**DATABASE DESIGN FOR THE BANKING AND FINANCE INDUSTRY**

**GROUP 5 PROJECT**

**INTRODUCTION**

In the dynamic landscape of the banking and finance industry, adept data management stands as a cornerstone for facilitating smooth operations, ensuring regulatory adherence, and elevating customer service standards. This database design offers a structured framework to house, organize, and retrieve vital financial data. The database manages customer accounts, transactions, loans, and financial reporting.

**GOAL**

The primary objective of this database design project is to develop a comprehensive system that efficiently manages diverse aspects of Sparks Bank. This encompasses the effective administration of customer accounts, secure transaction processing, loan management, and the generation of precise financial reports. Our goal is to bolster operational efficiency, guarantee data integrity, and empower informed decision-making. Through meticulous design, extensive security protocols, and robust data validation mechanisms, we aspire to establish a reliable platform that underpins the core functions of Spark Bank while upholding the highest standards of integrity and accuracy.

**TOOL**

Microsoft SQL Server Management Studio.

**Entity-Relationship Model (ERM)**

Central to the database design process is the creation of an Entity-Relationship Model (ERM), which serves as a blueprint for defining the structure and relationships between various entities within the system.

Key entities in this context may include

* Accounts
* Transactions
* Loans
* Employees

Through the ERM, we delineate the attributes and associations of each entity, elucidating how they interact and contribute to the overarching functionality of the system.

**TABLE DESIGNS**

1. **ACCOUNTS TABLE:**

**CODE**

--Database Design and Normalization

--Creating Tables for Customer Accounts, transactions, and Loans, since it is a banking Database added an extra table which is for employees

-- Accounts Table

CREATE TABLE accounts (

account\_id INT PRIMARY KEY,

account\_number INT,

account\_name VARCHAR(100),

account\_type VARCHAR(50),

balance DECIMAL(18, 2),

opening\_date DATE,

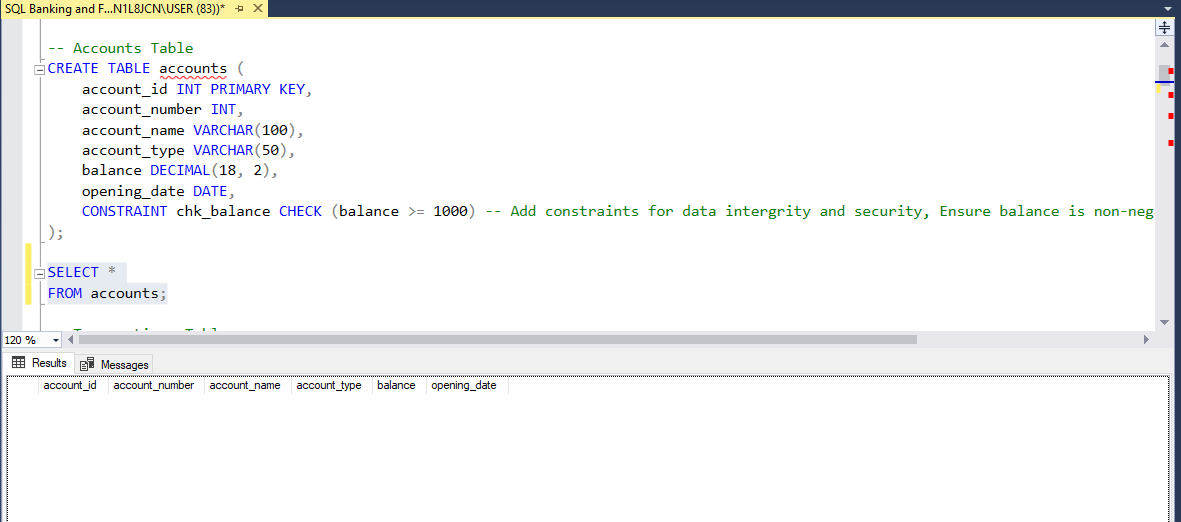
CONSTRAINT chk\_balance CHECK (balance >= 1000) -- Add constraints for data intergrity and security, Ensure balance is non-negative

);

SELECT \*

FROM accounts;

**RESULTS**



* **About:** This code sets up a database table called "accounts" to store information about different accounts. Each account has an ID, number, name, type, balance, and opening date. It ensures that account balances are always at least 1000 by adding a constraint. Finally, it retrieves all records from the "accounts" table.
* **Attributes:**

1. account\_id: This is the primary key and serves as a unique identifier for each account.
2. account\_number: This attribute stores the account number.
3. account\_name: This attribute stores the name associated with each account.
4. account\_type: This attribute stores the type of each account (savings, current).
5. balance: This attribute stores the balance of each account.
6. opening\_date: This attribute stores the date when each account was opened.

* **Data Types:** Varchar for account name and account type, Integer for account ID and account number, Decimal for balance, and Date for Opening date.

1. **TRANSACTION TABLE:**

**CODE**

-- Transactions Table

CREATE TABLE transactions (

transaction\_id INT PRIMARY KEY,

account\_id INT FOREIGN KEY REFERENCES accounts(account\_id),

transaction\_type VARCHAR(50),

amount DECIMAL(18, 2),

transaction\_date DATE,

transanction\_time TIME,

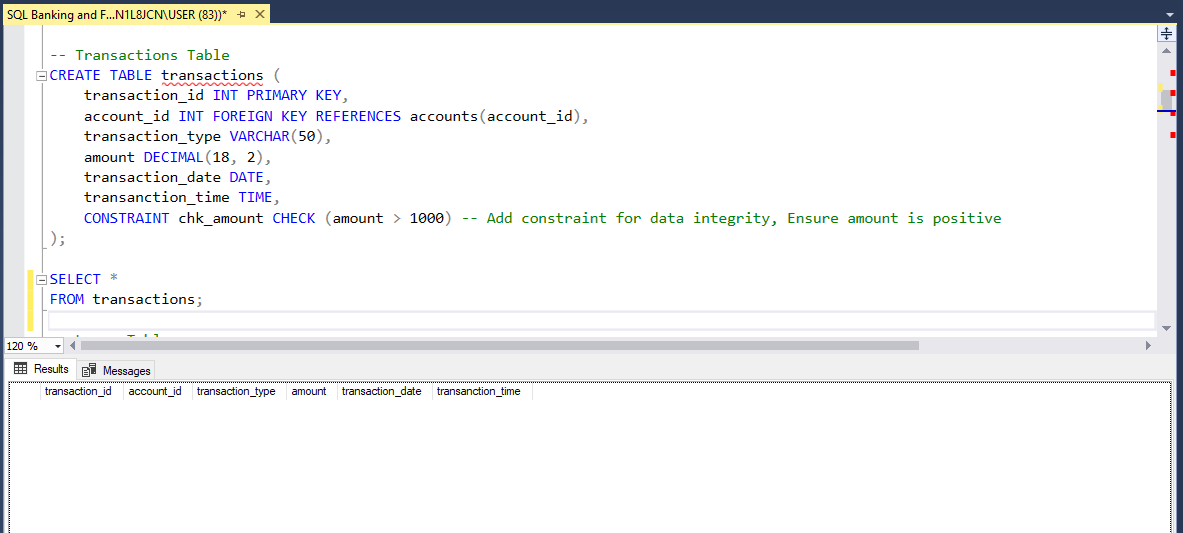
CONSTRAINT chk\_amount CHECK (amount > 1000) -- Add constraint for data integrity, Ensure amount is positive

);

SELECT \*

FROM transactions;

**RESULT**

****

* **About:** This code establishes a table structure for recording transactions, including relevant details such as transaction type, amount, date, and time. It also maintains data integrity by enforcing constraints on the transaction amount. Additionally, it establishes a relationship with the "accounts" table through the foreign key constraint on the account\_id attribute.
* **Attributes:**

1. transaction\_id: This is the primary key and serves as a unique identifier for each transaction.
2. account\_id: This attribute is a foreign key referencing the account\_id attribute in the "accounts" table, establishing a relationship between transactions and the respective accounts involved.
3. transaction\_type: This attribute stores the type of each transaction (deposit, withdrawal, transfers).
4. amount: This attribute stores the amount of each transaction.
5. transaction\_date: This attribute stores the date when each transaction occurred.
6. transaction\_time: This attribute stores the time when each transaction occurred.

* **Foreign Key:**

account\_id: This attribute is a foreign key that references the account\_id attribute in the "accounts" table, linking transactions to specific accounts.

* **Data Types:** Integer for transaction ID and account ID, Varchar for transaction type, Decimal for amount, Date for transaction date, and Time for transaction time.
* **Constraints:**

chk\_amount: This constraint ensures that the amount for each transaction is greater than 1000, enforcing data integrity by ensuring positive transaction amounts.

1. **LOAN TABLE :**

**CODE**

-- Loans Table

CREATE TABLE loans (

loan\_id INT PRIMARY KEY,

account\_id INT FOREIGN KEY REFERENCES accounts(account\_id),

loan\_type VARCHAR(50),

amount DECIMAL(18, 2),

interest\_rate DECIMAL(5, 2),

Payment\_schedule Varchar(50),

installment\_amount DECIMAL(18, 2),

due\_date DATE,

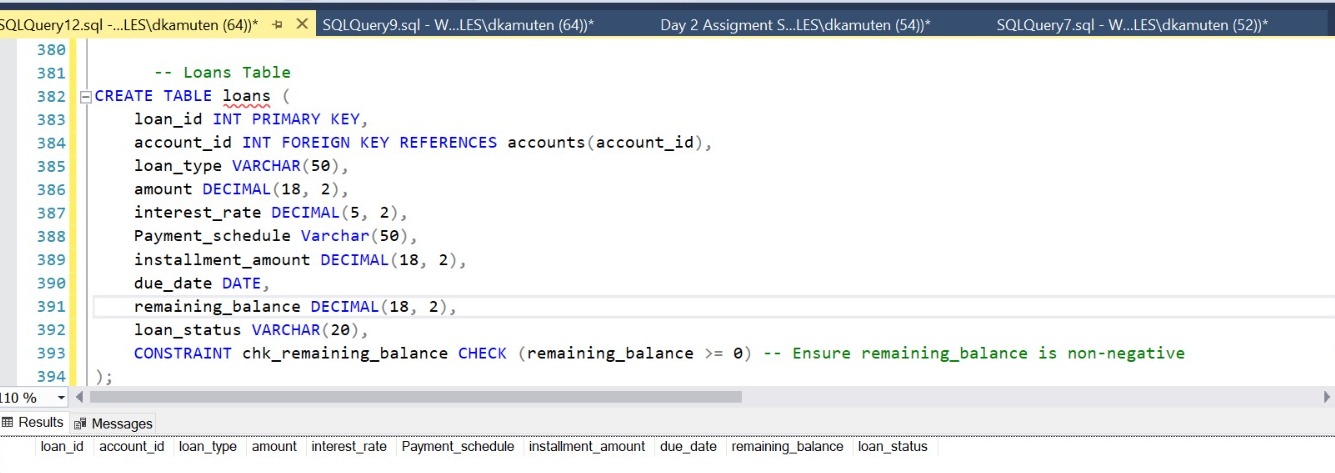
remaining\_balance DECIMAL(18, 2),

loan\_status VARCHAR(20),

CONSTRAINT chk\_remaining\_balance CHECK (remaining\_balance >= 0) -- Ensure remaining\_balance is non-negative

);

**RESULTS**



* **About:** the code establishes a structured way to store and manage information related to loans, including loan type, amount, interest rate, payment schedule, and status. Additionally, it enforces a constraint to maintain data integrity regarding the remaining balance on loans and retrieves all loan records from the database.
* **Attributes:**

1. loan\_id: This attribute serves as the primary key and provides a unique identifier for each loan.
2. account\_id: This attribute is a foreign key referencing the account\_id attribute in the "accounts" table, establishing a relationship between loans and the respective accounts involved.
3. loan\_type: This attribute stores the type of each loan (personal loan, Auto loan e.t.c).
4. amount: This attribute stores the initial amount of the loan.
5. interest\_rate: This attribute stores the interest rate associated with the loan.
6. Payment\_schedule: This attribute stores information about the payment schedule for the loan.
7. installment\_amount: This attribute stores the amount of each installment payment.
8. due\_date: This attribute stores the due date for each installment payment.
9. remaining\_balance: This attribute stores the remaining balance of the loan.
10. status: This attribute stores the status of the loan (active, completed).

* **Foreign Key:**

account\_id: This attribute is a foreign key that references the account\_id attribute in the "accounts" table, linking loans to specific accounts.

* **Data Types:**  Integer for loan ID and account ID, Varchar for loan type, payment schedule, and status, Decimal for amount, Interest rate, installment amount, and remaining balance, Date for due date.
* **Constraints:**

chk\_remaining\_balance: This constraint ensures that the remaining balance of the loan is non-negative, maintaining data integrity.

1. **EMPLOYEES TABLE:**

**CODE**

--Employee table

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(100),

years\_of\_service INT,

department VARCHAR(50)

);

SELECT \*

FROM employees;

**RESULT**

****

**About:** this code establishes a structured way to store basic information about employees, including their names, years of service, and department affiliation.

1. **Attributes:**  
     
   employee\_id: This is the primary key and serves as a unique identifier for each employee.
2. employee\_name: This column stores the name of each employee
3. years\_of\_service: This column stores the number of years of service for each employee
4. department: This column stores the department or division within the organization to which each employee belongs

**Data Type:** Varchar for employee name and department, Integer for employee ID and years of service.

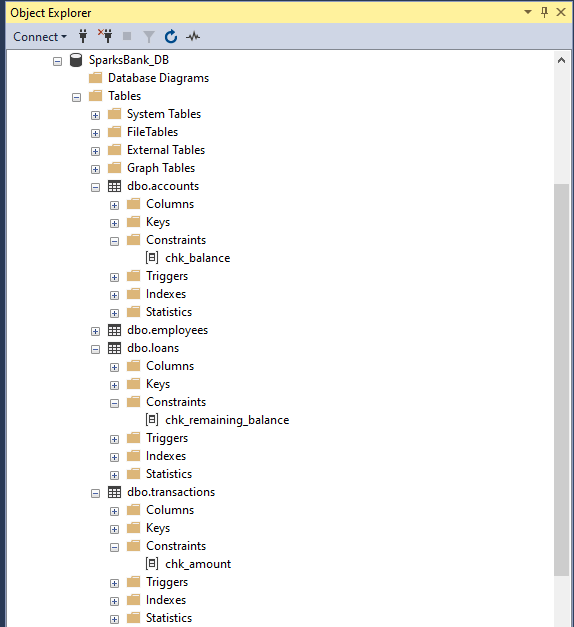
**NORMALIZATION**

Normalization has been applied to the database design, following the guidelines of the third normal form (3NF). This ensures that each table is organized to remove transitive dependencies, guaranteeing that every non-prime attribute is entirely functionally dependent on the primary key.

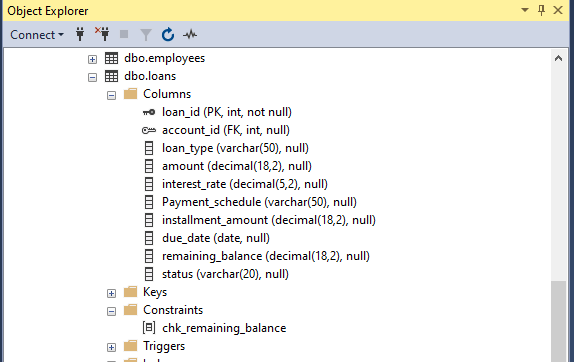
**CONSTRAINTS**

Constraints are employed in the database to uphold referential integrity among interconnected tables. Primary key constraints are enforced to guarantee uniqueness, while data type constraints are utilized to maintain data integrity.

**DATA TYPE AND CONSTRAINTS**

****

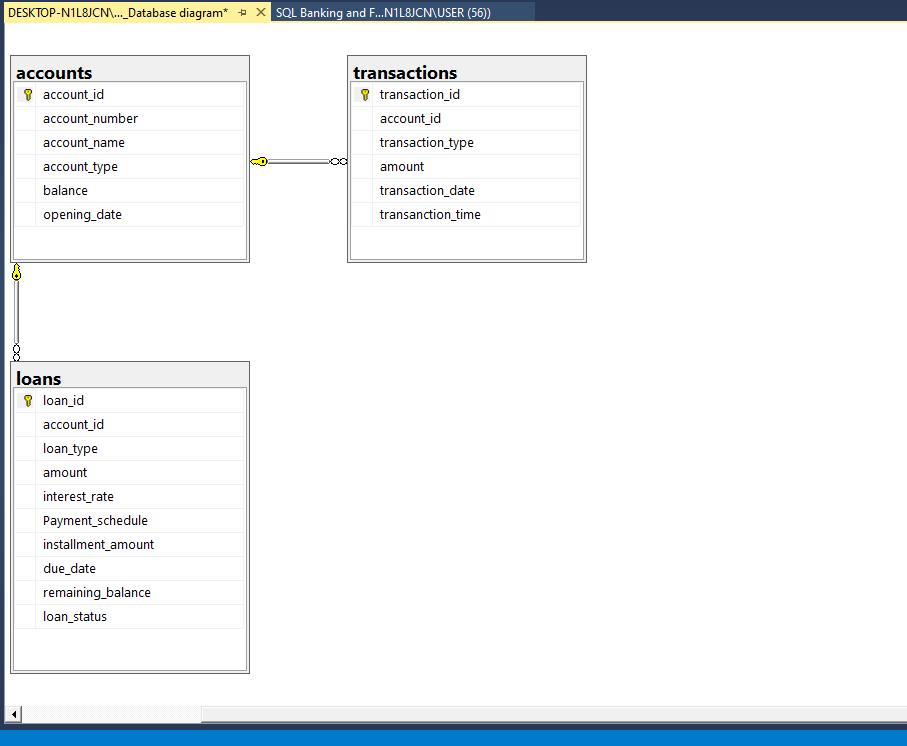
These constraints serve to maintain data consistency and reliability, minimizing errors and ensuring the accuracy and security of the database system.

****

Each column's data type was selected meticulously, weighing factors like storage efficiency and data accuracy to ensure optimal performance.

**Database Diagram:**

A database diagram was produced to visually outline the structure of the database, allowing for a clearer comprehension of how different tables are interconnected and related. This graphical representation serves as a helpful tool in understanding the underlying architecture of the database system, aiding in the analysis and management of data by providing a visual guide to the relationships between various components within the database schema.



**Populating Tables:**

* To ensure comprehensive testing, a varied and realistic set of sample data was created and inserted into the database tables.
* This process included the Accounts table, Transaction table, Loan table, and Employee table, with each containing a maximum of 20 records except the Employee table which contains 7 records.
* By populating these tables with meaningful data, the functionality and efficiency of the database could be thoroughly evaluated, ensuring that it meets the requirements and effectively handles real-world scenarios in the banking and finance sector.

**CODES**

-- Populate accounts table

INSERT INTO accounts (account\_id,account\_number,account\_name,account\_type,balance,opening\_date)

VALUES

('1','1001','John Richard','Savings','100000.00','2020-01-01'),

('2','1002','Jane Smith','Current','10000.00','2019-05-15'),

('3','1003','Michael Jackson','Savings','7500.00','2021-03-10'),

('4','1004','Emily Brown','Current','15000.00','2018-11-20'),

('5','1005','David Wilson','Savings','3000.00','2022-07-05'),

('6','1006','Sarah Cheng','Savings','20000.00','2020-09-15'),

('7','1007','Robert Miller','Savings','6000.00','2019-02-28'),

('8','1008','Jessica Taylor','Current','18000.00','2023-01-10'),

('9','1009','Christopher Bradford','Savings','8500.00','2020-06-20'),

('10','1010','Amanda Mendes','Current','12000.00','2021-09-30'),

('11','1011','James Cranes','Savings','4000.00','2019-11-15'),

('12','1012','Jennifer Lopez','Current','22000.00','2022-04-25'),

('13','1013','William King','Savings','6800.00','2020-03-03'),

('14','1014','Stephanie Lewis','Current','14000.00','2023-07-12'),

('15','1015','Daniel Adams', 'Savings','2000.00','2021-01-20'),

('16','1016','Elizabeth Hall','Current','25000.00','2019-08-08'),

('17','1017','Joseph Wright','Savings','9000.00','2022-05-10'),

('18','1018','Mary Clark','Current','16000.00','2020-10-18'),

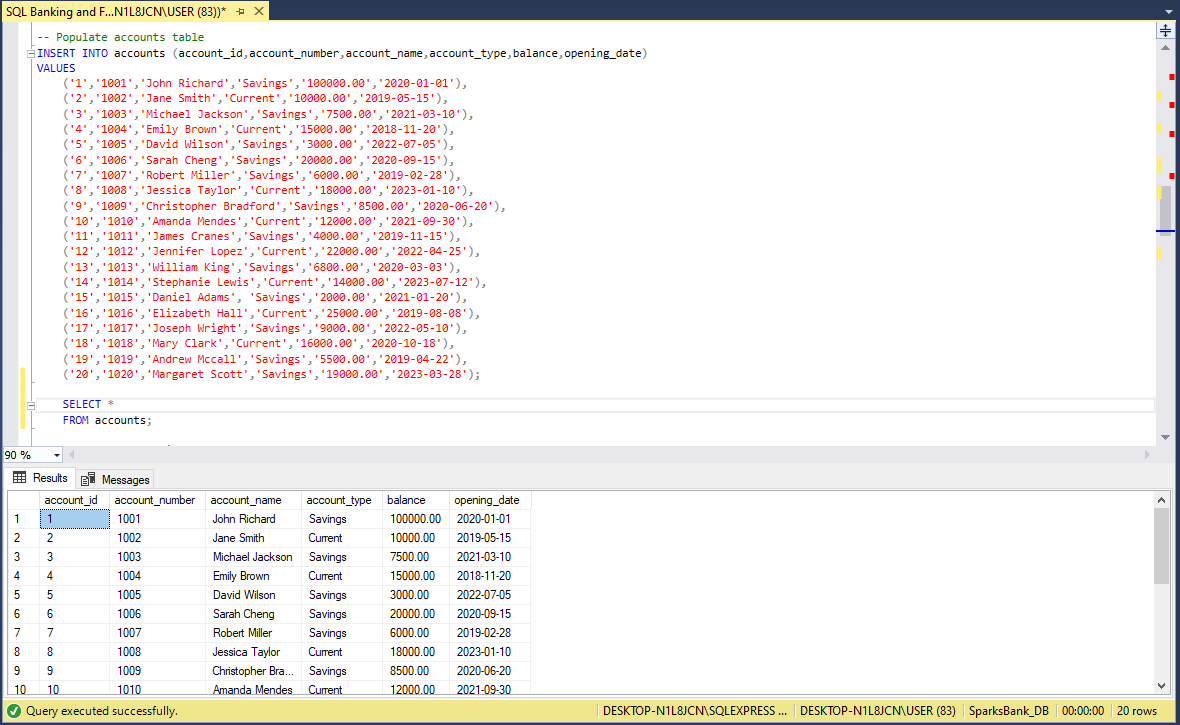
('19','1019','Andrew Mccall','Savings','5500.00','2019-04-22'),

('20','1020','Margaret Scott','Savings','19000.00','2023-03-28');

SELECT \*

FROM accounts;

**RESULT**

****

**CODES**

-- Populate Transactions Table

INSERT INTO transactions (transaction\_id, account\_id, transaction\_type, amount, transaction\_date,transanction\_time)

VALUES

(1,1, 'Deposit', 5000.00, '2024-03-01', '08:00:00'),

(2,2, 'Withdrawal', 2000.00, '2024-03-02', '10:30:00'),

(3,3, 'Deposit', 7000.00, '2024-03-03', '13:45:00'),

(4,4, 'Withdrawal', 1500.00, '2024-03-04', '11:20:00'),

(5,5, 'Deposit', 3000.00, '2024-03-05', '09:15:00'),

(6,6, 'Withdrawal', 2500.00, '2024-03-06', '14:00:00'),

(7,7, 'Deposit', 4000.00, '2024-03-07', '12:30:00'),

(8,8, 'Withdrawal', 3500.00, '2024-03-08', '10:45:00'),

(9,9, 'Deposit', 6000.00, '2024-03-09', '08:20:00'),

(10,1, 'Withdrawal', 18000.00, '2024-03-10', '15:10:00'),

(11,11, 'Deposit', 8000.00, '2024-03-11', '11:00:00'),

(12,1, 'Withdrawal', 5000.00, '2024-03-12', '09:40:00'),

(13,13, 'Deposit', 9000.00, '2024-03-13', '13:20:00'),

(14,14, 'Withdrawal', 1300.00, '2024-03-14', '10:55:00'),

(15,1, 'Deposit', 12000.00, '2024-03-15', '14:25:00'),

(16,15, 'Withdrawal', 3200.00, '2024-03-16', '09:05:00'),

(17,1, 'Deposit', 11000.00, '2024-03-17', '12:15:00'),

(18,16, 'Withdrawal', 2800.00, '2024-03-18', '13:35:00'),

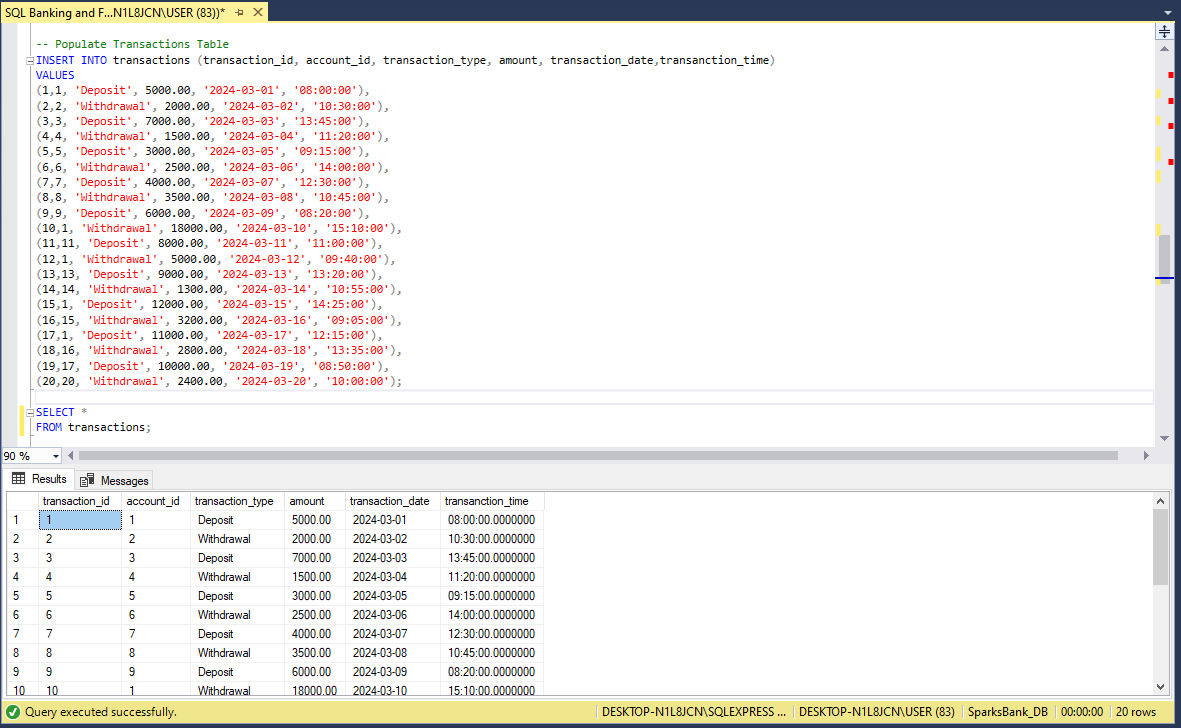
(19,17, 'Deposit', 10000.00, '2024-03-19', '08:50:00'),

(20,20, 'Withdrawal', 2400.00, '2024-03-20', '10:00:00');

SELECT \*

FROM transactions;

**RESULT**

****

**CODES**

-- Populate loans table

INSERT INTO loans (loan\_id, account\_id, loan\_type, amount, interest\_rate, Payment\_schedule, installment\_amount, due\_date, remaining\_balance,loan\_status)

VALUES

(1,1,'Personal Loan', 5000.00, 5.25, 'Monthly', 250.00, '2024-04-01', 5000.00, 'Completed'),

(2,2, 'Auto Loan',10000.00, 4.75, 'weekly', 350.00, '2024-04-15', 10000.00, 'Active'),

(3, 3, 'Home Loan',150000.00, 3.50, 'Monthly', 1200.00, '2024-05-01', 150000.00, 'Active'),

(4, 4, 'Education Loan',20000.00, 6.00,'Monthly', 400.00, '2024-04-10', 20000.00, 'Active'),

(5, 5, 'Business Loan',75000.00, 5.75, 'Monthly', 800.00, '2024-04-05', 75000.00, 'Active'),

(6, 6, 'Personal Loan', 8000.00, 4.25, 'weekly', 300.00, '2024-04-20', 8000.00, 'Active'),

(7, 7, 'Auto Loan', 12000.00, 3.75, 'Monthly', 450.00, '2024-04-25', 12000.00, 'Active'),

(8, 8, 'Home Loan', 180000.00, 4.00, 'weekly', 1300.00, '2024-05-05', 180000.00, 'Active'),

(9, 9, 'Education Loan', 25000.00, 5.50, 'Monthly', 500.00, '2024-04-30', 25000.00, 'Completed'),

(10,10, 'Business Loan', 100000.00, 6.25, 'Monthly', 1200.00, '2024-04-15', 100000.00, 'Active'),

(11,11, 'Personal Loan', 6000.00, 4.00, 'Monthly', 200.00, '2024-04-12', 6000.00, 'Completed'),

(12,12, 'Auto Loan', 15000.00, 5.00, 'weekly', 400.00, '2024-04-18', 15000.00, 'Active'),

(13,13, 'Home Loan', 200000.00, 4.50, 'Monthly', 1500.00, '2024-04-22', 200000.00, 'Active'),

(14,14, 'Education Loan', 30000.00, 6.50, 'Monthly', 600.00, '2024-04-08', 30000.00, 'Active'),

(15,15, 'Business Loan', 125000.00, 5.00, 'weekly', 1000.00, '2024-05-01', 125000.00, 'Active'),

(16,16, 'Personal Loan', 7000.00, 4.75, 'Monthly', 300.00, '2024-04-28', 7000.00, 'Active'),

(17,17, 'Auto Loan', 18000.00, 3.50, 'Monthly', 500.00, '2024-04-14', 18000.00, 'Active'),

(18,18, 'Home Loan', 220000.00, 4.25, 'weekly', 1600.00, '2024-04-30', 220000.00, 'Active'),

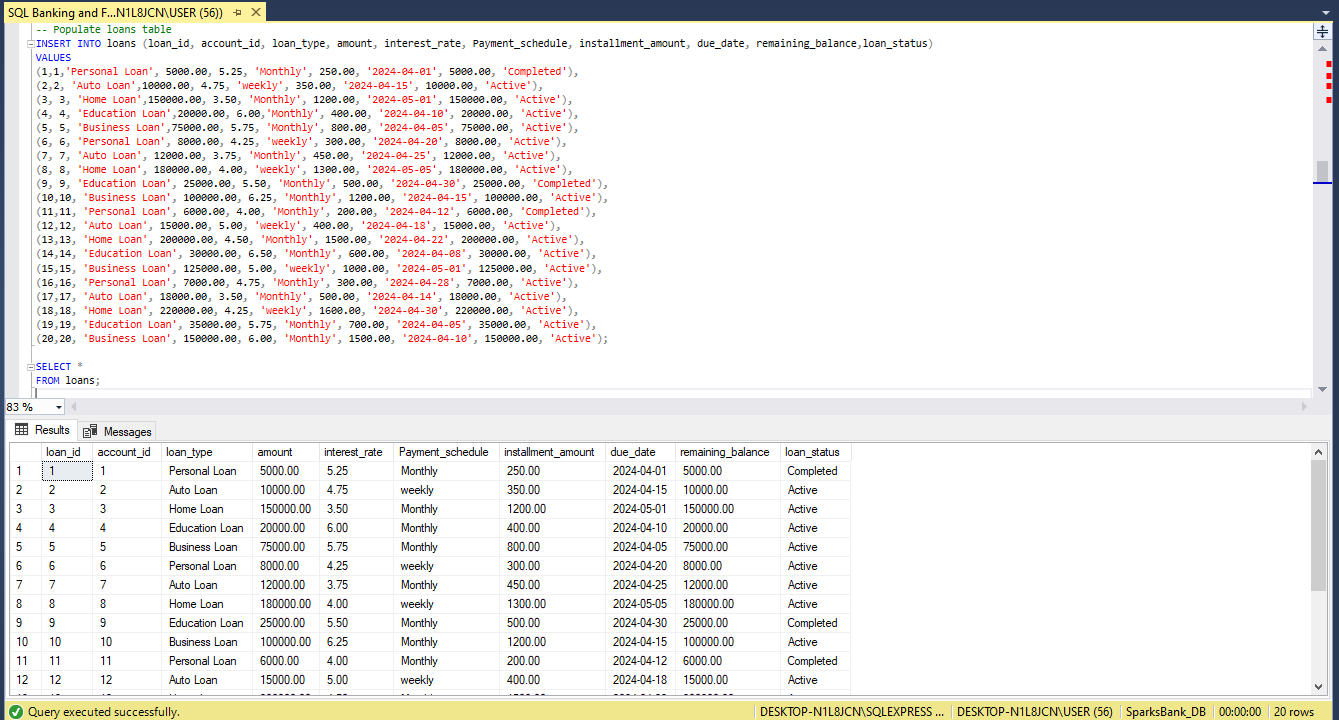
(19,19, 'Education Loan', 35000.00, 5.75, 'Monthly', 700.00, '2024-04-05', 35000.00, 'Active'),

(20,20, 'Business Loan', 150000.00, 6.00, 'Monthly', 1500.00, '2024-04-10', 150000.00, 'Active');

SELECT \*

FROM loans;

**RESULT**



**CODE**

--Populate Employees Table

INSERT INTO employees (employee\_id, employee\_name, years\_of\_service, department)

VALUES

('1','Alice Johnson','12','Management'),

('2','Bob Smith','8','Operations'),

('3',' Meredith Grey','6','Customer Care'),

('4','Chika Chukwu','3','Marketing'),

('5','Chidera Mbakwe','10','Credit Risk'),

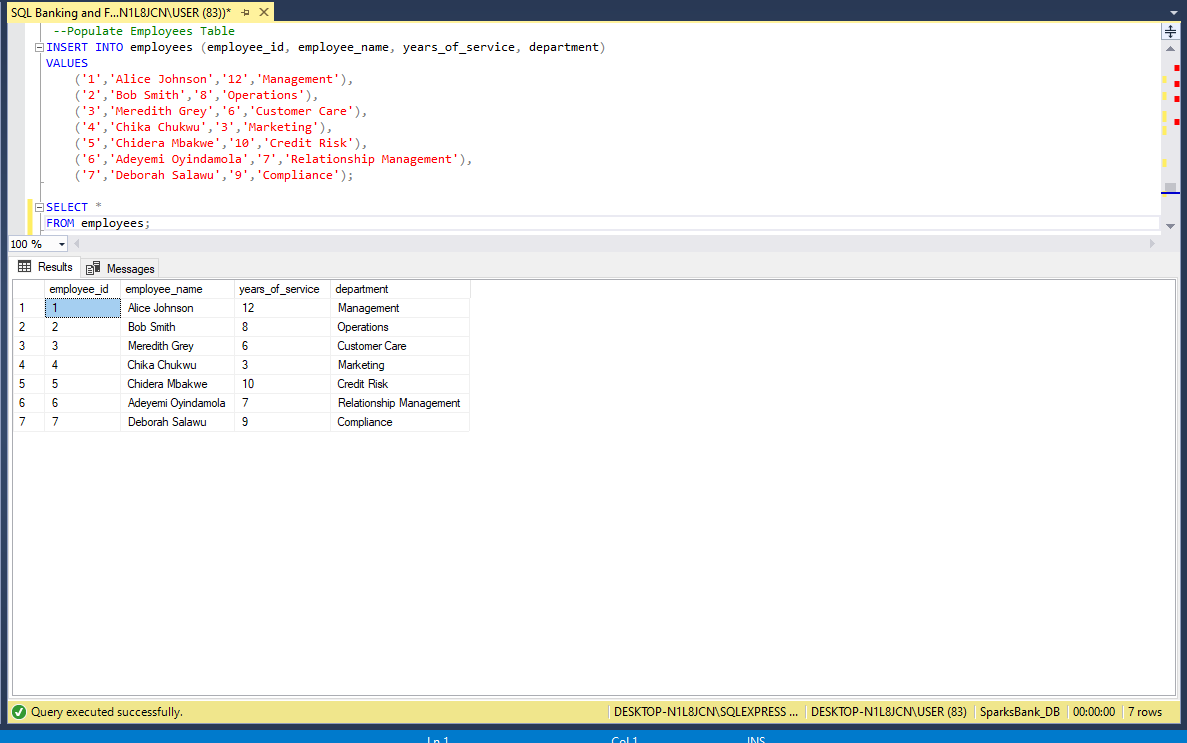
('6','Adeyemi Oyindamola','7','Relationship Management'),

('7','Deborah Salawu','9','Compliance');

SELECT \*

FROM employees;

**RESULT**

****

**FINANCIAL REPORT**

Generate financial reports such as balance sheets, income statements, and cash flow statements.

1. **BALANCE SHEET**

**CODE**

-- BALANCE SHEET

-- Total Assets(to check the total amount for all accounts)

SELECT SUM(balance) AS total\_assets

FROM accounts;

-- Total Liabilities(to check the total liabilities, which is the total amount of active loans)

SELECT SUM(amount) AS total\_liabilities

FROM loans

WHERE loan\_status = 'active';

-- Total Equity () (We have more debt than assets)

SELECT (SELECT SUM(balance) FROM accounts) - (SELECT SUM(amount) FROM loans) AS total\_equity;

-- Balance Sheet (Shows a summary of our balance sheet report)

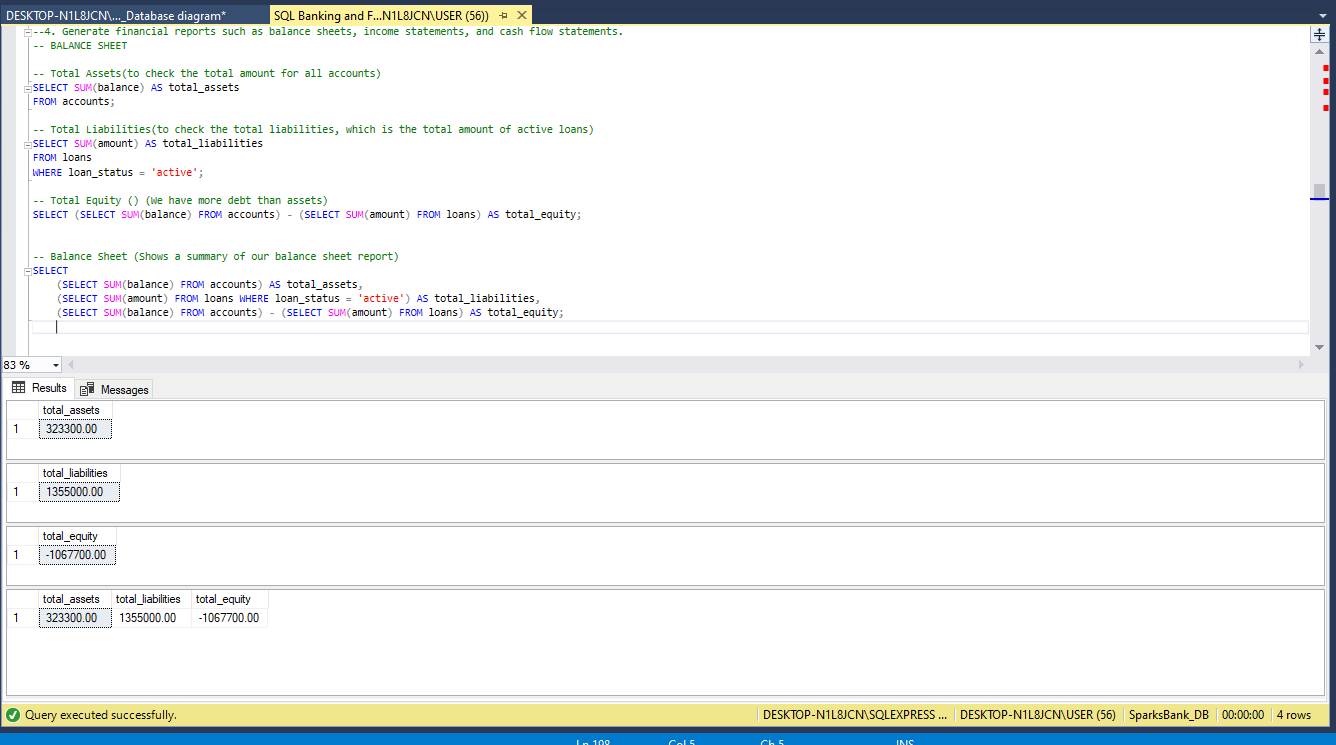
SELECT

(SELECT SUM(balance) FROM accounts) AS total\_assets,

(SELECT SUM(amount) FROM loans WHERE loan\_status = 'active') AS total\_liabilities,

(SELECT SUM(balance) FROM accounts) - (SELECT SUM(amount) FROM loans) AS total\_equity;

**RESULT**

****

**SUMMARY:** The provided code generates a summary of the balance sheet. It calculates total assets by summing up balances from accounts, total liabilities by summing up active loans, and total equity by subtracting total liabilities from total assets. To generate comprehensive financial documentation, additional queries are needed for income statements and cash flow statements. These statements would detail revenues, expenses, and cash movements over a specific period.

1. **INCOME STATEMENT**

**CODE**

--INCOME STATEMENT

-- Total Revenue (Total deposit by customers)

SELECT SUM(amount) AS total\_revenue

FROM transactions

WHERE transaction\_type = 'Deposit';

-- Total Withdrawal (Total withdrawals by customers)

SELECT SUM(amount) AS total\_withdrawal

FROM transactions

WHERE transaction\_type = 'Withdrawal';

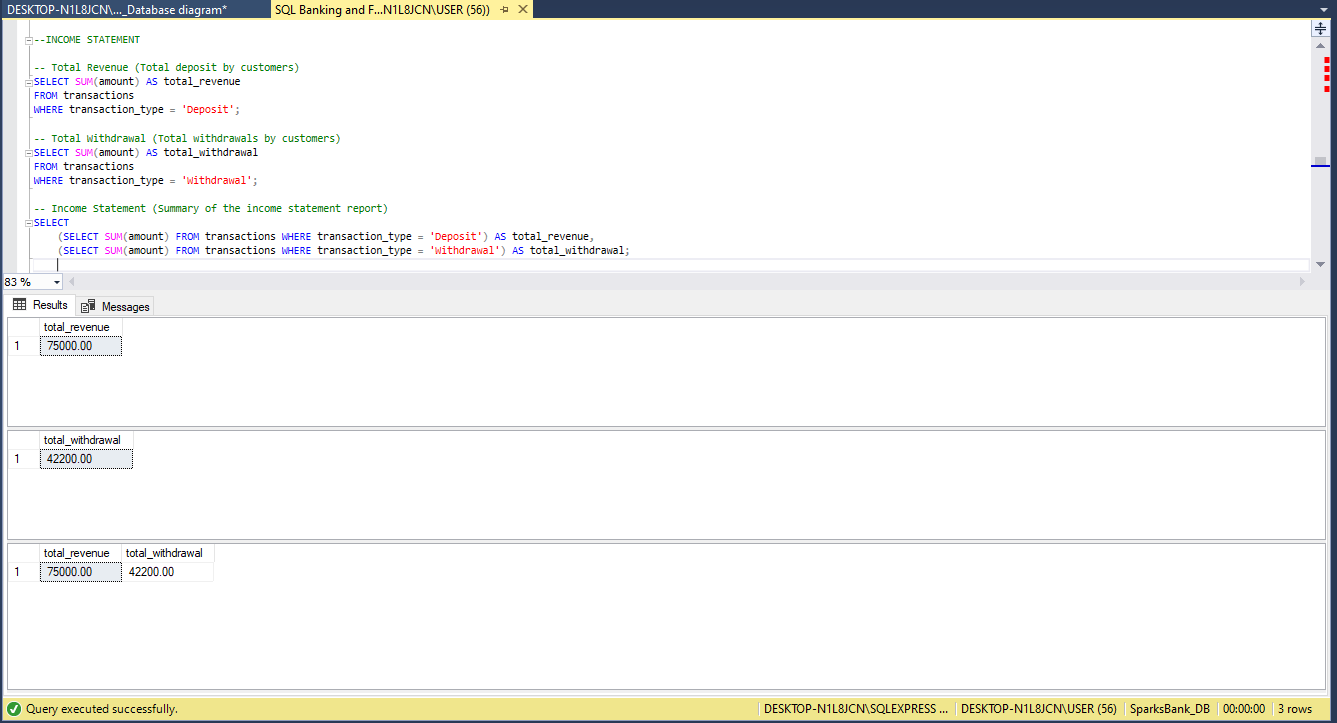
-- Income Statement (Summary of the income statement report)

SELECT

(SELECT SUM(amount) FROM transactions WHERE transaction\_type = 'Deposit') AS total\_revenue,

(SELECT SUM(amount) FROM transactions WHERE transaction\_type = 'Withdrawal') AS total\_withdrawal;

**RESULT**



**SUMMARY:** The provided code summarizes the income statement by calculating total revenue as the sum of deposit transactions and total withdrawals as the sum of withdrawal transactions. However, to produce a comprehensive income statement, additional calculations are required, including revenues, expenses, and net income over a specific period. These calculations would involve categorizing transactions and accounting for various income and expense items

1. **CASH FLOW STATEMENTS**

**CODE**

-- Cash Inflows (Deposit made into the bank)

SELECT SUM(amount) AS cash\_inflows

FROM transactions

WHERE transaction\_type = 'Deposit';

-- Cash Outflows (Withdrawal made out of the bank)

SELECT SUM(amount) AS cash\_outflows

FROM transactions

WHERE transaction\_type = 'Withdrawal';

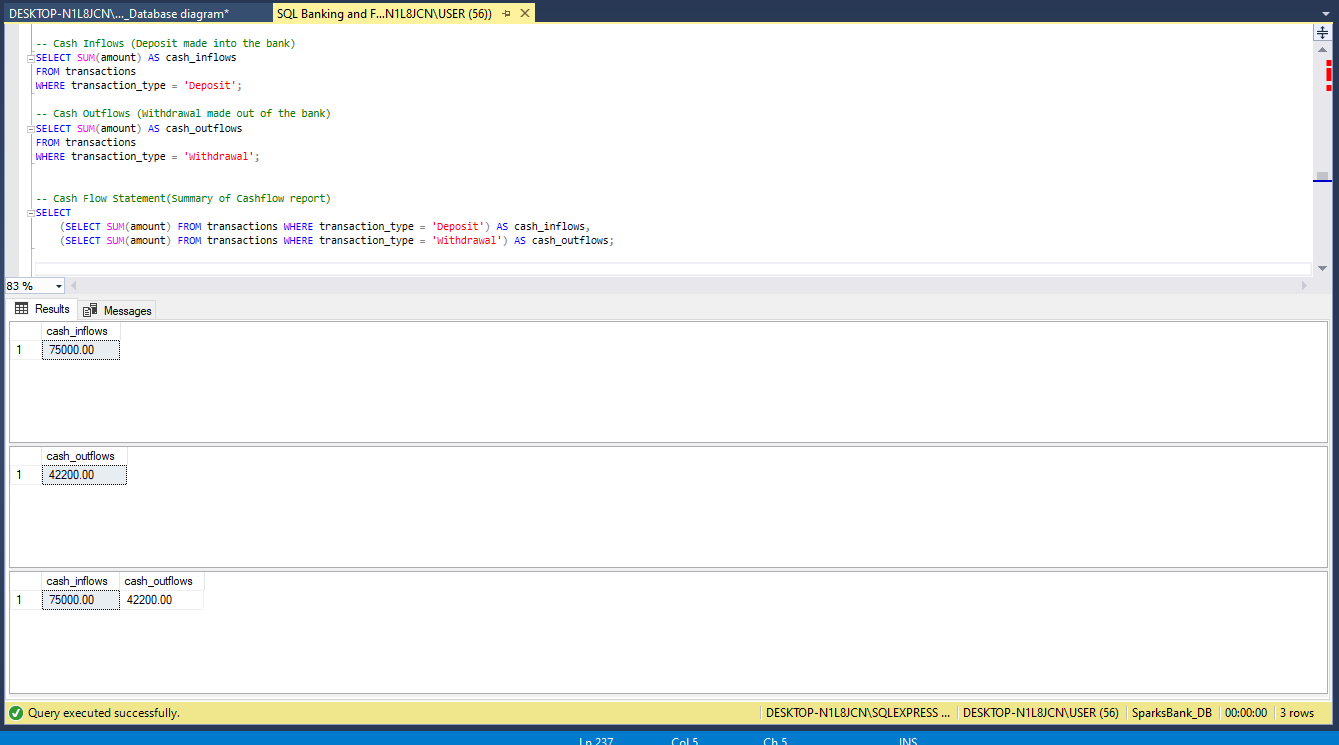
-- Cash Flow Statement(Summary of Cashflow report)

SELECT

(SELECT SUM(amount) FROM transactions WHERE transaction\_type = 'Deposit') AS cash\_inflows,

(SELECT SUM(amount) FROM transactions WHERE transaction\_type = 'Withdrawal') AS cash\_outflows;

**RESULT**

****

**SUMMARY:**  The provided code outlines a summary of the cash flow statement by computing cash inflows as the sum of deposit transactions and cash outflows as the sum of withdrawal transactions. However, to create a comprehensive cash flow statement, additional categorization and calculation of operating, investing, and financing activities are necessary. These activities would provide a clearer picture of the company's cash movements over a specified period.

**SELECTING DATA:**

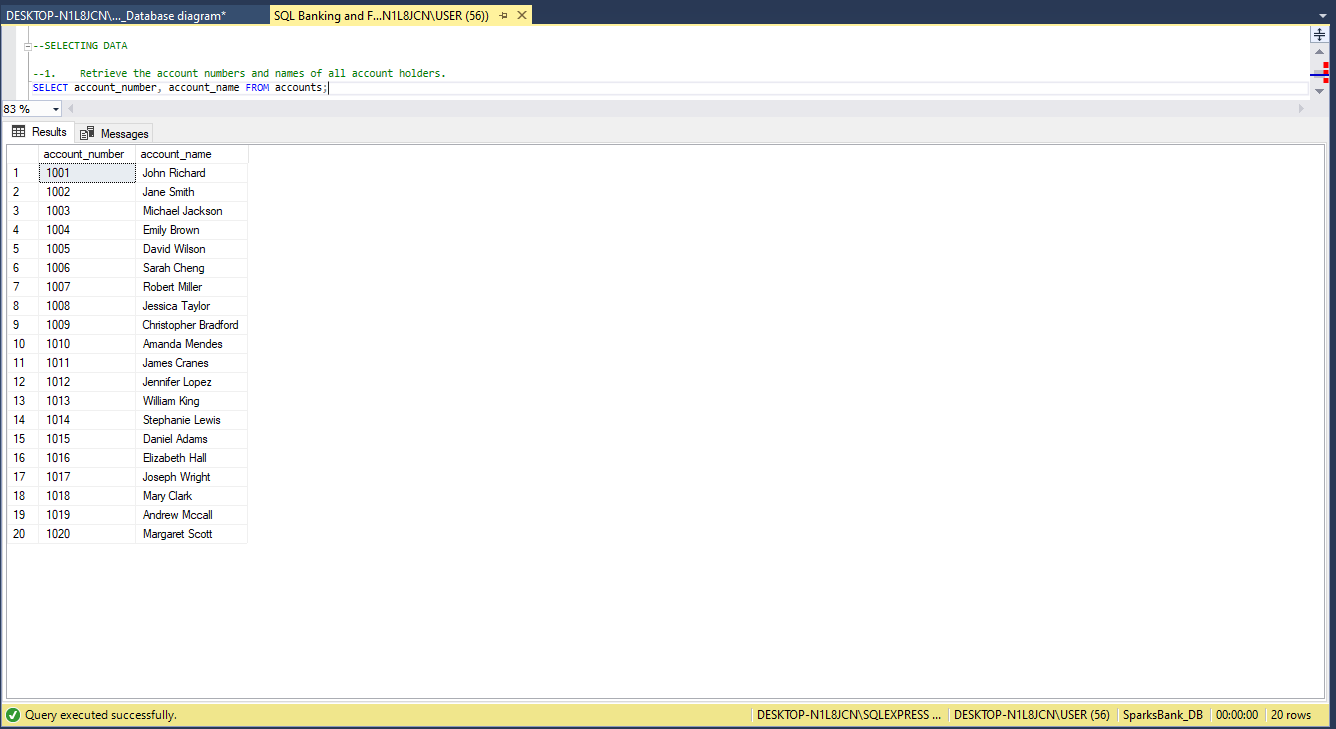
1. Retrieve the account numbers and names of all account holders.

**CODE**

--1. Retrieve the account numbers and names of all account holders.

SELECT account\_number, account\_name FROM accounts;

**RESULT**

****

**Summary:** This query fetches the account numbers and corresponding names of all account holders from the 'accounts' table.

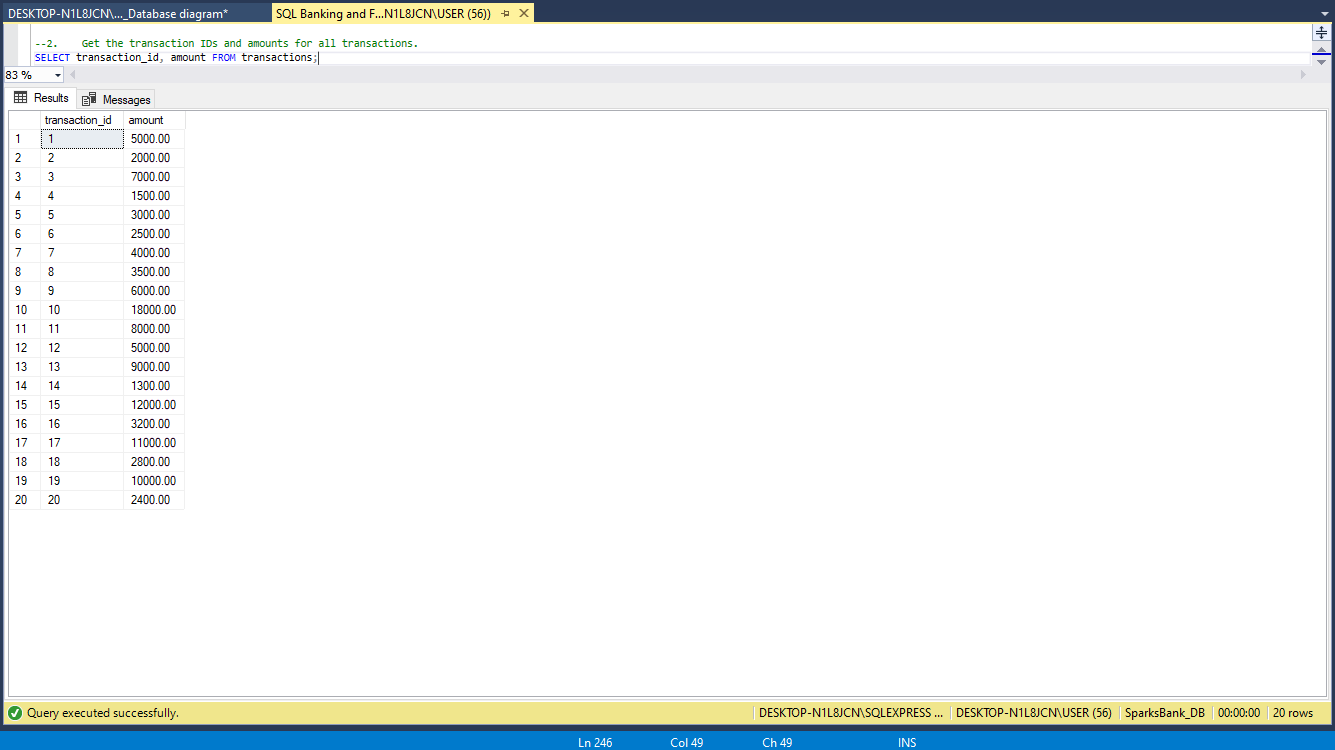
1. Get the transaction IDs and amounts for all transactions.

**CODE**

--2. Get the transaction IDs and amounts for all transactions.

SELECT transaction\_id, amount FROM transactions;

**RESULT**

****

**Summary:** This query retrieves the transaction IDs and their respective amounts from the 'transactions' table.

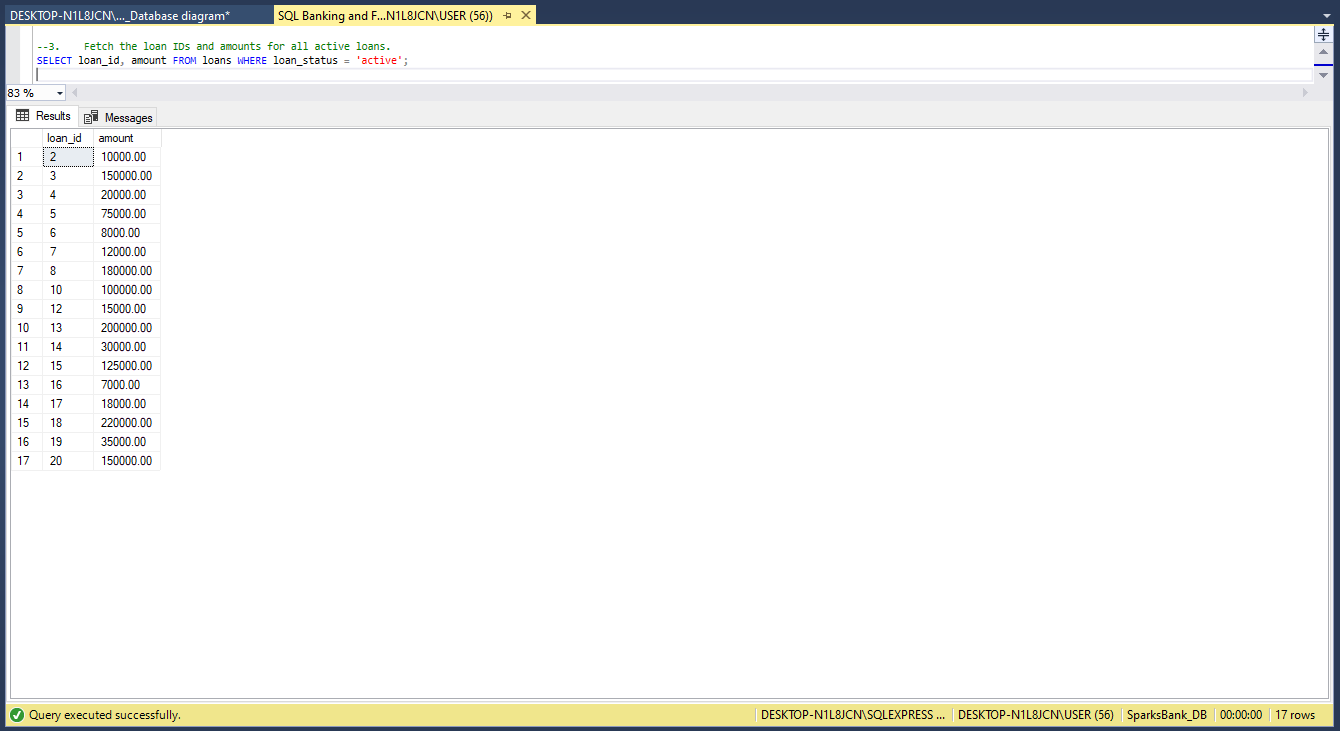
1. Fetch the loan IDs and amounts for all active loans.

**CODE**

--3. Fetch the loan IDs and amounts for all active loans.

SELECT loan\_id, amount FROM loans WHERE loan\_status = 'active';

**RESULT**

****

**Summary:** This query selects loan IDs and their corresponding amounts for loans that have an 'active' status from the 'loans' table.

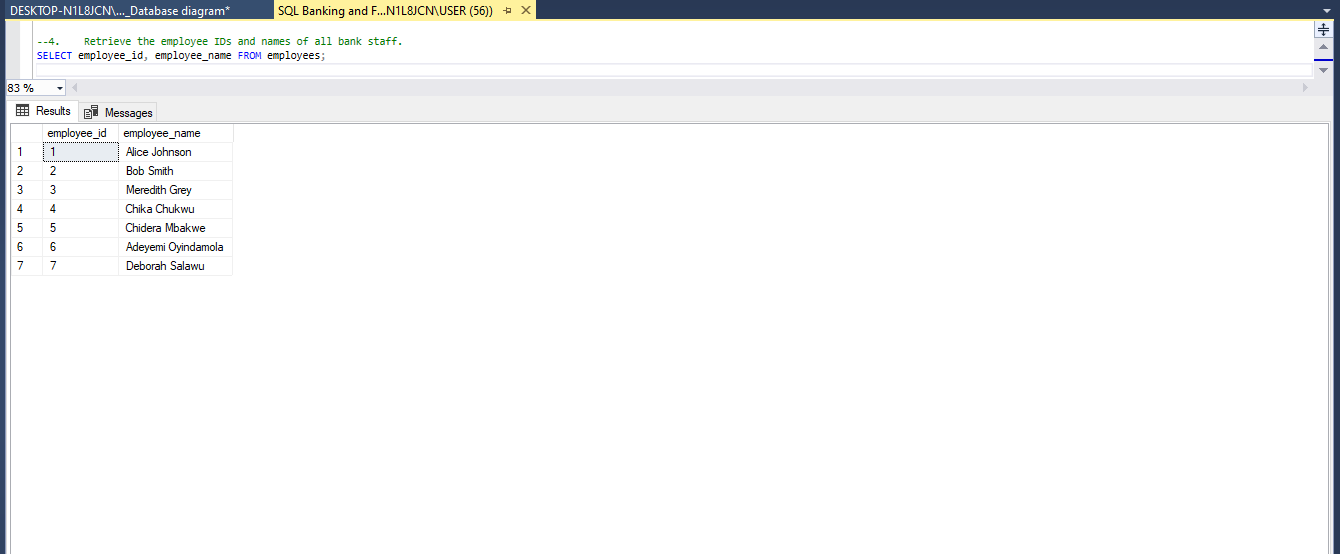
1. Retrieve the employee IDs and names of all bank staff.

**CODE**

--4. Retrieve the employee IDs and names of all bank staff.

SELECT employee\_id, employee\_name FROM employees;

**RESULT**

****

**Summary:** This query fetches the employee IDs and their associated names from the 'employees' table, identifying all bank staff.

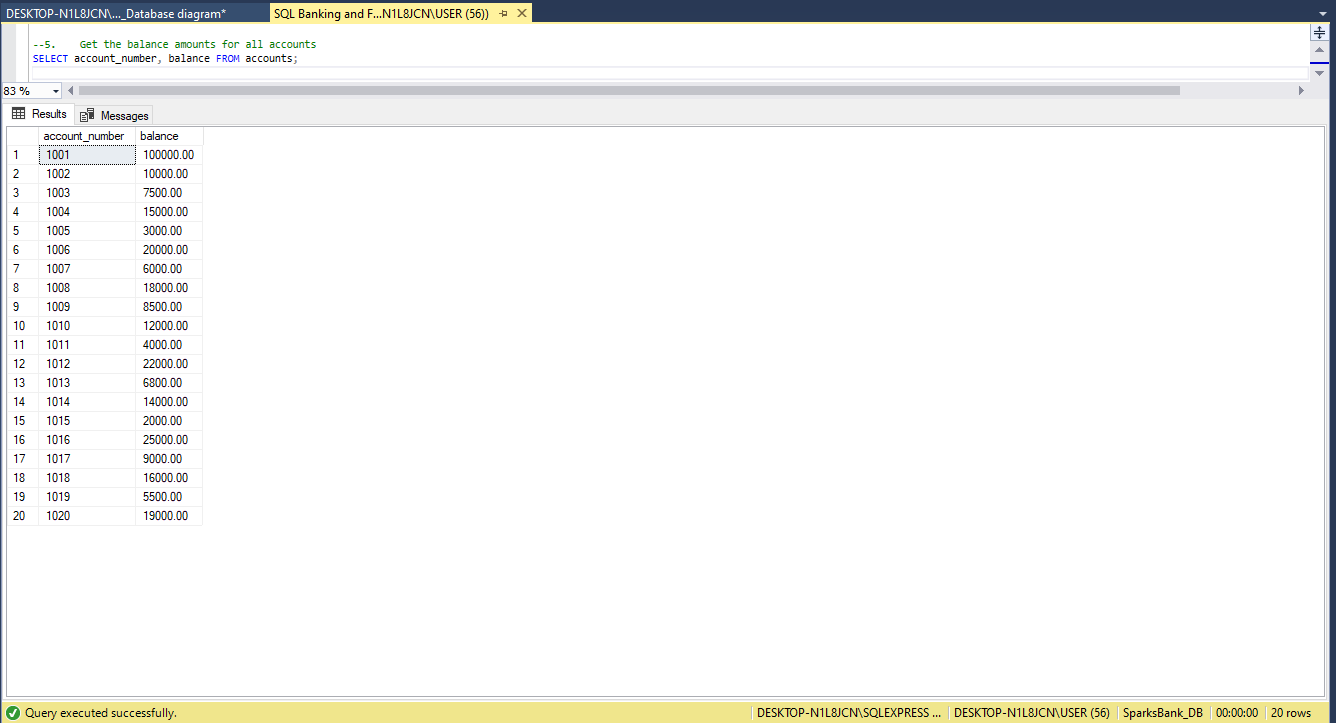
1. Get the balance amounts for all accounts.

**CODE**

--5. Get the balance amounts for all accounts

SELECT account\_number, balance FROM accounts;

**RESULT**

****

**Summary:** This query retrieves the account numbers and their respective balance amounts from the 'accounts' table, providing an overview of account balances.

**FILTERING**

1. Retrieve the account numbers and names of account holders with a balance greater than $10,000 and more than 5 transactions.

**CODE**

--Retrieve the account numbers and names of account holders with a balance greater than $10,000 and more than 5 transactions.

--According to this question our database does not have a customer with a balance greater than 10,000 for more than 5 transactions, but we have a customer with 5 transactions and a balance greater than $10000.

SELECT a.account\_number, a.account\_name

FROM accounts AS a

INNER JOIN (

SELECT account\_id, COUNT(\*) AS num\_transactions

FROM transactions

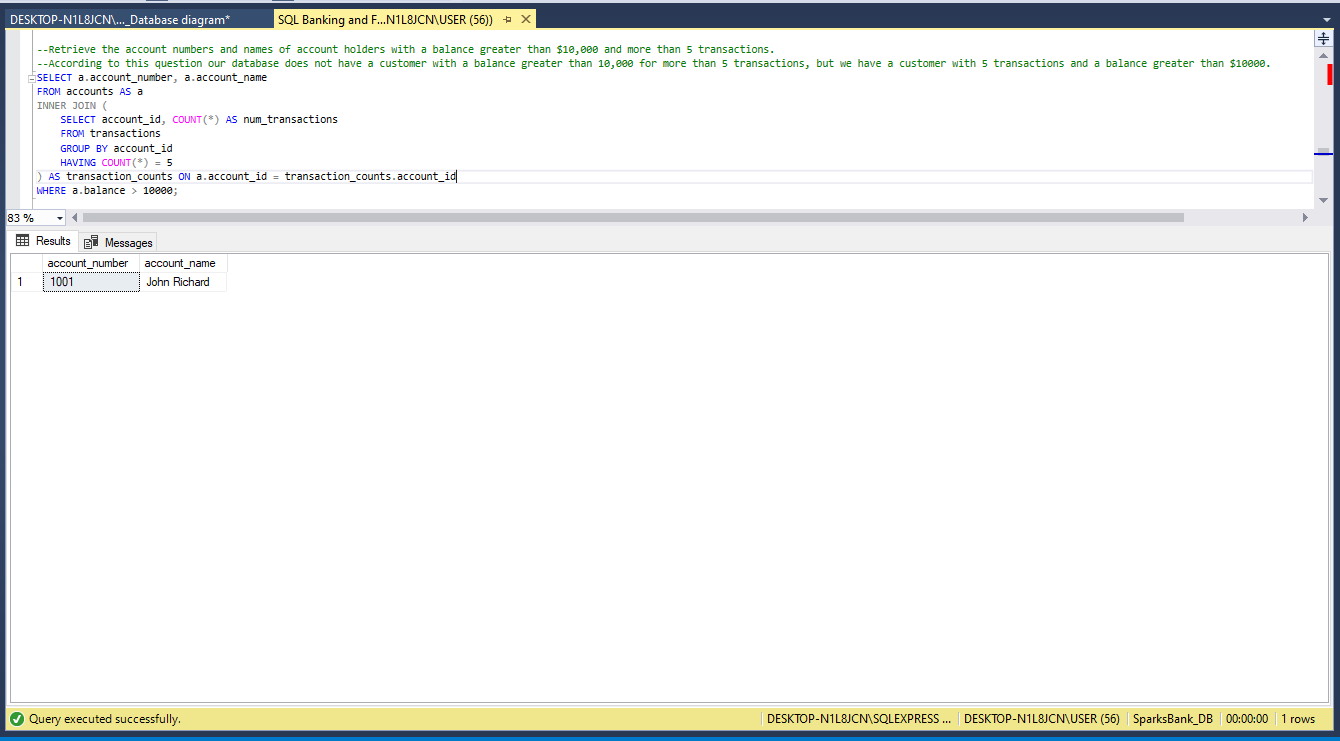
GROUP BY account\_id

HAVING COUNT(\*) = 5

) AS transaction\_counts ON a.account\_id = transaction\_counts.account\_id

WHERE a.balance > 10000;

**RESULT**

****

1. Retrieve the employee IDs and names of bank staff with more than 10 years of service and working in the 'Management' department.

**CODE**

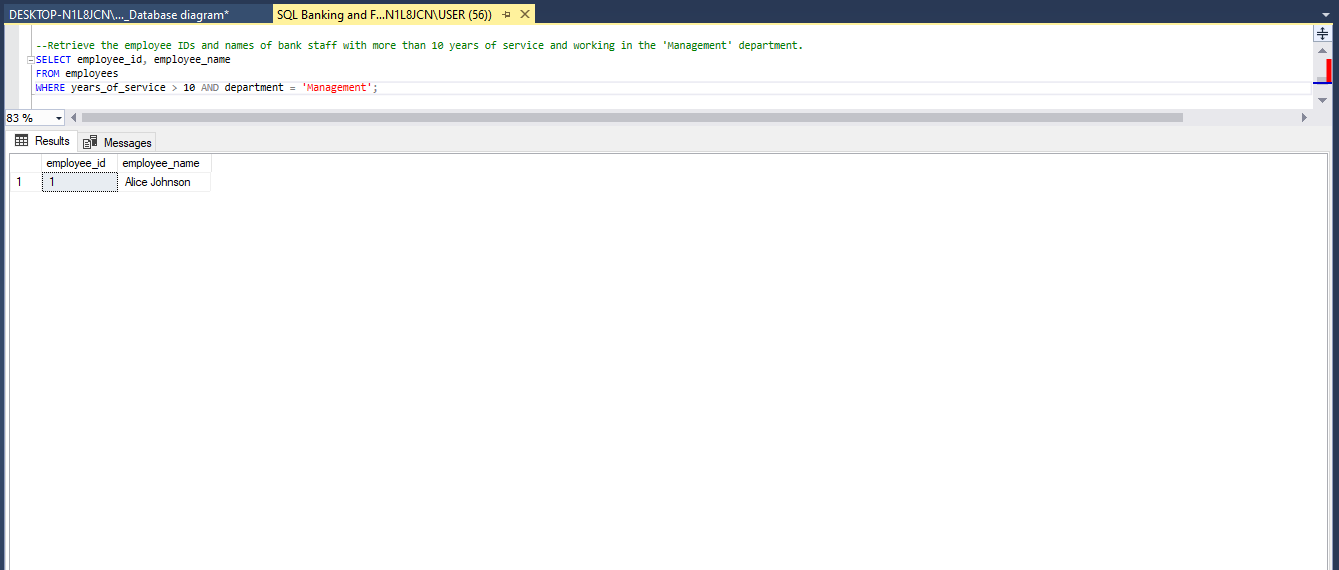
--Retrieve the employee IDs and names of bank staff with more than 10 years of service and working in the 'Management' department.

SELECT employee\_id, employee\_name

FROM employees

WHERE years\_of\_service > 10 AND department = 'Management';

**RESULT**

****

1. Get the balance amounts for accounts opened before January 1, 2010, and with a balance above $50,000.

**CODE**

--Get the balance amounts for accounts opened before January 1, 2010, and with a balance above $50,000.

SELECT account\_number, balance

FROM accounts

WHERE opening\_date < '2010-01-01' AND balance > 50000;

**RESULT**

**SORTING**

1. Retrieve the account numbers and names of all account holders, sorted alphabetically by account names.

**CODE**

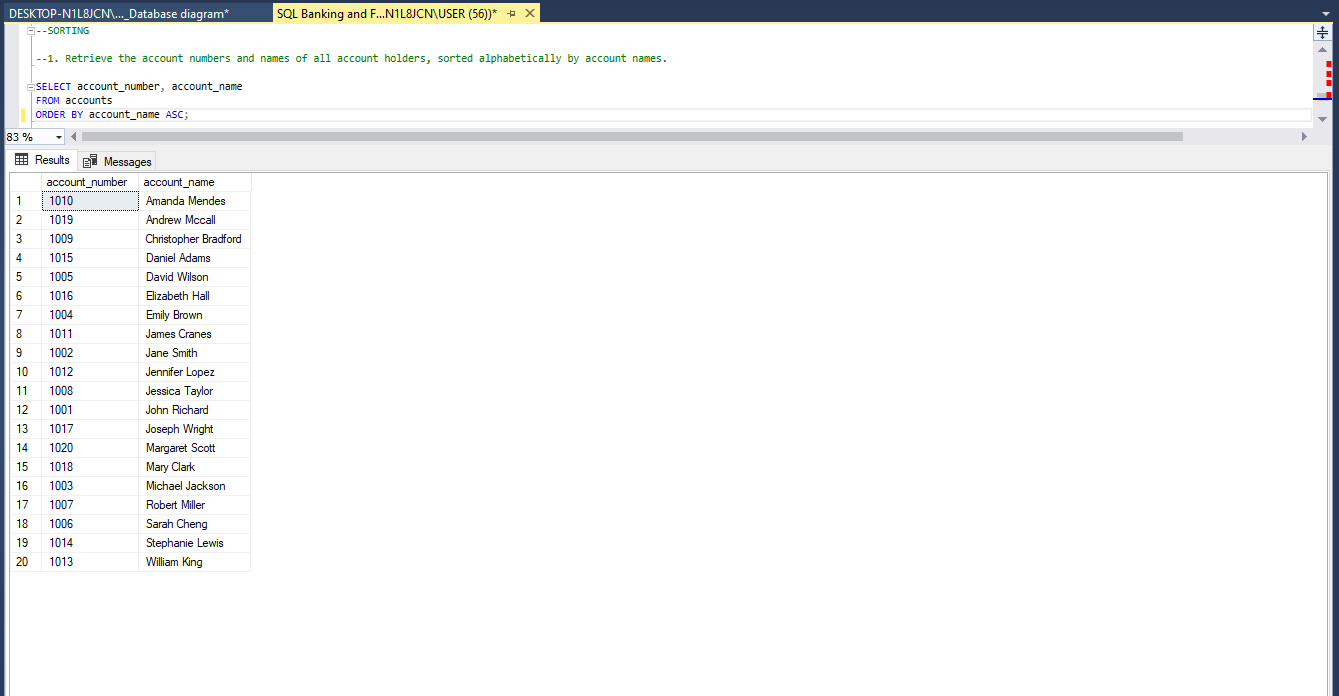
--1. Retrieve the account numbers and names of all account holders, sorted alphabetically by account names.

SELECT account\_number, account\_name

FROM accounts

ORDER BY account\_name ASC;

**RESULT**

****

1. Get the transaction IDs and amounts for all transactions, sorted in descending order of transaction amounts.

**CODE**

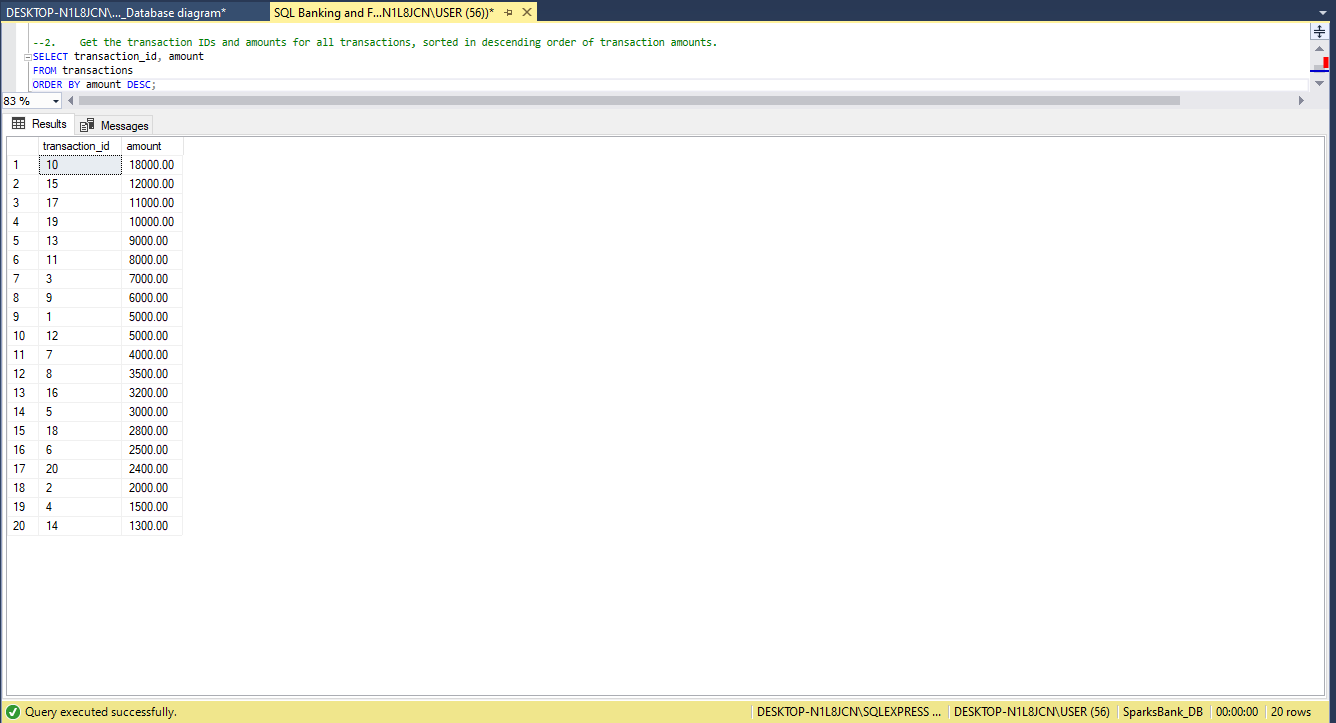
--2. Get the transaction IDs and amounts for all transactions, sorted in descending order of transaction amounts.

SELECT transaction\_id, amount

FROM transactions

ORDER BY amount DESC;

**RESULT**

****

1. Fetch the loan IDs and amounts for all active loans, sorted in ascending order of loan amounts.

**CODE**

--3. Fetch the loan IDs and amounts for all active loans, sorted in ascending order of loan amounts.

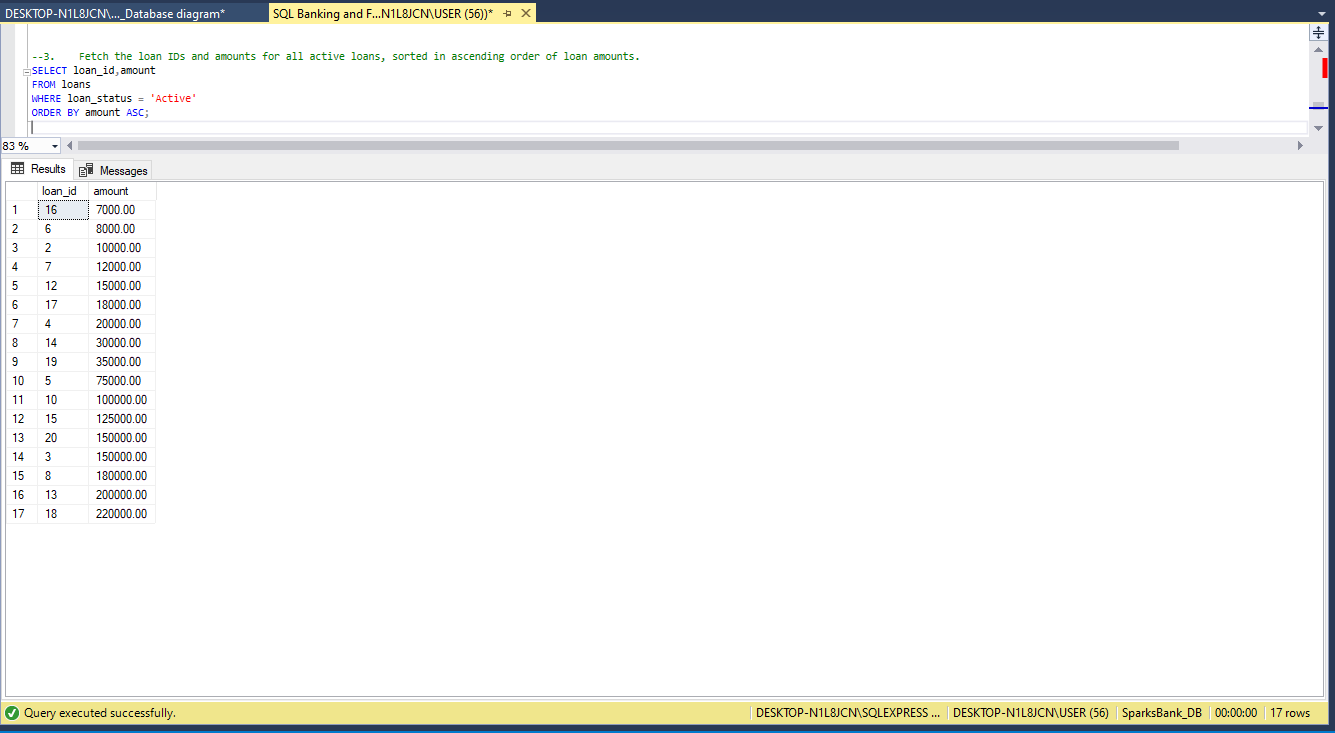
SELECT loan\_id,amount

FROM loans

WHERE loan\_status = 'Active'

ORDER BY amount ASC;

**RESULT**

****

1. Retrieve the employee IDs and names of all bank staff, sorted alphabetically by employee names.

**CODE**

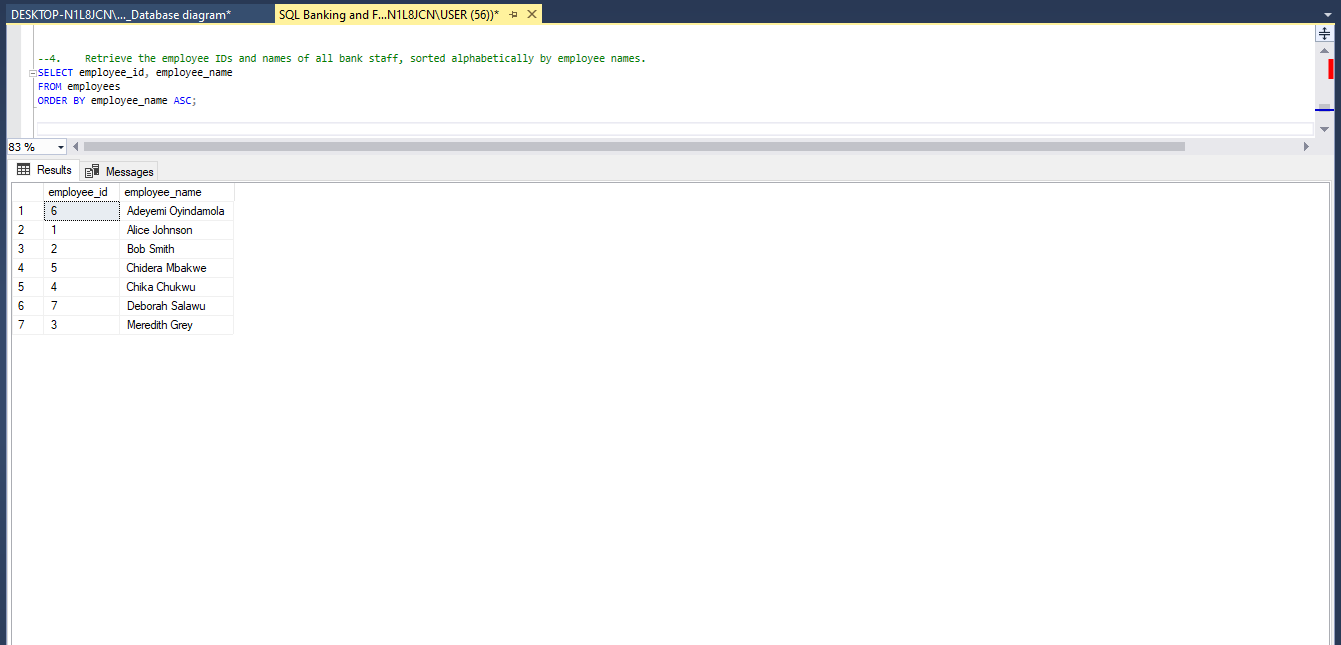
--4. Retrieve the employee IDs and names of all bank staff, sorted alphabetically by employee names.

SELECT employee\_id, employee\_name

FROM employees

ORDER BY employee\_name ASC;

**RESULT**

****

1. Get the balance amounts for all accounts, sorted in descending order of balance amounts.

**CODE**

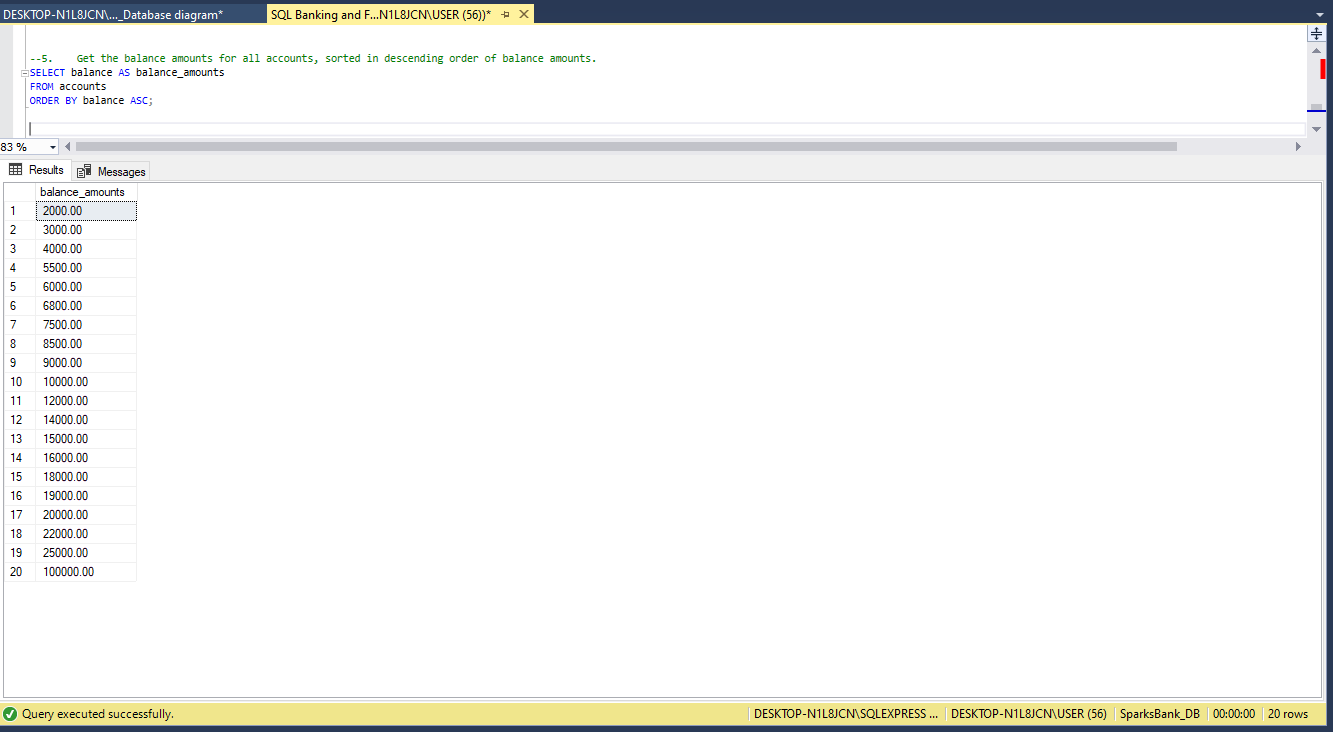
--5. Get the balance amounts for all accounts, sorted in descending order of balance amounts.

SELECT balance AS balance\_amounts

FROM accounts

ORDER BY balance ASC;

**RESULT**

****

**DISTINCT**

1. Retrieve distinct account types available in the bank.

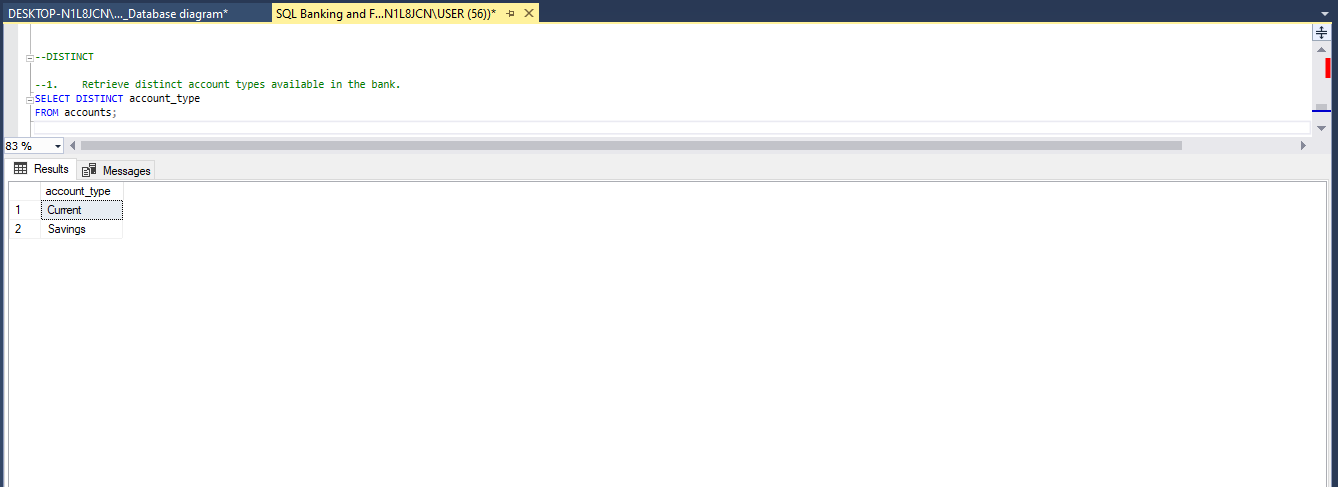
**CODE**

--1. Retrieve distinct account types available in the bank.

SELECT DISTINCT account\_type

FROM accounts;

**RESULT**



2. Get distinct transaction types for all transactions.

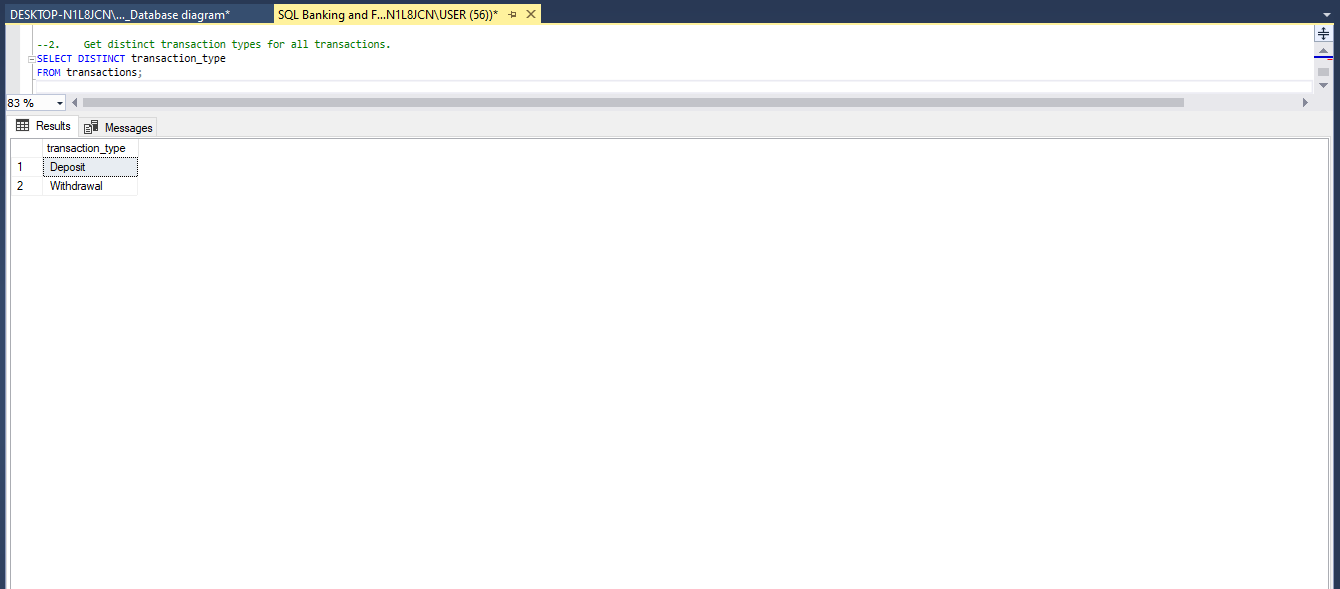
**CODE**

--2. Get distinct transaction types for all transactions.

SELECT DISTINCT transaction\_type

FROM transactions;

**RESULT**



1. Fetch distinct loan types for all active loans.

**CODE**

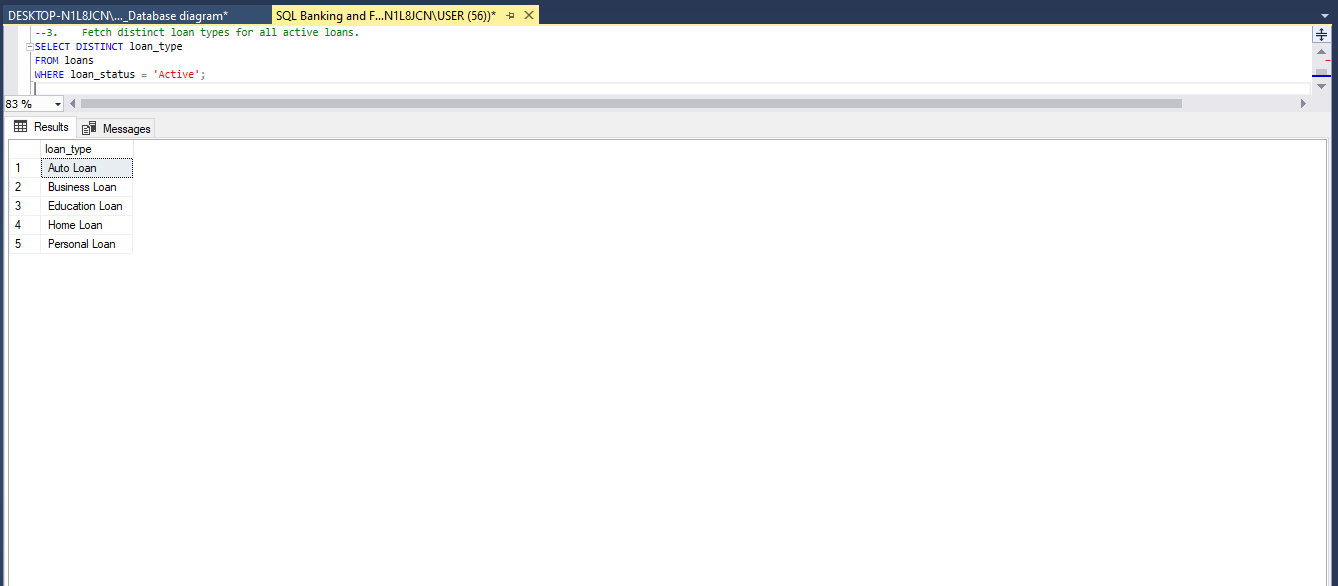
--3. Fetch distinct loan types for all active loans.

SELECT DISTINCT loan\_type

FROM loans

WHERE loan\_status = 'Active'

**RESULT**



1. Retrieve distinct department names for all bank staff.

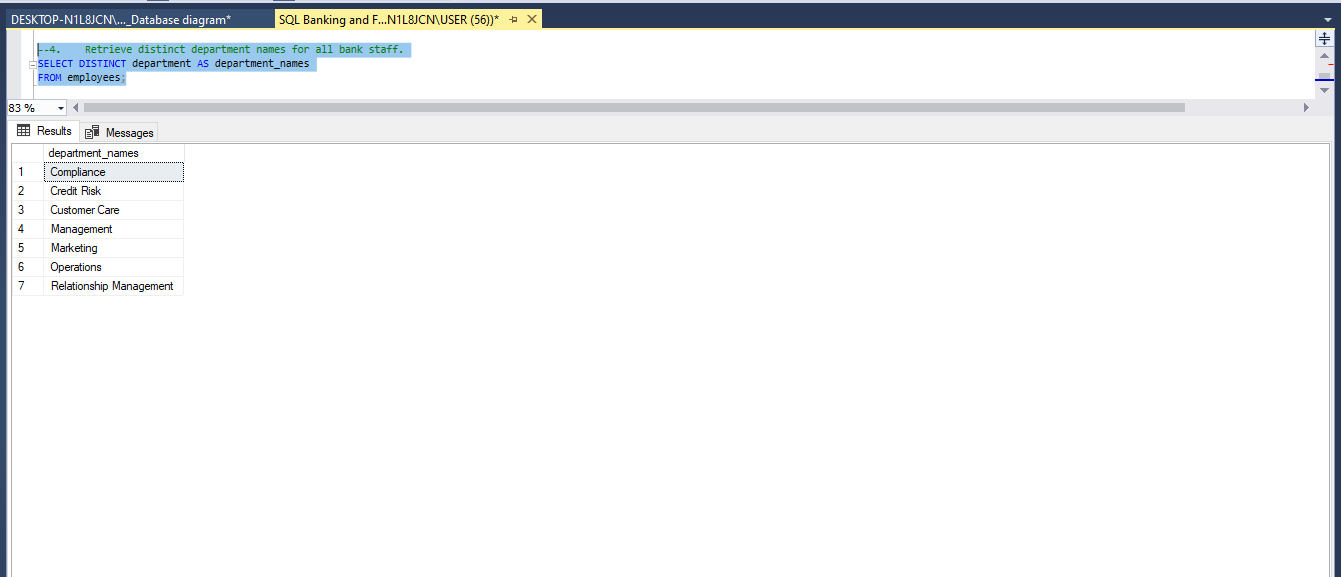
**CODE**

--4. Retrieve distinct department names for all bank staff.

SELECT DISTINCT department AS department\_names

FROM employees;

**RESULT**



1. Get distinct employee roles for all bank staff.

**CODE**

--5. Get distinct employee roles for all bank staff.

--Add new column employee role

ALTER TABLE employees

ADD employee\_role Varchar(50);

-- Populate employee\_role column

UPDATE employees

SET employee\_role =

CASE

WHEN employee\_id = 1 THEN 'Bank Manager'

WHEN employee\_id = 2 THEN 'Operayion Manager'

WHEN employee\_id = 3 THEN 'Customercare representative'

WHEN employee\_id = 4 THEN 'Retail Marketer'

WHEN employee\_id = 5 THEN 'Credit Review Officer'

WHEN employee\_id = 6 THEN 'Customer Relationship Manager'

WHEN employee\_id = 7 THEN 'Compliance officer'

ELSE employee\_role

END

WHERE employee\_id IN (1, 2, 3, 4, 5, 6, 7);

SELECT \*

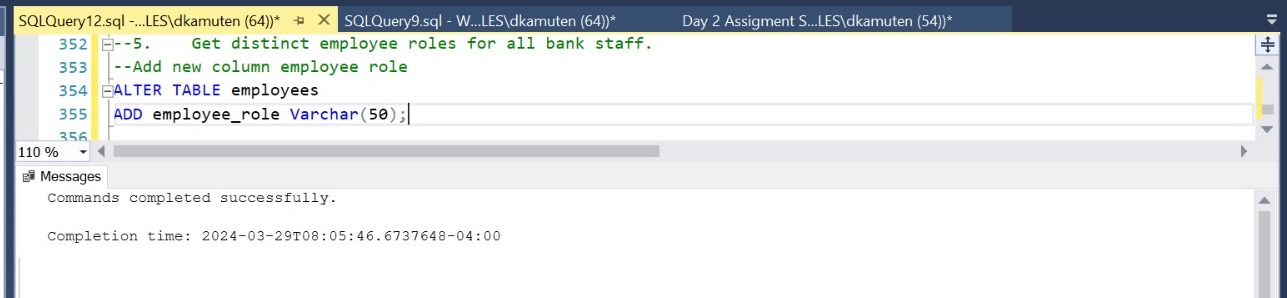
FROM employees;

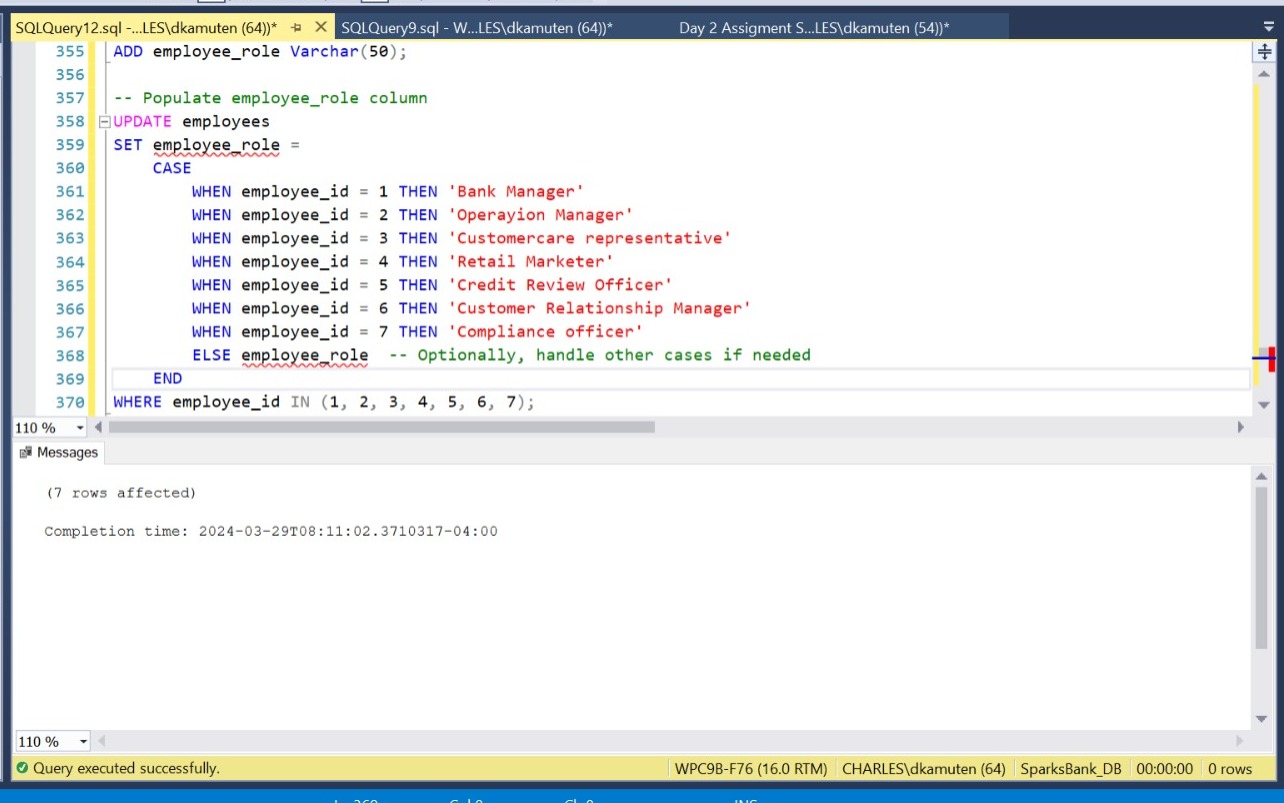
--Distint Employee role

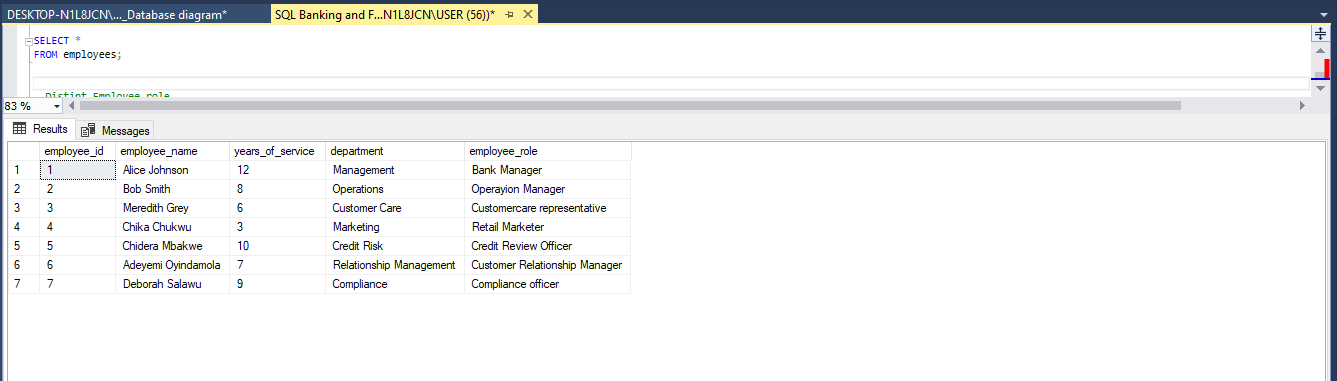
SELECT DISTINCT employee\_role

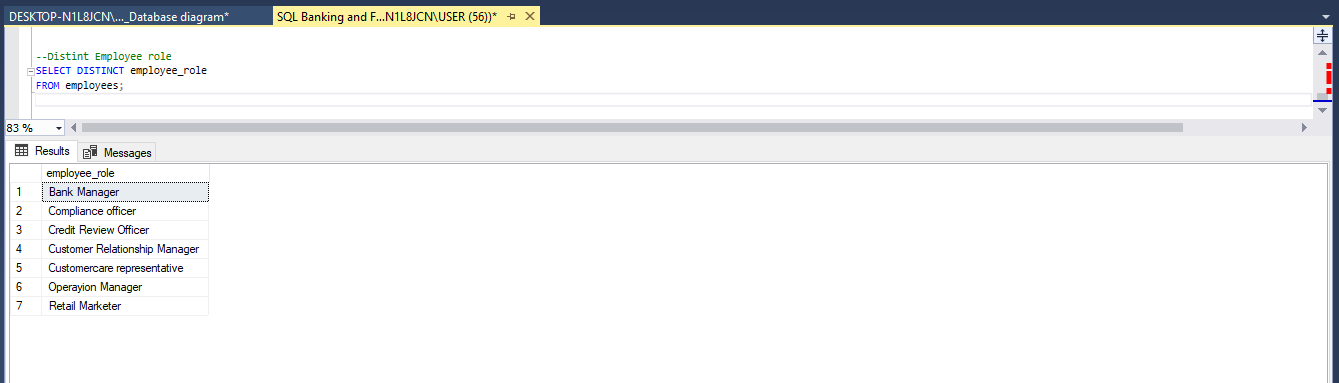
FROM employees;

**RESULT**









**CONCLUSION**

In the realm of banking and finance, effective data management is paramount for operational efficiency, regulatory compliance, and customer satisfaction. Through meticulous database design, our project has endeavored to establish a robust framework tailored to the needs of Sparks Bank. By organizing and centralizing crucial financial data, we aim to enhance administrative processes, ensure data integrity, and facilitate informed decision-making.

Our database design project has successfully addressed key aspects of Sparks Bank's operations, including managing customer accounts, processing transactions securely, administering loans, and generating accurate financial reports. Leveraging Microsoft SQL Server Management Studio, we have developed a structured database schema encompassing entities such as accounts, transactions, loans, and employees.

Employing the principles of normalization, our database design adheres to the third normal form (3NF), mitigating data redundancies and ensuring optimal storage efficiency. Furthermore, the implementation of constraints, including primary key constraints and data type constraints, safeguards data integrity and enhances security within the database system.

The inclusion of a comprehensive entity-relationship model (ERM) has provided a blueprint for defining the relationships and attributes of key entities, facilitating a deeper understanding of the database structure and functionality.

Through extensive testing and population of database tables with meaningful sample data, we have evaluated the functionality and performance of the database, ensuring its readiness to handle real-world scenarios in the banking and finance sector.

**RECOMMENDATION**

1. **Continuous Testing and Optimization:** It is imperative to conduct ongoing testing and optimization procedures to ensure the scalability, reliability, and performance of the database system. Regular performance tuning and monitoring can help identify and address potential inefficiencies.
2. **Enhanced Security Measures:** Given the sensitive nature of financial data, consider implementing additional security measures such as encryption techniques, access control policies, and audit trails to safeguard against unauthorized access or data breaches.
3. **Integration with Analytics Tools:** Explore opportunities to integrate the database with analytics tools and business intelligence platforms to derive actionable insights from the wealth of financial data available. This integration can support strategic decision-making and facilitate predictive analytics for risk management and customer engagement.
4. **User Training and Documentation:** Provide comprehensive training resources and documentation for bank staff responsible for interacting with the database system. Ensuring that users are proficient in utilizing database functionalities and adhering to best practices can mitigate errors and optimize productivity.
5. **Adaptability to Regulatory Changes:** Stay abreast of evolving regulatory requirements in the banking and finance industry and ensure that the database design remains adaptable to accommodate changes seamlessly. Regular audits and compliance assessments can aid in identifying and addressing compliance gaps effectively.
6. **Disaster Recovery and Business Continuity Planning:** Develop robust disaster recovery and business continuity plans to mitigate the impact of unforeseen events such as system failures, natural disasters, or cyber-attacks. Implementing backup and recovery mechanisms can minimize downtime and ensure the uninterrupted availability of critical financial services.

Ultimately, by embracing these recommendations and fostering a culture of innovation and continuous improvement, Sparks Bank can leverage its database system as a strategic asset to drive operational excellence, foster customer trust, and maintain a competitive edge in the dynamic landscape of the banking and finance industry.