((double) amount);
    }

    // Withdraw
    public void withdraw(double amount) throws
InsufficientFundException,OverDraftLimitExceededException {
        if (amount <= accountBalance && amount > 0) {
            accountBalance -= amount;
            System.out.println("Withdrawn: $" + amount);
        } else {
            throw new InsufficientFundException("Insufficient balance or
invalid amount.");
        }
    }
    // Calculate Interest (4.5%)
    public void calculateInterest() {
        if (accountType.equalsIgnoreCase("Savings")) {
            double interest = accountBalance * 0.045;
            accountBalance += interest;
```

```
            System.out.println("Interest of $" + interest + " added. New
Balance: $" + accountBalance);
        } else {
            System.out.println("Interest only applicable for Savings
accounts.");
        }
    }
    public static long getLastAccNo() {
        return lastAccNo;
    }
    public static void setLastAccNo(long accNo) {
        lastAccNo = accNo;
    }
}
```

Customer.java:

```
package bean;

public class Customer {
    private String customerId;
    private String firstName;
    private String lastName;
    private String email;
    private String phone;
    private String address;

    // Default Constructor
    public Customer() {
        this.customerId = "";
        this.firstName = "";
        this.lastName = "";
        this.email = "";
        this.phone = "";
        this.address = "";
    }

    // Parameterized Constructor
    public Customer(String customerId, String firstName, String lastName,
String email, String phone, String address) {
        this.customerId = customerId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.phone = phone;
        this.address = address;
    }

    // Getters and Setters
    public String getCustomerId() { return customerId; }
    public void setCustomerId(String customerId) { this.customerId =
customerId; }

    public String getFirstName() { return firstName; }
```

```
    public void setFirstName(String firstName) { this.firstName = firstName;
}

    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }

    public String getAddress() { return address; }
    public void setAddress(String address) { this.address = address; }

    // Method to print customer details
    public void printCustomerInfo() {
        System.out.println("Customer ID:" + customerId);
        System.out.println("Name: " + firstName + " " + lastName);
        System.out.println("Email: " + email);
        System.out.println("Phone: " + phone);
        System.out.println("Address: " + address);
    }
    public String getCustomerID() {
        return customerId;
    }

    public String getEmailAddress() {
        return email;
    }

    public String getPhoneNumber() {
        return phone;
    }
    @Override
    public String toString() {
        return firstName + " " + lastName;
    }
}
```

HMBank.java:

```
package bean;

import java.util.HashMap;
import java.util.Map;
import exception.InvalidAccountException;
import exception.InsufficientFundException;
import exception.OverDraftLimitExceededException;


public class HMBank {
    private Map<Long, Accounts> accountsMap;
    private static long nextAccountNumber = 1001;
```

```java
    public HMBank() {
        accountsMap = new HashMap<>();
    }

    // Create Account
    public long create_account(Customer customer, String accType, float
balance) {
        long accNo = nextAccountNumber++;
        String accNumStr = String.valueOf(accNo);

        Accounts account;
        if (accType.equalsIgnoreCase("Savings")) {
            account = new SavingsAccount(accNumStr, accType, balance,
customer, 4.5);
        } else if (accType.equalsIgnoreCase("Current")) {
            account = new CurrentAccount(accNumStr, accType, balance,
customer, 500.0);
        } else {
            System.out.println("Invalid account type. Account not created.");
            return -1;
        }

        account.setCustomer(customer);
        accountsMap.put(accNo, account);
        System.out.println("Account created with Account Number: " + accNo);
        return accNo;
    }

    // Get Account Balance
    public double get_account_balance(long accountNumber) throws
InvalidAccountException {
        Accounts account = accountsMap.get(accountNumber);
        if (account != null) return account.getAccountBalance();
        throw new InvalidAccountException("Account not found.");
    }

    // Deposit
    public double deposit(long accountNumber, float amount) {
        Accounts account = accountsMap.get(accountNumber);
        if (account != null) {
            account.deposit(amount);
            return account.getAccountBalance();
        } else {
            System.out.println("Account not found.");
            return -1;
        }
    }

    // Withdraw
    public double withdraw(long accountNumber, float amount)
            throws InvalidAccountException, InsufficientFundException,
OverDraftLimitExceededException {
        Accounts account = accountsMap.get(accountNumber);
        if (account == null) {
            throw new InvalidAccountException("Account number " +
accountNumber + " not found.");
```

```
        }

        if (account instanceof CurrentAccount) {
            ((CurrentAccount) account).withdraw(amount);
        } else {
            account.withdraw(amount);
        }

        return account.getAccountBalance();
    }


    // Transfer
    public void transfer(long fromAccount, long toAccount, float amount)
            throws InvalidAccountException, InsufficientFundException,
OverDraftLimitExceededException {
        Accounts fromAcc = accountsMap.get(fromAccount);
        Accounts toAcc = accountsMap.get(toAccount);

        if (fromAcc == null || toAcc == null) {
            throw new InvalidAccountException("One or both accounts not
found.");
        }

        if (fromAcc instanceof CurrentAccount) {
            ((CurrentAccount) fromAcc).withdraw(amount);
        } else {
            fromAcc.withdraw(amount);
        }

        toAcc.deposit(amount);
        System.out.println("Transferred ₹" + amount + " from " + fromAccount
+ " to " + toAccount);
    }


    // Get Account + bean.Customer Details
    public void getAccountDetails(long accountNumber) throws
InvalidAccountException {
        Accounts account = accountsMap.get(accountNumber);
        if (account != null) {
            account.printAccountInfo();
        } else {
            throw new InvalidAccountException("Account not found.");
        }
    }
}
```

CurrentAccount.java

```
package bean;
import exception.OverDraftLimitExceededException;

public class CurrentAccount extends Accounts {
```

```
    private double overdraftLimit;

    public CurrentAccount(String accountNumber, String accountType, double
balance, Customer customer, double overdraftLimit) {
        super(accountNumber, accountType, balance, customer);
        this.overdraftLimit = overdraftLimit;
    }

    public double getOverdraftLimit() { return overdraftLimit; }
    public void setOverdraftLimit(double overdraftLimit) {
this.overdraftLimit = overdraftLimit; }

    @Override
    public void withdraw(double amount) throws
OverDraftLimitExceededException {
        if (amount <= accountBalance + overdraftLimit) {
            accountBalance -= amount;
            System.out.println("Withdrawal of ₹" + amount + " successful. New
balance: ₹" + accountBalance);
        } else {
            throw new OverDraftLimitExceededException("Withdrawal failed.
Amount exceeds overdraft limit.");
        }
    }

    @Override
    public void calculateInterest() {
        System.out.println("ℹ No interest is applicable on Current Account.");
    }
}
```

SavingsAccount.java:

```
package bean;

public class SavingsAccount extends Accounts {
    private double interestRate;

    public SavingsAccount(String accountNumber, String accountType, double
balance, Customer customer, double interestRate) {
        super(accountNumber, accountType, balance, customer);
        this.interestRate = interestRate;
    }

    public double getInterestRate() { return interestRate; }
    public void setInterestRate(double interestRate) { this.interestRate =
interestRate; }


    @Override
    public void calculateInterest() {
        double interest = accountBalance * (interestRate / 100);
        accountBalance += interest;
        System.out.println("Interest of ₹" + interest + " added. New Balance:
```

```
₹" + accountBalance);
    }
}
```

ZeroBalanceAccount.java:

```
package bean;

public class ZeroBalanceAccount extends Accounts {

    public ZeroBalanceAccount(String accountNumber, String accountType,
double balance, Customer customer) {
        super(accountNumber, accountType, balance, customer);
    }
@Override
    public void calculateInterest() {
        System.out.println("ℹ No interest for Zero Balance Account.");
    }
}
```

Transaction.java:

```
package bean;

import java.time.LocalDateTime;

public class Transaction {
    private Accounts account;
    private String description;
    private LocalDateTime dateTime;
    private String transactionType;
    private double transactionAmount;


    public Transaction() {
    }


    public Transaction(Accounts account, String description, String
transactionType, double transactionAmount) {
        this.account = account;
        this.description = description;
        this.transactionType = transactionType;
        this.transactionAmount = transactionAmount;
        this.dateTime = LocalDateTime.now();
    }

    public Accounts getAccount() { return account; }
    public void setAccount(Accounts account) { this.account = account; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description =
```

```
description; }

    public LocalDateTime getDateTime() { return dateTime; }
    public void setDateTime(LocalDateTime dateTime) { this.dateTime =
dateTime; }

    public String getTransactionType() { return transactionType; }
    public void setTransactionType(String transactionType) {
this.transactionType = transactionType; }

    public double getTransactionAmount() { return transactionAmount; }
    public void setTransactionAmount(double transactionAmount) {
this.transactionAmount = transactionAmount; }

    @Override
    public String toString() {
        return "Transaction [" + transactionType + " of ₹" +
transactionAmount +
                " on " + dateTime + "]";
    }
}
```

SERVICE PACKAGE:

ICustomerServiceProvider.java:

```
package service;
import bean.Accounts;
import java.util.Date;
import java.util.List;
import exception.InsufficientFundException;
import exception.OverDraftLimitExceededException;

public interface ICustomerServiceProvider {
    double get_account_balance(long accountNumber);

    double deposit(long accountNumber, float amount);

    double withdraw(long accountNumber, float amount)
            throws InsufficientFundException,
OverDraftLimitExceededException;

    void transfer(long fromAccount, int toAccount, float amount)
            throws InsufficientFundException,
OverDraftLimitExceededException;

    void printAccountDetails(long accountNumber); // clearer name
    List<bean.Transaction> getTransactions(long accountNumber, Date from,
Date to);
}
```

IBankServiceProvider.java:

```
package service;

import bean.Accounts;
import bean.Customer;
import java.util.List;

public interface IBankServiceProvider {
    void create_account(Customer customer,long accNo,String accType, double
balance);
    List<Accounts> listAccounts();
    Accounts getAccountDetails(long accountNumber);
    void calculateInterest();
}
```

IBankRepository.java:

```
package service;

import bean.Accounts;
import bean.Customer;
import bean.Transaction;

import java.sql.Date;
import java.util.List;
import java.sql.Timestamp;

public interface IBankRepository {
    void createAccount(Customer customer, long accNo, String accType, float
balance) throws Exception;

    List<Accounts> listAccounts() throws Exception;

    void calculateInterest() throws Exception;

    double getAccountBalance(long accountNumber) throws Exception;

    double deposit(long accountNumber, float amount) throws Exception;

    double withdraw(long accountNumber, float amount) throws Exception;

    void transfer(long fromAccount, long toAccount, float amount) throws
Exception;

    Accounts getAccountDetails(long accountNumber) throws Exception;

    List<Transaction> getTransactions(long accountNumber, Timestamp fromDate,
Timestamp toDate) throws Exception;
}
```

IMPLEMENTATIONS:

BankRepositoryImpl.java:

```
package repository;

import bean.*;
import service.IBankRepository;
import util.DBConnUtil;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class BankRepositoryImpl implements IBankRepository {

    @Override
    public void createAccount(Customer customer, long accNo, String accType,
float balance) throws Exception {
        Connection conn = DBConnUtil.getConnection();

        String insertCustomer = "INSERT INTO customers (customer_id,
first_name, last_name, email, phone_number, address) VALUES (?, ?, ?, ?, ?,
?)";
        PreparedStatement cs = conn.prepareStatement(insertCustomer);
        cs.setString(1, customer.getCustomerID());
        cs.setString(2, customer.getFirstName());
        cs.setString(3, customer.getLastName());
        cs.setString(4, customer.getEmailAddress());
        cs.setString(5, customer.getPhoneNumber());
        cs.setString(6, customer.getAddress());
        cs.executeUpdate();

        String insertAccount = "INSERT INTO accounts (account_id,
account_type, balance, customer_id) VALUES (?, ?, ?, ?)";
        PreparedStatement as = conn.prepareStatement(insertAccount);
        as.setLong(1, accNo);
        as.setString(2, accType);
        as.setFloat(3, balance);
        as.setString(4, customer.getCustomerID());
        as.executeUpdate();

        conn.close();
    }

    @Override
    public List<Accounts> listAccounts() throws Exception {
        List<Accounts> list = new ArrayList<>();
        Connection conn = DBConnUtil.getConnection();

        String query = "SELECT a.*, c.first_name, c.last_name, c.email,
c.phone_number, c.address, c.customer_id " +
                "FROM accounts a JOIN customers c ON a.customer_id =
c.customer_id";
```

```
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query);

        while (rs.next()) {
            Accounts acc = new Accounts();
            acc.setAccountNumber(String.valueOf(rs.getLong("account_id")));
            acc.setAccountType(rs.getString("account_type"));
            acc.setAccountBalance(rs.getDouble("balance"));

            Customer c = new Customer(
                    rs.getString("customer_id"),
                    rs.getString("first_name"),
                    rs.getString("last_name"),
                    rs.getString("email"),
                    rs.getString("phone_number"),
                    rs.getString("address")
            );

            acc.setCustomer(c);
            list.add(acc);
        }

        conn.close();
        return list;
    }

    @Override
    public void calculateInterest() throws Exception {
        Connection conn = DBConnUtil.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM accounts WHERE
account_type='Savings'");

        while (rs.next()) {
            long accNo = rs.getLong("account_id");
            double balance = rs.getDouble("balance");
            double interest = balance * 0.045;
            double newBal = balance + interest;

            PreparedStatement ps = conn.prepareStatement("UPDATE accounts SET
balance=? WHERE account_id=?");
            ps.setDouble(1, newBal);
            ps.setLong(2, accNo);
            ps.executeUpdate();
        }

        conn.close();
    }

    @Override
    public double getAccountBalance(long accountNumber) throws Exception {
        Connection conn = DBConnUtil.getConnection();
        PreparedStatement ps = conn.prepareStatement("SELECT balance FROM
accounts WHERE account_id=?");
        ps.setLong(1, accountNumber);
        ResultSet rs = ps.executeQuery();
```

```
        if (rs.next()) {
            return rs.getDouble("balance");
        }

        throw new Exception("Account not found.");
    }

    @Override
    public double deposit(long accountNumber, float amount) throws Exception
{
        double newBalance = getAccountBalance(accountNumber) + amount;
        Connection conn = DBConnUtil.getConnection();
        PreparedStatement ps = conn.prepareStatement("UPDATE accounts SET
balance=? WHERE account_id=?");
        ps.setDouble(1, newBalance);
        ps.setLong(2, accountNumber);
        ps.executeUpdate();

        recordTransaction(accountNumber, "Deposit", amount);
        return newBalance;
    }

    @Override
    public double withdraw(long accountNumber, float amount) throws Exception
{
        double balance = getAccountBalance(accountNumber);
        if (balance < amount) throw new Exception("Insufficient funds.");

        double newBalance = balance - amount;
        Connection conn = DBConnUtil.getConnection();
        PreparedStatement ps = conn.prepareStatement("UPDATE accounts SET
balance=? WHERE account_id=?");
        ps.setDouble(1, newBalance);
        ps.setLong(2, accountNumber);
        ps.executeUpdate();

        recordTransaction(accountNumber, "Withdrawal", amount);
        return newBalance;
    }

    @Override
    public void transfer(long fromAccount, long toAccount, float amount)
throws Exception {
        withdraw(fromAccount, amount);
        deposit(toAccount, amount);
        recordTransaction(fromAccount, "Transfer", amount);
        recordTransaction(toAccount, "Transfer", amount);
    }

    private void recordTransaction(long accNo, String type, double amount)
throws Exception {
        Connection conn = DBConnUtil.getConnection();
        String insertTxn = "INSERT INTO transactions (account_id,
transaction_type, amount) VALUES (?, ?, ?)";
        PreparedStatement ps = conn.prepareStatement(insertTxn);
        ps.setLong(1, accNo);
        ps.setString(2, type);
```

```
        ps.setDouble(3, amount);
        ps.executeUpdate();
        conn.close();
    }

    @Override
    public Accounts getAccountDetails(long accountNumber) throws Exception {
        Connection conn = DBConnUtil.getConnection();

        String sql = "SELECT a.account_id, a.account_type, a.balance, " +
                "c.customer_id, c.first_name, c.last_name, c.email,
c.phone_number, c.address " +
                "FROM accounts a JOIN customers c ON a.customer_id =
c.customer_id " +
                "WHERE a.account_id = ?";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setLong(1, accountNumber);
        ResultSet rs = ps.executeQuery();

        if (rs.next()) {
            Accounts acc = new Accounts();
            acc.setAccountNumber(String.valueOf(rs.getLong("account_id")));
            acc.setAccountType(rs.getString("account_type"));
            acc.setAccountBalance(rs.getDouble("balance"));

            Customer cust = new Customer(
                    rs.getString("customer_id"),
                    rs.getString("first_name"),
                    rs.getString("last_name"),
                    rs.getString("email"),
                    rs.getString("phone_number"),
                    rs.getString("address")
            );
            acc.setCustomer(cust);

            return acc;
        }

        throw new Exception("Account not found.");
    }

    @Override
    public List<Transaction> getTransactions(long accountNumber, Timestamp
fromDate, Timestamp toDate) throws Exception {
        List<Transaction> transactions = new ArrayList<>();
        Connection conn = DBConnUtil.getConnection();
        String query = "SELECT * FROM transactions WHERE account_id = ? AND
transaction_date BETWEEN ? AND ?";
        PreparedStatement ps = conn.prepareStatement(query);
        ps.setLong(1, accountNumber);
        ps.setTimestamp(2, fromDate);
        ps.setTimestamp(3, toDate);

        ResultSet rs = ps.executeQuery();

        while (rs.next()) {
            Transaction t = new Transaction();
```

```
            t.setDescription(rs.getString("transaction_type"));
            t.setTransactionType(rs.getString("transaction_type"));
            t.setTransactionAmount(rs.getDouble("amount"));

t.setDateTime(rs.getTimestamp("transaction_date").toLocalDateTime());
            transactions.add(t);
        }

        conn.close();
        return transactions;
    }
}
```

CustomerServiveProviderImpl.java(bean package):

```
package bean;

import service.ICustomerServiceProvider;
import exception.InsufficientFundException;
import exception.OverDraftLimitExceededException;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;
import bean.Transaction;

public class CustomerServiceProviderImpl implements ICustomerServiceProvider
{

    protected List<Accounts> accountList = new ArrayList<>();
    protected List<Transaction> transactionList = new ArrayList<>();

    protected Accounts findAccount(long accNo) {
        for (Accounts acc : accountList) {
            if (acc.getAccountNumber().equals(String.valueOf(accNo))) {
                return acc;
            }
        }
        return null;
    }

    public double get_account_balance(long accountNumber) {
        Accounts acc = findAccount(accountNumber);
        if (acc != null) return acc.getAccountBalance();
        System.out.println("Account not found.");
        return -1;
    }

    public double deposit(long accountNumber, float amount) {
        Accounts acc = findAccount(accountNumber);
        if (acc != null) {
```

```
            acc.deposit(amount);
            transactionList.add(new Transaction(acc, "Deposit", "Deposit",
amount));
            return acc.getAccountBalance();
        }
        System.out.println("Account not found.");
        return -1;
    }

    public double withdraw(long accountNumber, float amount)
            throws InsufficientFundException, OverDraftLimitExceededException
{
        Accounts acc = findAccount(accountNumber);
        if (acc != null) {
            acc.withdraw(amount); // may throw exceptions
            transactionList.add(new Transaction(acc, "Withdrawal",
"Withdraw", amount));
            return acc.getAccountBalance();
        }
        System.out.println("Account not found.");
        return -1;
    }

    public void transfer(long fromAccount, int toAccount, float amount)
            throws InsufficientFundException, OverDraftLimitExceededException
{
        Accounts from = findAccount(fromAccount);
        Accounts to = findAccount(toAccount);
        if (from != null && to != null) {
            if (from.getAccountBalance() >= amount) {
                from.withdraw(amount);
                to.deposit(amount);
                transactionList.add(new Transaction(from, "Transfer to " +
to.getAccountNumber(), "Transfer", amount));
                transactionList.add(new Transaction(to, "Transfer from " +
from.getAccountNumber(), "Transfer", amount));
                System.out.println("Transfer successful.");
            } else {
                throw new InsufficientFundException("Insufficient funds.");
            }
        } else {
            System.out.println("One or both accounts not found.");
        }
    }

    public void printAccountDetails(long accountNumber) {
        Accounts acc = findAccount(accountNumber);
        if (acc != null) acc.printAccountInfo();
        else System.out.println("Account not found.");
    }

    public List<Transaction> getTransations(long accountNumber,
java.util.Date fromDate, java.util.Date toDate) {
        return transactionList.stream()
                .filter(t ->
t.getAccount().getAccountNumber().equals(String.valueOf(accountNumber)))
                .filter(t -> {
```

```
                    Date txnDate =
java.sql.Timestamp.valueOf(t.getDateTime());
                    return !txnDate.before(fromDate) &&
!txnDate.after(toDate);
                })
                .collect(Collectors.toList());
    }
    @Override
    public List<Transaction> getTransactions(long accountNumber, Date
fromDate, Date toDate) {
        System.out.println("Transaction retrieval is not supported in local
implementation.");
        return new ArrayList<>(); // or return Collections.emptyList();
    }
}
```

BankServiceProviderImpl.java:

```
package bean;

import service.IBankServiceProvider;
import java.util.List;

public class BankServiceProviderImpl extends CustomerServiceProviderImpl
implements IBankServiceProvider {
    private String branchName = "Main Branch";
    private String branchAddress = "123 Bank Street";

    @Override
    public void create_account(Customer customer, long accNo, String accType,
double balance) {
        Accounts account = null;
        String accNumStr = String.valueOf(accNo);

        switch (accType.toLowerCase()) {
            case "savings":
                if (balance < 500) {
                    System.out.println("Minimum balance for Savings Account
is ₹500.");
                    return;
                }
                account = new SavingsAccount(accNumStr, "Savings", balance,
customer, 4.5);
                break;

            case "current":
                account = new CurrentAccount(accNumStr, "Current", balance,
customer, 10000.0);
                break;

            case "zero":
            case "zerobalance":
                account = new ZeroBalanceAccount(accNumStr, "ZeroBalance",
balance, customer);
```

```
                break;

            default:
                System.out.println("Invalid account type.");
                return;
        }

        accountList.add(account);
        Accounts.setLastAccNo(accNo);

        System.out.println("Account created successfully. Account Number: " +
account.getAccountNumber());
    }

    @Override
    public Accounts getAccountDetails(long accountNumber) {
        for (Accounts acc : accountList) {
            if (Long.parseLong(acc.getAccountNumber()) == accountNumber) {
                return acc;
            }
        }
        System.out.println("Account not found.");
        return null;
    }

    @Override
    public List<Accounts> listAccounts() {
        return accountList;
    }

    @Override
    public void calculateInterest() {
        for (Accounts acc : accountList) {
            acc.calculateInterest();
        }
    }

    @Override
    public void printAccountDetails(long accountNumber) {
        Accounts acc = getAccountDetails(accountNumber);
        if (acc != null) {
            acc.printAccountInfo();
        }
    }
}
```

UTIL PACKAGE:

DBConnUtil.java:

```
package util;

import java.io.IOException;
import java.sql.Connection;
```

```java
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnUtil {

    private static final String fileName = "db.properties";

    public static Connection getConnection() {
        Connection con = null;
        String connString = null;

        try {
            connString = DBPropertyUtil.getConnectionString(fileName);
        } catch (IOException e) {
            System.out.println("Connection String Creation Failed");
            e.printStackTrace();
        }

        if (connString != null) {
            try {
                con = DriverManager.getConnection(connString);
            } catch (SQLException e) {
                System.out.println("Error While Establishing DB
Connection...");
                e.printStackTrace();
            }
        }

        return con;
    }
}
```

DBPropertyUtil.java:

```java
package util;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class DBPropertyUtil {

    public static String getConnectionString(String fileName) throws
IOException {
        String connStr;
        Properties props = new Properties();

        FileInputStream fis = new FileInputStream("resources/" + fileName);
        props.load(fis);

        String user = props.getProperty("user");
        String password = props.getProperty("password");
        String protocol = props.getProperty("protocol");
        String system = props.getProperty("system");
```

```
        String port = props.getProperty("port");
        String database = props.getProperty("database");

        connStr = protocol + "//" + system + ":" + port + "/" + database +
                "?user=" + user + "&password=" + password;

        return connStr;
    }
}
```

db.properties:

```
protocol=jdbc:mysql:
system=localhost
port=3306
database=hmbank
user=root
password=Sandhiya_21
```

EXCEPTION PACKAGE:

InSufficientFundException.java:

```
package exception;

public class InsufficientFundException extends Exception {
    public InsufficientFundException(String message) {
        super(message);
    }
}
```

InValidAccountException.java:

```
package exception;

public class InvalidAccountException extends Exception {
    public InvalidAccountException(String message) {
        super(message);
    }
}
```

OverDraftLimitExceededException.java:

```
package exception;

public class OverDraftLimitExceededException extends Exception {
    public OverDraftLimitExceededException(String message) {
```

```
        super(message);
    }
}
```

MAIN:

BankApp.java:

```
package app;

import bean.Customer;
import bean.Transaction;
import repository.BankRepositoryImpl;
import service.IBankRepository;

import java.sql.Date;
import java.time.LocalDate;
import java.util.List;
import java.util.Scanner;
import java.sql.Timestamp;
import java.time.LocalDateTime;

public class BankApp {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        IBankRepository bankRepo = new BankRepositoryImpl();

        while (true) {
            try {
                System.out.println("\n=========  BANK MENU =========");
                System.out.println("1. Create Account");
                System.out.println("2. Deposit");
                System.out.println("3. Withdraw");
                System.out.println("4. Get Account Balance");
                System.out.println("5. Transfer Funds");
                System.out.println("6. Get Account Details");
                System.out.println("7. List Accounts");
                System.out.println("8. Get Transactions");
                System.out.println("9. Exit");
                System.out.print("Enter your choice: ");
                int choice = sc.nextInt();
                sc.nextLine();

                switch (choice) {
                    case 1: // Create Account
                        System.out.print("Enter Customer ID: ");
                        String id = sc.nextLine();
                        System.out.print("First Name: ");
                        String fName = sc.nextLine();
                        System.out.print("Last Name: ");
                        String lName = sc.nextLine();

                        String email;
```

```
                        while (true) {
                            System.out.print("Email Address: ");
                            email = sc.nextLine();
                            if (email.matches("^(.+)@(.+)$")) break;
                            System.out.println("Invalid email. Try again.");
                        }

                        String phone;
                        while (true) {
                            System.out.print("Phone Number (10 digits): ");
                            phone = sc.nextLine();
                            if (phone.matches("\\d{10}")) break;
                            System.out.println("Invalid phone number. Try
again.");
                        }

                        System.out.print("Address: ");
                        String address = sc.nextLine();

                        Customer customer = new Customer(id, fName, lName,
email, phone, address);

                        System.out.print("Choose Account Type (Savings /
Current / ZeroBalance): ");
                        String accType = sc.nextLine();
                        System.out.print("Initial Deposit Amount: ");
                        float bal = sc.nextFloat();
                        sc.nextLine();

                        long accNo = System.currentTimeMillis(); // temporary
account number generation
                        bankRepo.createAccount(customer, accNo, accType,
bal);
                        System.out.println(" Account created successfully!
Your Account Number is: " + accNo);
                        break;

                    case 2: // Deposit
                        System.out.print("Enter Account Number: ");
                        long depAcc = sc.nextLong();
                        System.out.print("Enter amount to deposit: ");
                        float depAmount = sc.nextFloat();
                        double updatedBal = bankRepo.deposit(depAcc,
depAmount);
                        System.out.println("Deposit successful. Updated
Balance: ₹" + updatedBal);
                        break;

                    case 3: // Withdraw
                        System.out.print("Enter Account Number: ");
                        long witAcc = sc.nextLong();
                        System.out.print("Enter amount to withdraw: ");
                        float witAmount = sc.nextFloat();
                        double newBal = bankRepo.withdraw(witAcc, witAmount);
                        System.out.println("Withdrawal successful. New
Balance: ₹" + newBal);
                        break;
```

```
                    case 4: // Get Balance
                        System.out.print("Enter Account Number: ");
                        long balAcc = sc.nextLong();
                        double balance = bankRepo.getAccountBalance(balAcc);
                        System.out.println("Current Balance: ₹" + balance);
                        break;

                    case 5: // Transfer
                        System.out.print("From Account Number: ");
                        long fromAcc = sc.nextLong();
                        System.out.print("To Account Number: ");
                        long toAcc = sc.nextLong();
                        System.out.print("Amount to transfer: ");
                        float amount = sc.nextFloat();
                        bankRepo.transfer(fromAcc, toAcc, amount);
                        System.out.println("Transfer successful.");
                        break;

                    case 6: // Get Account Details
                        System.out.print("Enter Account Number: ");
                        long accDetails = sc.nextLong();

System.out.println(bankRepo.getAccountDetails(accDetails));
                        break;

                    case 7: // List All Accounts
                        bankRepo.listAccounts().forEach(a ->
a.printAccountInfo());
                        break;
                    case 8: // Get Transactions
                        System.out.print("Enter Account Number: ");
                        long accTx = sc.nextLong();
                        sc.nextLine();

                        System.out.print("From Date (yyyy-mm-dd): ");
                        String fromDateStr = sc.nextLine();
                        LocalDateTime fromDateTime =
LocalDate.parse(fromDateStr).atStartOfDay();
                        Timestamp from = Timestamp.valueOf(fromDateTime);

                        System.out.print("To Date (yyyy-mm-dd): ");
                        String toDateStr = sc.nextLine();

                        LocalDateTime toDateTime =
LocalDate.parse(toDateStr).atTime(23, 59, 59);
                        Timestamp to = Timestamp.valueOf(toDateTime);

                        List<Transaction> txns =
bankRepo.getTransactions(accTx, from, to);
                        if (txns.isEmpty()) {
                            System.out.println("ℹ No transactions found
between the given dates.");
                        } else {
                            System.out.println("Transaction History:");
                            for (Transaction t : txns) {
```

```
System.out.println("────────────────────────────────");
                                System.out.println("Type       : " +
t.getTransactionType());
                                System.out.println("Amount     : ₹" +
t.getTransactionAmount());
                                System.out.println("Date & Time: " +
t.getDateTime());
                                System.out.println("Note       : " +
t.getDescription());
                            }

System.out.println("────────────────────────────────");
                        }
                        break;



                    case 9:
                        System.out.println("Thank you for using the Banking
System!");
                        sc.close();
                        System.exit(0);

                    default:
                        System.out.println("Invalid choice. Try again.");
                }
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }
}
```

OUTPUT:

1.Create Account:

```
=========  BANK MENU =========
1. Create Account
2. Deposit
3. Withdraw
4. Get Account Balance
5. Transfer Funds
6. Get Account Details
7. List Accounts
8. Get Transactions
9. Exit
Enter your choice: 1
Enter Customer ID: 4
First Name: raj
Last Name: kumar
Email Address: raj@gmail.com
Phone Number (10 digits): 9807564123
Address: 563,gopal street,chennai
Choose Account Type (Savings / Current / ZeroBalance): current
Initial Deposit Amount: 3000
 Account created successfully! Your Account Number is: 1744338704424
```

SQL OUTPUT:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cor

customer_id	first_name	last_name	dob	email	phone_number	address
1	abi	ram	NULL	abi@gmail.com	7654390811	124,nehru st,chennai
2	sai	lal	NULL	sai@gmail.com	7654321890	167,rajaji street,mumbai
3	swathi	paraman	NULL	swa@gmail.com	6578901121	145,kali st, chennai
4	raj	kumar	NULL	raj@gmail.com	9807564123	563,gopal street,chennai
NULL	NULL	NULL	NULL	NULL	NULL	NULL

2.Deposit:

```
Enter your choice: 2
Enter Account Number: 1744338704424
Enter amount to deposit: 3000
Deposit successful. Updated Balance: ₹6000.0
```

SQL OUTPUT:

Result Grid | Filter Rows: | Edit:

account_id	customer_id	account_type	balance
1744302408447	1	Savings	5000
1744302623540	2	Savings	6000
1744310272016	3	Savings	3000
1744338704424	4	Current	6000
NULL	NULL	NULL	NULL

3.Withdraw:

```
Enter your choice: 3
Enter Account Number: 1744338704424
Enter amount to withdraw: 2000
Withdrawal successful. New Balance: ₹4000.0
```

SQL OUTPUT:

Result Grid | Filter Rows: | Edit:

account_id	customer_id	account_type	balance
1744302408447	1	Savings	5000
1744302623540	2	Savings	6000
1744310272016	3	Savings	3000
1744338704424	4	Current	4000
NULL	NULL	NULL	NULL

4.Get account balance:

```
Enter your choice: 4
Enter Account Number: 1744338704424
Current Balance: ₹4000.0
```

5.Transfer Funds:

```
Enter your choice: 5
From Account Number: 1744338704424
To Account Number: 1744310272016
Amount to transfer: 1000
Transfer successful.
```

SQL OUTPUT AFTER TRANSFERING FUND:

Result Grid | Filter Rows: | Edit:

account_id	customer_id	account_type	balance
1744302408447	1	Savings	5000
1744302623540	2	Savings	6000
1744310272016	3	Savings	4000
1744338704424	4	Current	3000
NULL	NULL	NULL	NULL

6.Get Account Details:

```
Enter your choice: 6
Enter Account Number: 1744338704424
Account Details:
Account Number: 1744338704424
Name           : raj kumar
Balance        : 3000.0
```

7.List Accounts:

```
Enter your choice: 7
Account Number: 1744302408447
Account Type   : Savings
Balance        : $5000.0
Name           : abi ram
Email          : abi@gmail.com
Phone          : 7654390811
Address        : 124,nehru st,chennai
Account Number: 1744302623540
Account Type   : Savings
Balance        : $6000.0
Name           : sai lal
Email          : sai@gmail.com
Phone          : 7654321890
Address        : 167,rajaji street,mumbai
```

8.Get Transactions:

```
Enter your choice: 8
Enter Account Number: 1744338704424
From Date (yyyy-mm-dd): 2025-04-11
To Date (yyyy-mm-dd): 2025-04-11
📄 Transaction History:
────────────────────────────────
Type        : Deposit
Amount      : ₹3000.0
Date & Time: 2025-04-11T08:06:03
Note        : Deposit
────────────────────────────────
Type        : Withdrawal
Amount      : ₹2000.0
Date & Time: 2025-04-11T08:07:16
Note        : Withdrawal
────────────────────────────────
Type        : Withdrawal
Amount      : ₹1000.0
Date & Time: 2025-04-11T08:09:52
Note        : Withdrawal
────────────────────────────────
Type        : Transfer
Amount      : ₹1000.0
Date & Time: 2025-04-11T08:09:52
Note        : Transfer
────────────────────────────────
```

9. Exit:

```
Enter your choice: 9
✅ Thank you for using the Banking System!

Process finished with exit code 0
```