Lab 5

Banking System - Project Report

Course: DBI202 | Instructor: Ms. Nguyễn Thị Thu Thảo

Group Members:

Student ID	Full Name	Group	Group Mark	Contribution (%)	Mark	Note
SE203334	Hà Nguyễn Tiến Đạt	7		100		
SE190596	Trần Hữu Việt	7		100		
SE193659	Mai Thành Được	7		100		
SS190849	Lê Nguyên Ngọc	7		100		

1. Objective

The lab's goal is to practice SQL programming on the "Banking System" database from Lab 4. This involves writing a variety of SQL queries (from basic to advanced), implementing functions, stored procedures, and triggers to automate tasks and enforce business rules. Students will also create views to simplify data access and indexes to improve performance.

2. SQL Queries

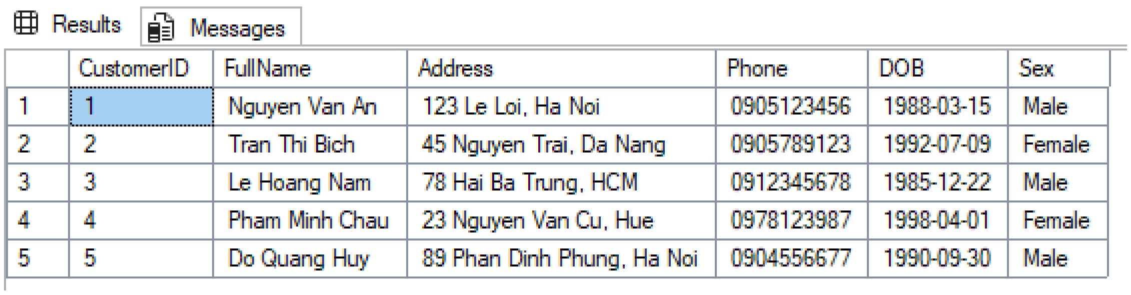
This section presents the SQL commands executed on the "Banking System" database, ranging from basic to advanced.

2.1. Basic Queries

2.1.1. Select all records

- **Description:** Retrieve all customers in the system.

```
SELECT * FROM CUSTOMER;
```

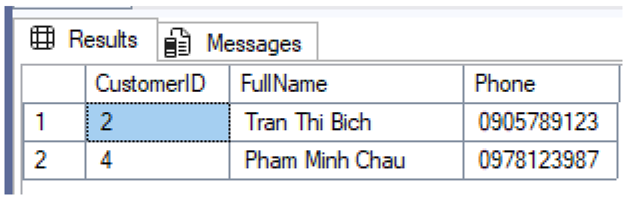


Results | Messages

	CustomerID	FullName	Address	Phone	DOB	Sex
1	1	Nguyen Van An	123 Le Loi, Ha Noi	0905123456	1988-03-15	Male
2	2	Tran Thi Bich	45 Nguyen Trai, Da Nang	0905789123	1992-07-09	Female
3	3	Le Hoang Nam	78 Hai Ba Trung, HCM	0912345678	1985-12-22	Male
4	4	Pham Minh Chau	23 Nguyen Van Cu, Hue	0978123987	1998-04-01	Female
5	5	Do Quang Huy	89 Phan Dinh Phung, Ha Noi	0904556677	1990-09-30	Male

2.1.2. Filter rows (WHERE)

- **Description:** Find all 'Female' customers. (CUSTOMER)

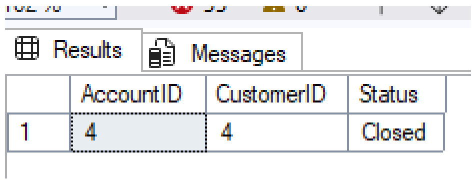


Results | Messages

	CustomerID	FullName	Phone
1	2	Tran Thi Bich	0905789123
2	4	Pham Minh Chau	0978123987

```
SELECT CustomerID, FullName, Phone FROM CUSTOMER
WHERE Sex = 'Female';
```

- **Description:** Find all accounts with a 'Closed' status. (ACCOUNT)

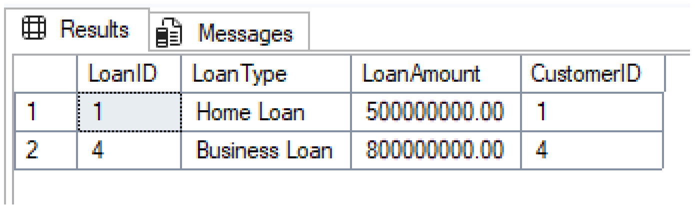


Results | Messages

	AccountID	CustomerID	Status
1	4	4	Closed

```
SELECT AccountID, CustomerID, Status FROM ACCOUNT
WHERE Status = 'Closed';
```

- **Description:** Find all loans with an amount greater than 400,000,000. (LOAN)



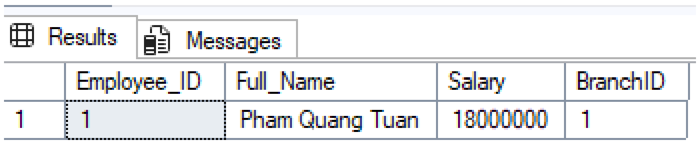
Results | Messages

	LoanID	LoanType	LoanAmount	CustomerID
1	1	Home Loan	500000000.00	1
2	4	Business Loan	800000000.00	4

```
SELECT LoanID, LoanType, LoanAmount, CustomerID FROM LOAN
WHERE LoanAmount > 400000000;
```

- **Description:** Find all employees in Branch 1 with a salary over 17,000,000. (EMPLOYEE)

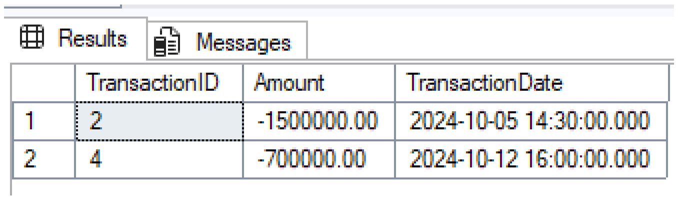
```
SELECT Employee_ID, Full_Name, Salary, BranchID FROM EMPLOYEE
WHERE BranchID = 1 AND Salary > 17000000;
```



	Employee_ID	Full_Name	Salary	BranchID
1	1	Pham Quang Tuan	18000000	1

- **Description:** Find all withdrawal transactions (Amount < 0). (TRANSACTIONS)

```
SELECT TransactionID, Amount, TransactionDate FROM TRANSACTIONS
WHERE Amount < 0;
```

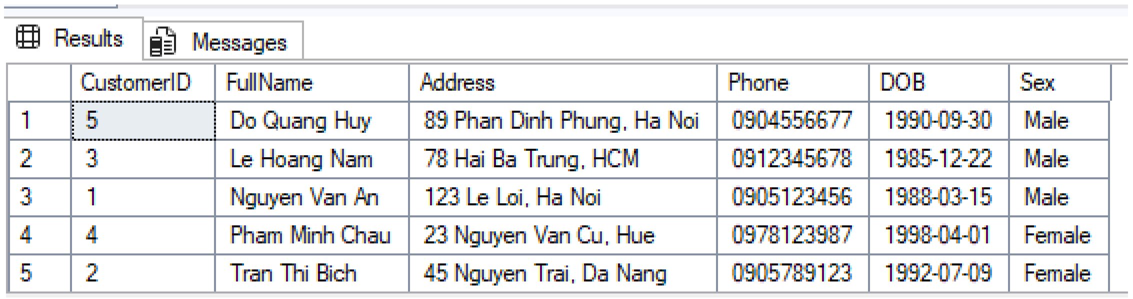


	TransactionID	Amount	TransactionDate
1	2	-1500000.00	2024-10-05 14:30:00.000
2	4	-700000.00	2024-10-12 16:00:00.000

2.1.3. Sort results (ORDER BY)

- **Description:** List all customers sorted alphabetically by FullName. (CUSTOMER)

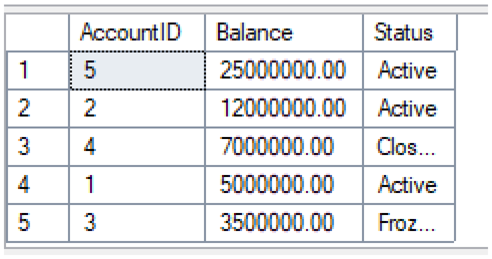
```
SELECT * FROM CUSTOMER ORDER BY FullName ASC;
```



	CustomerID	FullName	Address	Phone	DOB	Sex
1	5	Do Quang Huy	89 Phan Dinh Phung, Ha Noi	0904556677	1990-09-30	Male
2	3	Le Hoang Nam	78 Hai Ba Trung, HCM	0912345678	1985-12-22	Male
3	1	Nguyen Van An	123 Le Loi, Ha Noi	0905123456	1988-03-15	Male
4	4	Pham Minh Chau	23 Nguyen Van Cu, Hue	0978123987	1998-04-01	Female
5	2	Tran Thi Bich	45 Nguyen Trai, Da Nang	0905789123	1992-07-09	Female

- **Description:** List all accounts, sorted from highest Balance to lowest. (ACCOUNT)

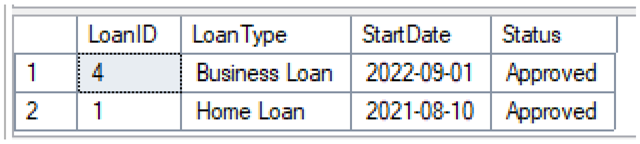
```
SELECT AccountID, Balance, Status FROM ACCOUNT
ORDER BY Balance DESC;
```



	AccountID	Balance	Status
1	5	25000000.00	Active
2	2	12000000.00	Active
3	4	7000000.00	Clos...
4	1	5000000.00	Active
5	3	3500000.00	Froz...

- **Description:** List all 'Approved' loans, sorted by StartDate (newest first). (LOAN)

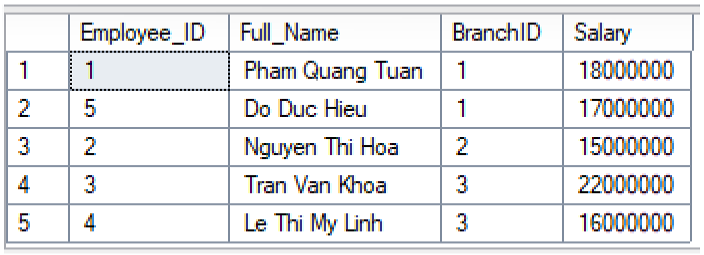
```
SELECT LoanID, LoanType, StartDate, Status FROM LOAN
WHERE Status = 'Approved'
ORDER BY StartDate DESC;
```



	LoanID	LoanType	StartDate	Status
1	4	Business Loan	2022-09-01	Approved
2	1	Home Loan	2021-08-10	Approved

- **Description:** List employees, sorted by BranchID and then by Salary (highest first). (EMPLOYEE)

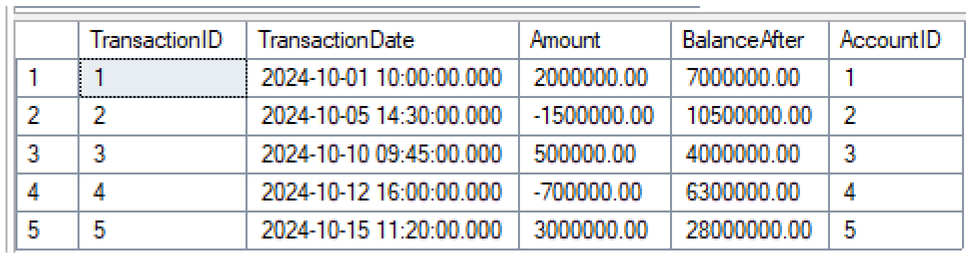
```
SELECT Employee_ID, Full_Name, BranchID, Salary FROM EMPLOYEE
ORDER BY BranchID, Salary DESC;
```



	Employee_ID	Full_Name	BranchID	Salary
1	1	Pham Quang Tuan	1	18000000
2	5	Do Duc Hieu	1	17000000
3	2	Nguyen Thi Hoa	2	15000000
4	3	Tran Van Khoa	3	22000000
5	4	Le Thi My Linh	3	16000000

- **Description:** List transactions from October 1st to October 15th, 2024, sorted by date. (TRANSACTIONS)

```
SELECT * FROM TRANSACTIONS
WHERE TransactionDate >= '2024-10-01' AND TransactionDate < '2024-10-16'
ORDER BY TransactionDate ASC;
```

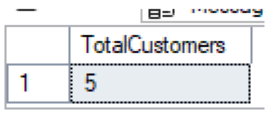


	TransactionID	TransactionDate	Amount	BalanceAfter	AccountID
1	1	2024-10-01 10:00:00.000	2000000.00	7000000.00	1
2	2	2024-10-05 14:30:00.000	-1500000.00	10500000.00	2
3	3	2024-10-10 09:45:00.000	500000.00	4000000.00	3
4	4	2024-10-12 16:00:00.000	-700000.00	6300000.00	4
5	5	2024-10-15 11:20:00.000	3000000.00	28000000.00	5

2.1.4. Aggregation (COUNT, AVG, AVG, MAX, MIN)

- **Description:** Count the total number of customers.

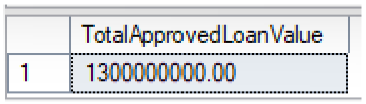
```
SELECT COUNT(CustomerID) AS TotalCustomers FROM CUSTOMER;
```



	TotalCustomers
1	5

- **Description:** Calculate the total LoanAmount for all 'Approved' loans.

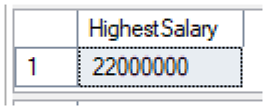
```
SELECT SUM(LoanAmount) AS TotalApprovedLoanValue FROM LOAN
WHERE Status = 'Approved';
```



	TotalApprovedLoanValue
1	1300000000.00

- **Description:** Find the highest Salary among all employees.

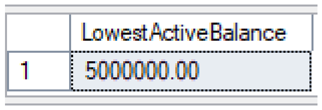
```
SELECT MAX(Salary) AS HighestSalary FROM EMPLOYEE;
```



	HighestSalary
1	22000000

- **Description:** Find the lowest Balance of any 'Active' account.

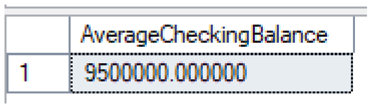
```
SELECT MIN(Balance) AS LowestActiveBalance FROM ACCOUNT
WHERE Status = 'Active';
```



	LowestActiveBalance
1	5000000.00

- **Description:** Calculate the average balance (AVG) for 'Checking' accounts.

```
SELECT AVG(Balance) AS AverageCheckingBalance FROM ACCOUNT
WHERE AccountType = N'Checking';
```



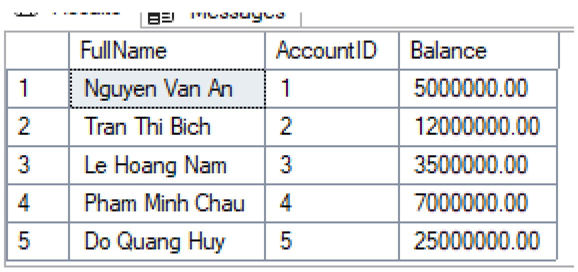
	AverageCheckingBalance
1	9500000.000000

2.2. Intermediate Queries

2.2.1. Join multiple tables

- Description (INNER JOIN 1): Retrieve the customer's FullName and the Balance of all accounts they own.

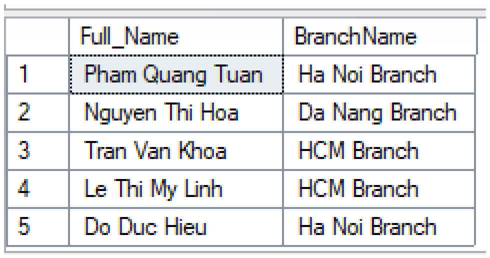
```
SELECT c.FullName, a.AccountID, a.Balance
FROM CUSTOMER c
INNER JOIN ACCOUNT a ON c.CustomerID = a.CustomerID;
```



	FullName	AccountID	Balance
1	Nguyen Van An	1	5000000.00
2	Tran Thi Bich	2	12000000.00
3	Le Hoang Nam	3	3500000.00
4	Pham Minh Chau	4	7000000.00
5	Do Quang Huy	5	25000000.00

- Description (INNER JOIN 2): Get the Full_Name of employees and the BranchName they work at.

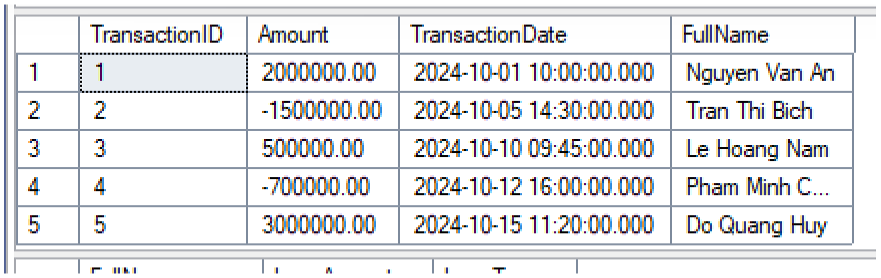
```
SELECT e.Full_Name, b.BranchName
FROM EMPLOYEE e
JOIN BRANCH b ON e.BranchID = b.BranchID;
```



	Full_Name	BranchName
1	Pham Quang Tuan	Ha Noi Branch
2	Nguyen Thi Hoa	Da Nang Branch
3	Tran Van Khoa	HCM Branch
4	Le Thi My Linh	HCM Branch
5	Do Duc Hieu	Ha Noi Branch

- Description (INNER JOIN 3): List all transactions with the FullName of the customer who owns the account.

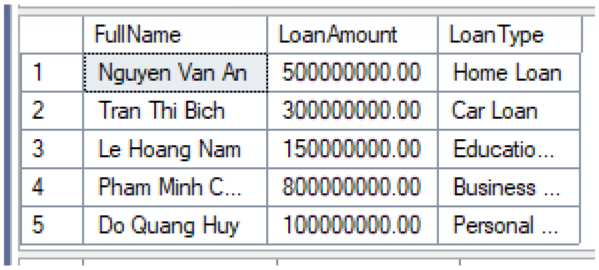
```
SELECT t.TransactionID, t.Amount, t.TransactionDate, c.FullName
FROM TRANSACTIONS t
JOIN ACCOUNT a ON t.AccountID = a.AccountID
JOIN CUSTOMER c ON a.CustomerID = c.CustomerID;
```



	TransactionID	Amount	TransactionDate	FullName
1	1	2000000.00	2024-10-01 10:00:00.000	Nguyen Van An
2	2	-1500000.00	2024-10-05 14:30:00.000	Tran Thi Bich
3	3	500000.00	2024-10-10 09:45:00.000	Le Hoang Nam
4	4	-700000.00	2024-10-12 16:00:00.000	Pham Minh C...
5	5	3000000.00	2024-10-15 11:20:00.000	Do Quang Huy

- Description (LEFT JOIN 1): List all customers and, if they have one, their LoanAmount. Customers without loans will still be listed with NULL.

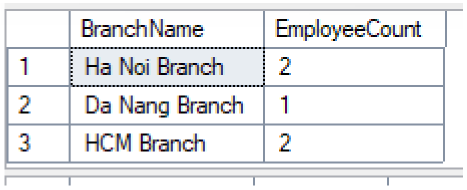
```
SELECT c.FullName, l.LoanAmount, l.LoanType
FROM CUSTOMER c
LEFT JOIN LOAN l ON c.CustomerID = l.CustomerID;
```



	FullName	LoanAmount	LoanType
1	Nguyen Van An	500000000.00	Home Loan
2	Tran Thi Bich	300000000.00	Car Loan
3	Le Hoang Nam	150000000.00	Educatio...
4	Pham Minh C...	800000000.00	Business ...
5	Do Quang Huy	100000000.00	Personal ...

- Description (LEFT JOIN 2): Show all branches and the count of employees in each (even if a branch has zero employees).

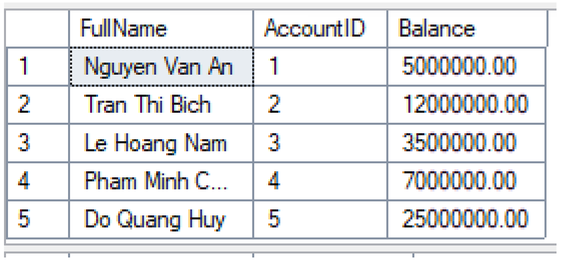
```
SELECT b.BranchName, COUNT(e.Employee_ID) AS EmployeeCount
FROM BRANCH b
LEFT JOIN EMPLOYEE e ON b.BranchID = e.BranchID
GROUP BY b.BranchName;
```



	BranchName	EmployeeCount
1	Ha Noi Branch	2
2	Da Nang Branch	1
3	HCM Branch	2

- Description (RIGHT JOIN): List all accounts and their customer details. This ensures all accounts are listed, even if they (hypothetically) had no matching customer.

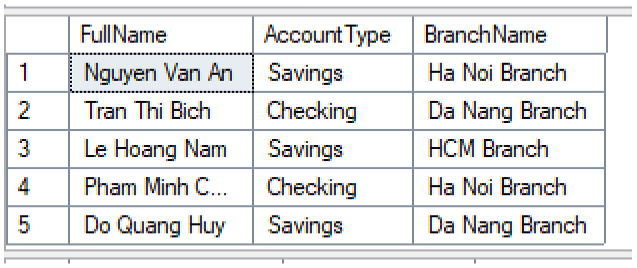
```
SELECT c.FullName, a.AccountID, a.Balance
FROM CUSTOMER c
RIGHT JOIN ACCOUNT a ON c.CustomerID = a.CustomerID;
```



	FullName	AccountID	Balance
1	Nguyen Van An	1	5000000.00
2	Tran Thi Bich	2	12000000.00
3	Le Hoang Nam	3	3500000.00
4	Pham Minh C...	4	7000000.00
5	Do Quang Huy	5	25000000.00

- Description (3-Table JOIN 1): Find the FullName of customers, their AccountType, and the BranchName where their account is held.

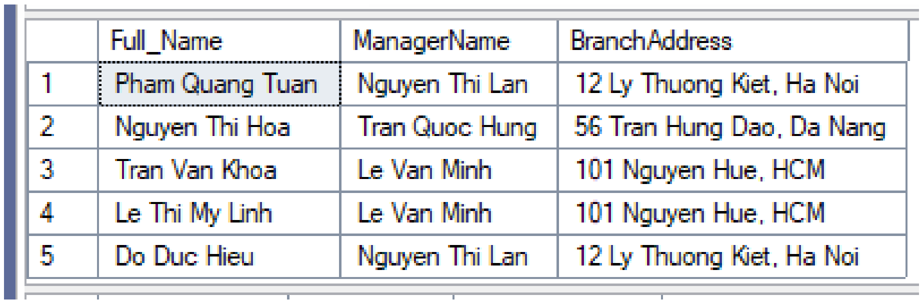
```
SELECT c.FullName, a.AccountType, b.BranchName
FROM CUSTOMER c
JOIN ACCOUNT a ON c.CustomerID = a.CustomerID
JOIN BRANCH b ON a.BranchID = b.BranchID;
```



	FullName	AccountType	BranchName
1	Nguyen Van An	Savings	Ha Noi Branch
2	Tran Thi Bich	Checking	Da Nang Branch
3	Le Hoang Nam	Savings	HCM Branch
4	Pham Minh C...	Checking	Ha Noi Branch
5	Do Quang Huy	Savings	Da Nang Branch

- Description (3-Table JOIN 2): List employees, their branch ManagerName, and the branch Address.

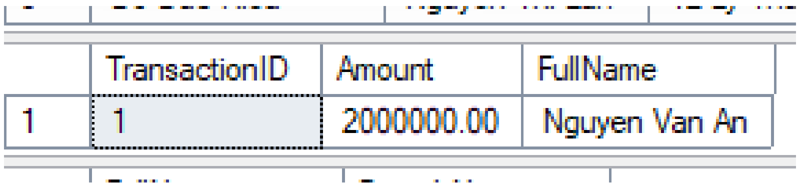
```
SELECT e.Full_Name, b.ManagerName, b.Address AS BranchAddress
FROM EMPLOYEE e
JOIN BRANCH b ON e.BranchID = b.BranchID;
```



	Full_Name	ManagerName	BranchAddress
1	Pham Quang Tuan	Nguyen Thi Lan	12 Ly Thuong Kiet, Ha Noi
2	Nguyen Thi Hoa	Tran Quoc Hung	56 Tran Hung Dao, Da Nang
3	Tran Van Khoa	Le Van Minh	101 Nguyen Hue, HCM
4	Le Thi My Linh	Le Van Minh	101 Nguyen Hue, HCM
5	Do Duc Hieu	Nguyen Thi Lan	12 Ly Thuong Kiet, Ha Noi

- Description (4-Table JOIN): Get transaction details (Amount, Date) for customers who have a 'Home Loan'.

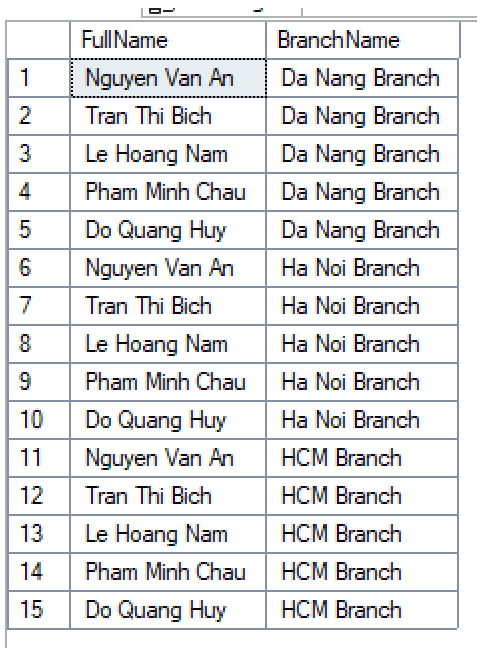
```
SELECT t.TransactionID, t.Amount, c.FullName
FROM TRANSACTIONS t
JOIN ACCOUNT a ON t.AccountID = a.AccountID
JOIN CUSTOMER c ON a.CustomerID = c.CustomerID
JOIN LOAN l ON c.CustomerID = l.CustomerID
WHERE l.LoanType = 'Home Loan';
```



	TransactionID	Amount	FullName
1	1	2000000.00	Nguyen Van An

- Description (CROSS JOIN): Create all possible pairings of CUSTOMER and BRANCH (e.g., for a marketing mail merge).

```
SELECT c.FullName, b.BranchName
FROM CUSTOMER c
CROSS JOIN BRANCH b;
```

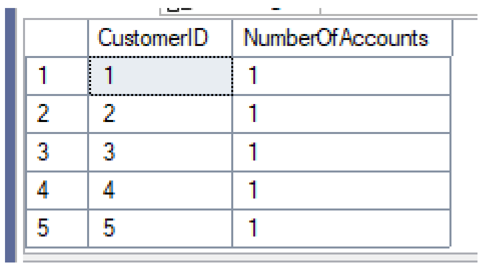


	FullName	BranchName
1	Nguyen Van An	Da Nang Branch
2	Tran Thi Bich	Da Nang Branch
3	Le Hoang Nam	Da Nang Branch
4	Pham Minh Chau	Da Nang Branch
5	Do Quang Huy	Da Nang Branch
6	Nguyen Van An	Ha Noi Branch
7	Tran Thi Bich	Ha Noi Branch
8	Le Hoang Nam	Ha Noi Branch
9	Pham Minh Chau	Ha Noi Branch
10	Do Quang Huy	Ha Noi Branch
11	Nguyen Van An	HCM Branch
12	Tran Thi Bich	HCM Branch
13	Le Hoang Nam	HCM Branch
14	Pham Minh Chau	HCM Branch
15	Do Quang Huy	HCM Branch

2.2.2. Group results (GROUP BY) and (HAVING)

- Description (GROUP BY 1): Count the number of accounts each CustomerID owns.

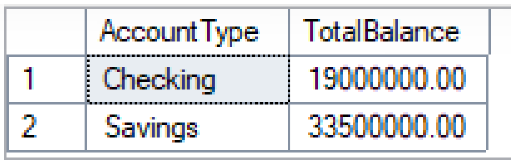
```
SELECT CustomerID, COUNT(AccountID) AS NumberOfAccounts
FROM ACCOUNT
GROUP BY CustomerID;
```



	CustomerID	NumberOfAccounts
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1

- Description (GROUP BY 2): Calculate the total Balance for each AccountType.

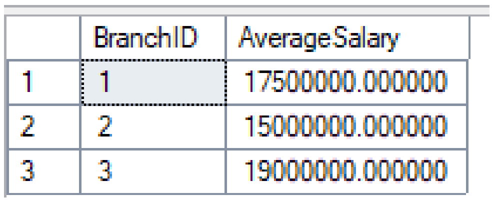
```
SELECT AccountType, SUM(Balance) AS TotalBalance
FROM ACCOUNT
GROUP BY AccountType;
```



	AccountType	TotalBalance
1	Checking	19000000.00
2	Savings	33500000.00

- Description (GROUP BY 3): Find the average Salary for each BranchID.

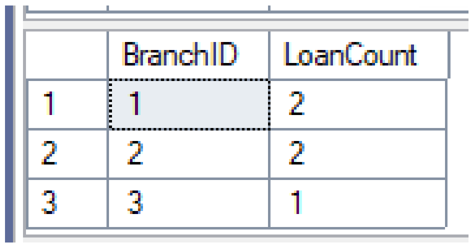
```
SELECT BranchID, AVG(Salary) AS AverageSalary
FROM EMPLOYEE
GROUP BY BranchID;
```



	BranchID	AverageSalary
1	1	17500000.000000
2	2	15000000.000000
3	3	19000000.000000

- Description (GROUP BY 4): Count the number of loans per BranchID.

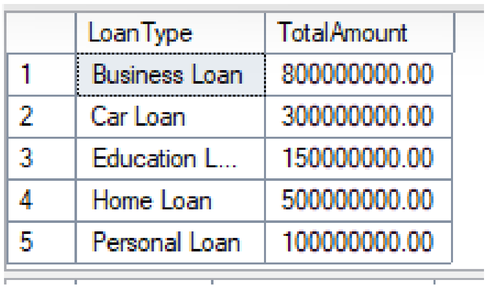
```
SELECT BranchID, COUNT(LoanID) AS LoanCount
FROM LOAN
GROUP BY BranchID;
```



	BranchID	LoanCount
1	1	2
2	2	2
3	3	1

- Description (GROUP BY 5): Find the total LoanAmount grouped by LoanType.

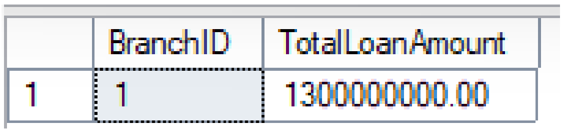
```
SELECT LoanType, SUM(LoanAmount) AS TotalAmount
FROM LOAN
GROUP BY LoanType;
```



	LoanType	TotalAmount
1	Business Loan	800000000.00
2	Car Loan	300000000.00
3	Education L...	150000000.00
4	Home Loan	500000000.00
5	Personal Loan	100000000.00

- Description (GROUP BY / HAVING 1): Calculate the total LoanAmount at each BranchID, but only show branches with a total loan sum greater than 700,000,000.

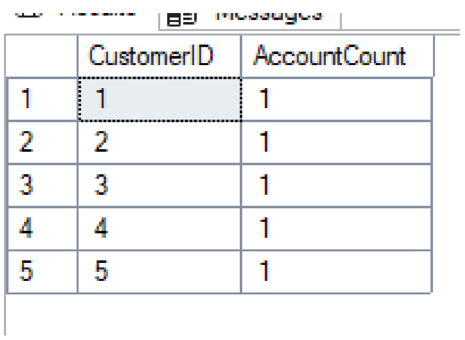
```
SELECT BranchID, SUM(LoanAmount) AS TotalLoanAmount
FROM LOAN
GROUP BY BranchID
HAVING SUM(LoanAmount) > 700000000;
```



	BranchID	TotalLoanAmount
1	1	1300000000.00

- Description (GROUP BY / HAVING 2): Show CustomerIDs who have equal or more than one account.

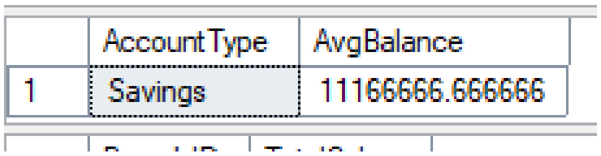
```
SELECT CustomerID, COUNT(AccountID) AS AccountCount
FROM ACCOUNT
GROUP BY CustomerID
HAVING COUNT(AccountID) >= 1;
```



	CustomerID	AccountCount
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1

- Description (GROUP BY / HAVING 3): Show AccountTypes where the average balance is greater than 10,000,000.

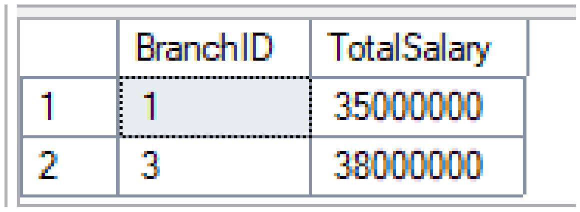
```
SELECT AccountType, AVG(Balance) AS AvgBalance
FROM ACCOUNT
GROUP BY AccountType
HAVING AVG(Balance) > 10000000;
```



	AccountType	AvgBalance
1	Savings	11166666.666666

- Description (GROUP BY / HAVING 4): List branches where the total employee salary budget exceeds 30,000,000.

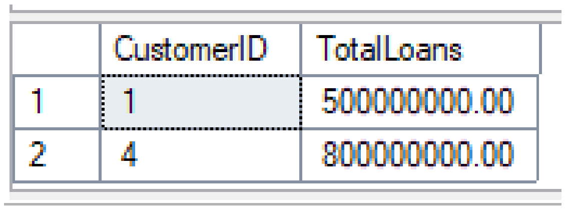
```
SELECT BranchID, SUM(Salary) AS TotalSalary
FROM EMPLOYEE
GROUP BY BranchID
HAVING SUM(Salary) > 30000000;
```



	BranchID	TotalSalary
1	1	35000000
2	3	38000000

- Description (GROUP BY / HAVING 5): Show customers who have a total loan amount greater than 400,000,000.

```
SELECT CustomerID, SUM(LoanAmount) AS TotalLoans
FROM LOAN
GROUP BY CustomerID
HAVING SUM(LoanAmount) > 400000000;
```

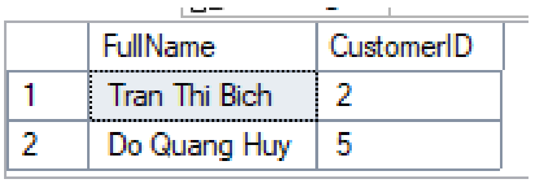


	CustomerID	TotalLoans
1	1	500000000.00
2	4	800000000.00

2.2.3. Subqueries (in WHERE or FROM)

- Description (Subquery in WHERE 1): Find the FullName of all customers who have at least one account with a balance over 10,000,000.

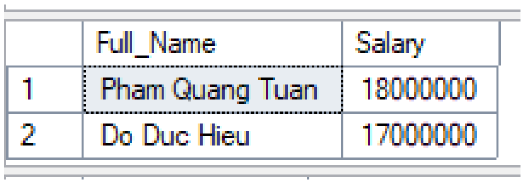
```
SELECT FullName, CustomerID FROM CUSTOMER
WHERE CustomerID IN (
   SELECT CustomerID FROM ACCOUNT WHERE Balance > 10000000
);
```



	FullName	CustomerID
1	Tran Thi Bich	2
2	Do Quang Huy	5

- Description (Subquery in WHERE 2): List all employees who work at the 'Ha Noi Branch'.

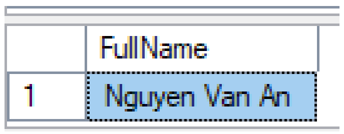
```
SELECT Full_Name, Salary FROM EMPLOYEE
WHERE BranchID = (
   SELECT BranchID FROM BRANCH WHERE BranchName = 'Ha Noi Branch'
);
```



	Full_Name	Salary
1	Pham Quang Tuan	18000000
2	Do Duc Hieu	17000000

- Description (Subquery in WHERE 3): Find customers who have a 'Home Loan'.

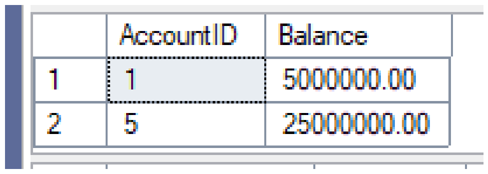
```
SELECT FullName FROM CUSTOMER
WHERE CustomerID IN (
   SELECT CustomerID FROM LOAN WHERE LoanType = 'Home Loan'
);
```



	FullName
1	Nguyen Van An

- Description (Subquery in WHERE 4): Find accounts that have had transactions greater than 1,000,000.

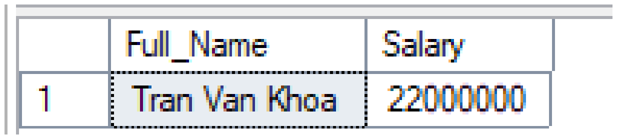
```
SELECT AccountID, Balance FROM ACCOUNT
WHERE AccountID IN (
   SELECT AccountID FROM TRANSACTIONS WHERE Amount > 1000000
);
```



	AccountID	Balance
1	1	5000000.00
2	5	25000000.00

- Description (Subquery in WHERE 5): Find employees who earn more than the average salary of employees in 'HCM Branch'.

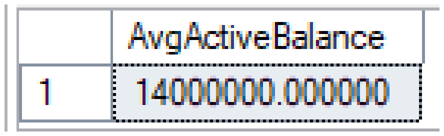
```
SELECT Full_Name, Salary FROM EMPLOYEE
WHERE Salary > (
   SELECT AVG(Salary) FROM EMPLOYEE
   WHERE BranchID = (SELECT BranchID FROM BRANCH WHERE BranchName = 'HCM
Branch')
);
```



	Full_Name	Salary
1	Tran Van Khoa	22000000

- Description (Subquery in FROM 1): Select the average balance from the derived table of 'Active' accounts.

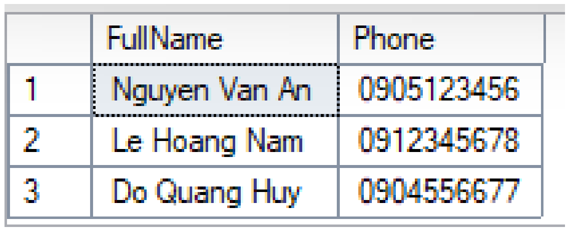
```
SELECT AVG(ActiveAccounts.Balance) AS AvgActiveBalance
FROM (
   SELECT Balance FROM ACCOUNT WHERE Status = 'Active'
) AS ActiveAccounts;
```



	AvgActiveBalance
1	14000000.000000

- Description (Subquery in FROM 2): Get the details of customers who own 'Savings' accounts, using a subquery in the JOIN clause.

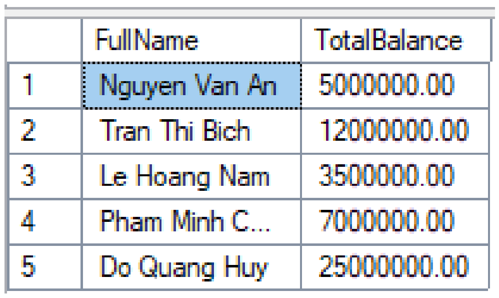
```
SELECT c.FullName, c.Phone
FROM CUSTOMER c
JOIN (
   SELECT CustomerID FROM ACCOUNT WHERE AccountType = 'Savings'
) AS sa ON c.CustomerID = sa.CustomerID;
```



	FullName	Phone
1	Nguyen Van An	0905123456
2	Le Hoang Nam	0912345678
3	Do Quang Huy	0904556677

- Description (Correlated Subquery in SELECT): Show each customer's FullName and their total balance calculated with a subquery.

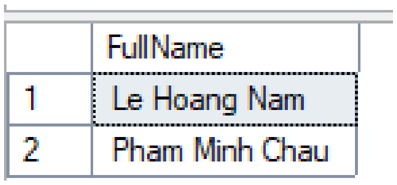
```
SELECT c.FullName, (
   SELECT SUM(a.Balance)
   FROM ACCOUNT a
   WHERE a.CustomerID = c.CustomerID
) AS TotalBalance
FROM CUSTOMER c;
```



	FullName	TotalBalance
1	Nguyen Van An	5000000.00
2	Tran Thi Bich	12000000.00
3	Le Hoang Nam	3500000.00
4	Pham Minh C...	7000000.00
5	Do Quang Huy	25000000.00

- Description (Subquery in WHERE with NOT IN): Find customers who have no 'Active' accounts.

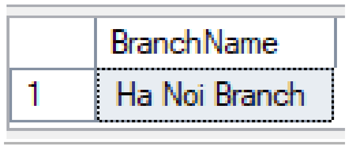
```
SELECT FullName FROM CUSTOMER
WHERE CustomerID NOT IN (
   SELECT CustomerID FROM ACCOUNT WHERE Status = 'Active'
);
```



	FullName
1	Le Hoang Nam
2	Pham Minh Chau

- Description (Subquery in WHERE): Find branches that have at least one 'Approved' loan.

```
SELECT BranchName FROM BRANCH
WHERE BranchID IN (
   SELECT BranchID FROM LOAN WHERE Status = 'Approved'
);
```



	BranchName
1	Ha Noi Branch

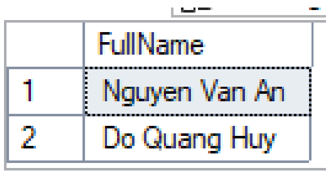
2.3. Advanced Queries

2.3.1.

Nested subqueries

- Description (Nested): Find the names of customers who have an 'Active' account and have made a transaction (deposit) over 1,000,000.

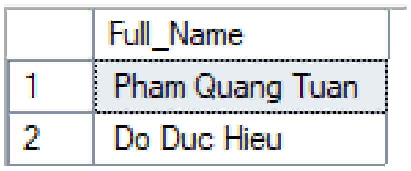
```
SELECT FullName FROM CUSTOMER
WHERE CustomerID IN (
   SELECT CustomerID FROM ACCOUNT
   WHERE Status = 'Active' AND AccountID IN (
      SELECT AccountID FROM TRANSACTIONS WHERE Amount > 1000000
   )
);
```



	FullName
1	Nguyen Van An
2	Do Quang Huy

- Description (Nested): Find employees who work in a branch that manages a 'Business Loan'.

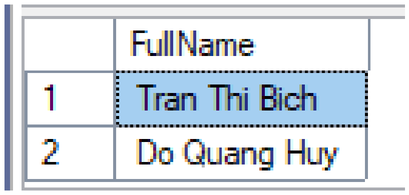
```
SELECT Full_Name FROM EMPLOYEE
WHERE BranchID IN (
   SELECT BranchID FROM BRANCH
   WHERE BranchID IN (
      SELECT BranchID FROM LOAN WHERE LoanType = 'Business Loan'
   )
);
```



	Full_Name
1	Pham Quang Tuan
2	Do Duc Hieu

- Description (Nested): Find the FullName of customers who have a balance higher than the average balance of all 'Savings' accounts.

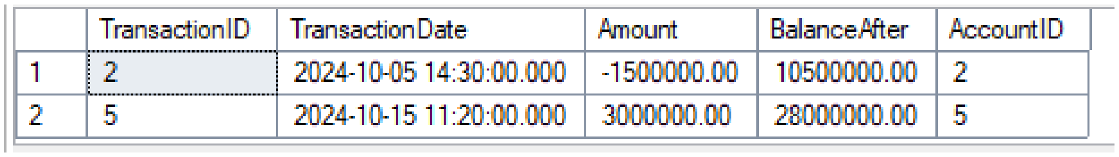
```
SELECT FullName FROM CUSTOMER
WHERE CustomerID IN (
   SELECT CustomerID FROM ACCOUNT
   WHERE Balance > (
      SELECT AVG(Balance) FROM ACCOUNT WHERE AccountType = 'Savings'
   )
);
```



	FullName
1	Tran Thi Bich
2	Do Quang Huy

- Description (Nested): List all transactions for accounts held at the 'Da Nang Branch'.

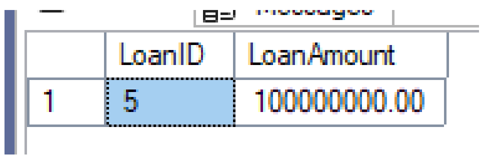
```
SELECT * FROM TRANSACTIONS
WHERE AccountID IN (
   SELECT AccountID FROM ACCOUNT
   WHERE BranchID = (
      SELECT BranchID FROM BRANCH WHERE BranchName = 'Da Nang Branch'
   )
);
```



	TransactionID	TransactionDate	Amount	BalanceAfter	AccountID
1	2	2024-10-05 14:30:00.000	-1500000.00	10500000.00	2
2	5	2024-10-15 11:20:00.000	3000000.00	28000000.00	5

- Description (Nested): Find the 'Default' loans for customers who have an 'Active' or an 'Checking' account.

```
SELECT LoanID, LoanAmount FROM LOAN
WHERE Status = 'Default'
AND CustomerID IN (
   SELECT CustomerID FROM ACCOUNT
   WHERE AccountType = 'Checking' OR Status = 'Active'
);
```



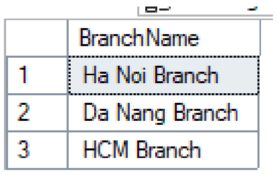
	LoanID	LoanAmount
1	5	100000000.00

2.3.2.

Use of EXISTS, IN, and ANY/ALL

- Description (EXISTS 1): Retrieve the BranchName of all branches that manage at least one LOAN.

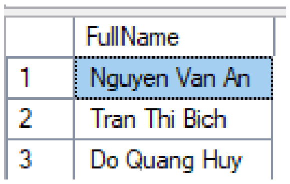
```
SELECT b.BranchName
FROM BRANCH b
WHERE EXISTS (
   SELECT 1 FROM LOAN l WHERE l.BranchID = b.BranchID
);
```



	BranchName
1	Ha Noi Branch
2	Da Nang Branch
3	HCM Branch

- Description (EXISTS 2): Find all customers who have at least one 'Active' account.

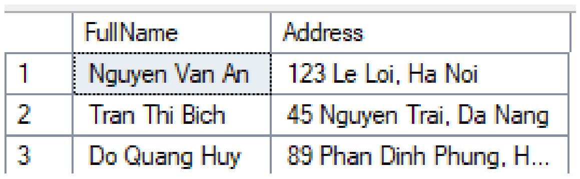
```
SELECT c.FullName
FROM CUSTOMER c
WHERE EXISTS (
   SELECT 1 FROM ACCOUNT a WHERE a.CustomerID = c.CustomerID AND a.Status =
'Active'
);
```



	FullName
1	Nguyen Van An
2	Tran Thi Bich
3	Do Quang Huy

- Description (IN): Find customers who live in 'Ha Noi' or 'Da Nang'. (Note: IN was also used in 2.2.3).

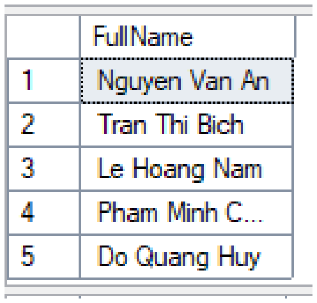
```
SELECT FullName, Address FROM CUSTOMER
WHERE Address LIKE '%Ha Noi' OR Address LIKE '%Da Nang';
-- A better example if we had a City column:
-- WHERE City IN ('Ha Noi', 'Da Nang');
```



	FullName	Address
1	Nguyen Van An	123 Le Loi, Ha Noi
2	Tran Thi Bich	45 Nguyen Trai, Da Nang
3	Do Quang Huy	89 Phan Dinh Phung, H...

- Description (ANY): Find customers whose CustomerID matches any CustomerID in the LOAN table (equivalent to IN).

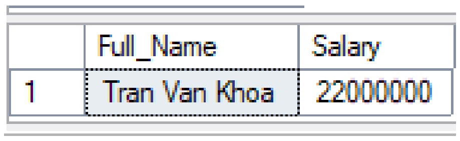
```
SELECT FullName FROM CUSTOMER
WHERE CustomerID = ANY (
   SELECT CustomerID FROM LOAN
);
```



	FullName
1	Nguyen Van An
2	Tran Thi Bich
3	Le Hoang Nam
4	Pham Minh C...
5	Do Quang Huy

- Description (ALL): Find the employee(s) with the highest salary.

```
SELECT Full_Name, Salary FROM EMPLOYEE
WHERE Salary >= ALL (
   SELECT Salary FROM EMPLOYEE
);
```



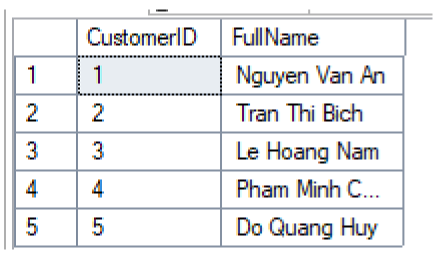
	Full_Name	Salary
1	Tran Van Khoa	22000000

2.3.3.

Set Operations (UNION, INTERSECT, EXCEPT)

- Description (INTERSECT): Get a list (CustomerID, FullName) of customers who have *both* an ACCOUNT and a LOAN with the bank.

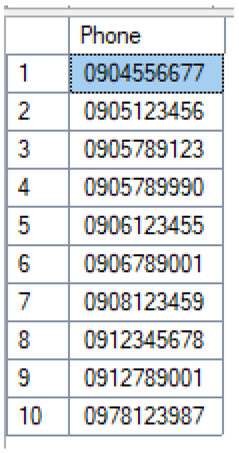
```
SELECT CustomerID, FullName FROM CUSTOMER
WHERE CustomerID IN (SELECT CustomerID FROM ACCOUNT)
INTERSECT
SELECT CustomerID, FullName FROM CUSTOMER
WHERE CustomerID IN (SELECT CustomerID FROM LOAN);
```



	CustomerID	FullName
1	1	Nguyen Van An
2	2	Tran Thi Bich
3	3	Le Hoang Nam
4	4	Pham Minh C...
5	5	Do Quang Huy

- Description (UNION): Get a combined list of all Phone numbers from CUSTOMER and EMPLOYEE tables (duplicates removed).

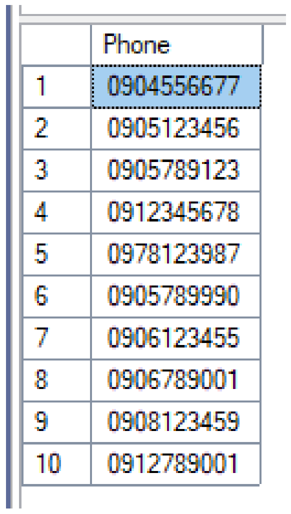
```
SELECT Phone FROM CUSTOMER
UNION
SELECT Phone_Number FROM EMPLOYEE;
```



	Phone
1	0904556677
2	0905123456
3	0905789123
4	0905789990
5	0906123455
6	0906789001
7	0908123459
8	0912345678
9	0912789001
10	0978123987

- Description (UNION ALL): Get a combined list of all Phone numbers from CUSTOMER and EMPLOYEE tables (including duplicates).

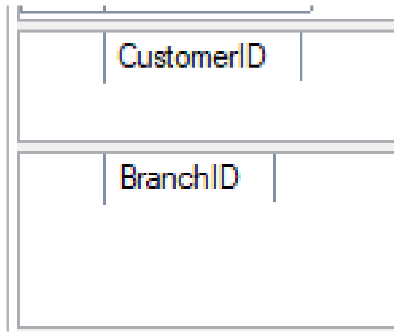
```
SELECT Phone FROM CUSTOMER
UNION ALL
SELECT Phone_Number FROM EMPLOYEE;
```



	Phone
1	0904556677
2	0905123456
3	0905789123
4	0912345678
5	0978123987
6	0905789990
7	0906123455
8	0906789001
9	0908123459
10	0912789001

- Description (EXCEPT 1): Find customers who have an ACCOUNT but do *not* have a LOAN.

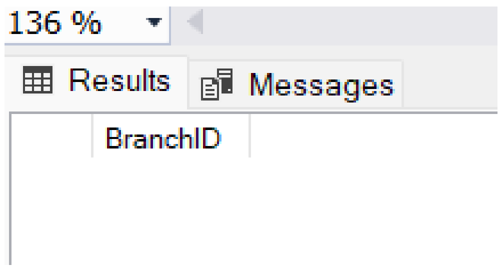
```
SELECT CustomerID FROM ACCOUNT
EXCEPT
SELECT CustomerID FROM LOAN;
```



```
CustomerID

BranchID
```

- Description (EXCEPT 2): Find branches that have EMPLOYEEs but no LOANs.

```
SELECT BranchID FROM EMPLOYEE
EXCEPT
SELECT BranchID FROM LOAN;
```



```
BranchID
```

3. Functions

Per the requirement, here are four user-defined functions (UDFs) relevant to the banking domain.

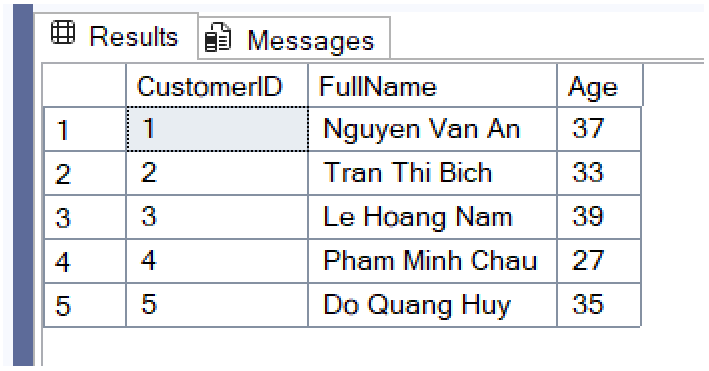
3.1. Function 1: fn_CalculateCustomerAge (Scalar)

- **Description:** Calculates a customer's current age based on their Date of Birth (DOB).

```
CREATE FUNCTION fn_CalculateCustomerAge (@DOB DATE)
RETURNS INT
AS
BEGIN
   RETURN DATEDIFF(YEAR, @DOB, GETDATE()) -
        CASE
           WHEN (MONTH(@DOB) > MONTH(GETDATE())) OR
              (MONTH(@DOB) = MONTH(GETDATE()) AND DAY(@DOB) >
DAY(GETDATE()))
           THEN 1
           ELSE 0
        END;
END;
```

- Execution:

```
SELECT CustomerID, FullName, dbo.fn_CalculateCustomerAge(DOB) AS Age
FROM CUSTOMER;
```



	CustomerID	FullName	Age
1	1	Nguyen Van An	37
2	2	Tran Thi Bich	33
3	3	Le Hoang Nam	39
4	4	Pham Minh Chau	27
5	5	Do Quang Huy	35

3.2. Function 2: fn_GetCustomerTotalBalance (Scalar)

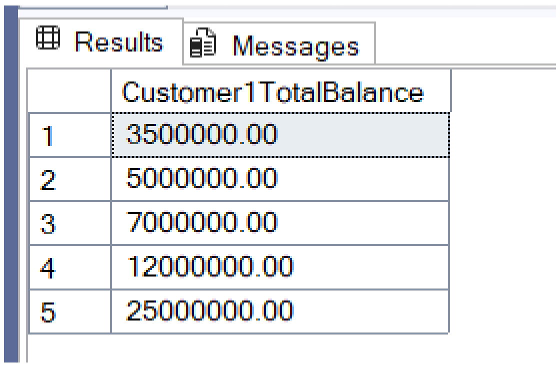
- **Description:** Calculates the total balance of a specific customer (CustomerID) by summing the balances of all their accounts.

```
CREATE FUNCTION fn_GetCustomerTotalBalance (@CustomerID INT)
RETURNS DECIMAL(18, 2)
```

```
AS
BEGIN
   DECLARE @TotalBalance DECIMAL(18, 2);
   SELECT @TotalBalance = SUM(Balance)
   FROM ACCOUNT
   WHERE CustomerID = @CustomerID;
   RETURN ISNULL(@TotalBalance, 0);
END;
```

- Execution:

```
SELECT DISTINCT dbo.fn_GetCustomerTotalBalance(CustomerID)
AS Customer1TotalBalance
FROM ACCOUNT
```



```
   Customer1TotalBalance
1  3500000.00
2  5000000.00
3  7000000.00
4  12000000.00
5  25000000.00
```

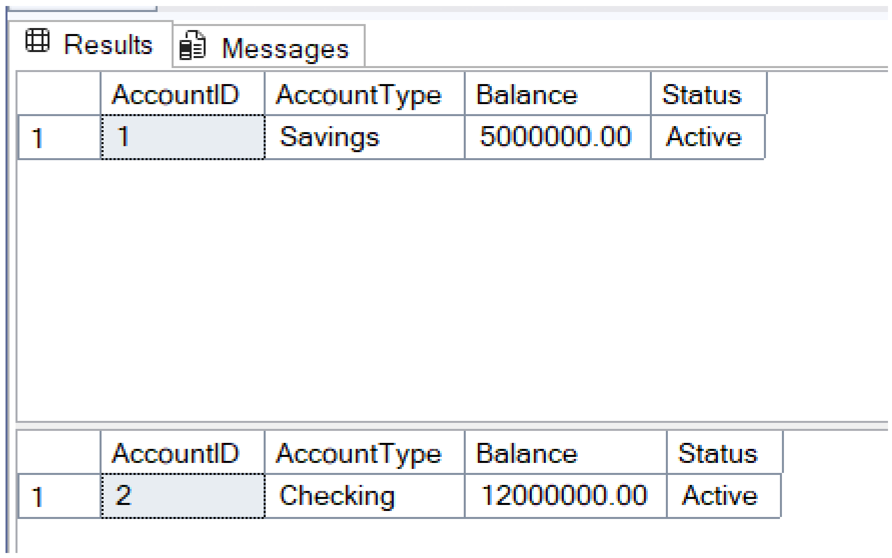
3.3. Function 3: fn_GetAccountsByCustomer (Table-Valued)

- **Description:** Returns a table containing all accounts (AccountID, AccountType, Balance, Status) for a specific CustomerID.

```
CREATE FUNCTION fn_GetAccountsByCustomer (@CustomerID INT)
RETURNS TABLE
AS
RETURN (
   SELECT AccountID, AccountType, Balance, Status
   FROM ACCOUNT
   WHERE CustomerID = @CustomerID
);
```

- Execution:

```
    SELECT * FROM dbo.fn_GetAccountsByCustomer(1);
    SELECT * FROM dbo.fn_GetAccountsByCustomer(2);
```



Results | Messages

	AccountID	AccountType	Balance	Status
1	1	Savings	5000000.00	Active

	AccountID	AccountType	Balance	Status
1	2	Checking	12000000.00	Active

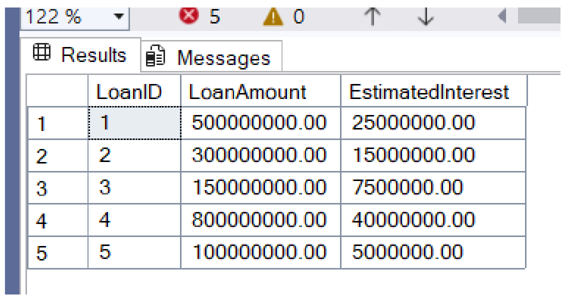
3.4. Function 4: fn_CalculateSimpleInterest (Scalar)

- **Description:** A helper function to calculate the simple interest on a loan, demonstrating a calculation function.

```
CREATE FUNCTION fn_CalculateSimpleInterest
(
   @LoanAmount DECIMAL(18, 2),
   @AnnualRate FLOAT,
   @Years INT
)
RETURNS DECIMAL(18, 2)
AS
BEGIN
   RETURN @LoanAmount * @AnnualRate * @Years;
END;
```

- Execution:

```
SELECT LoanID, LoanAmount, dbo.fn_CalculateSimpleInterest(LoanAmount, 0.05, 1) AS
EstimatedInterest FROM LOAN;
```



122 % | 5 | 0

Results | Messages

	LoanID	LoanAmount	EstimatedInterest
1	1	500000000.00	25000000.00
2	2	300000000.00	15000000.00
3	3	150000000.00	7500000.00
4	4	800000000.00	40000000.00
5	5	100000000.00	5000000.00

4. Stored Procedures

Here are four stored procedures that perform multi-step operations.

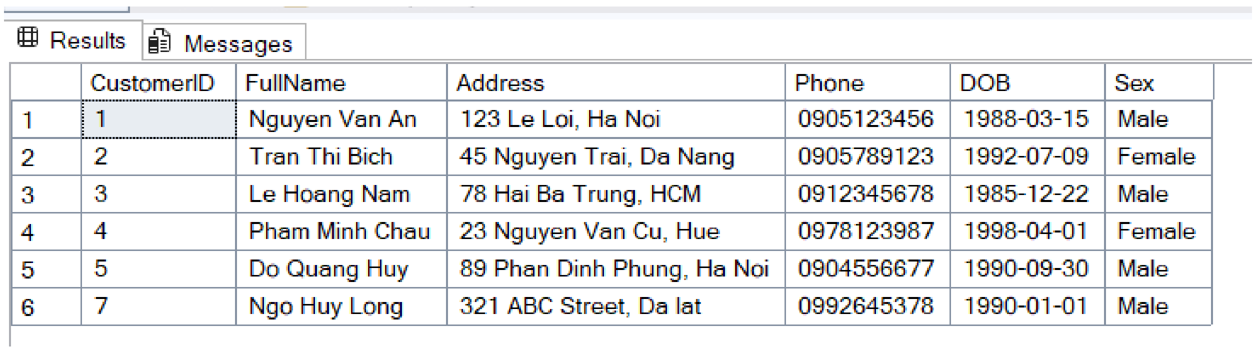
4.1. Procedure 1: sp_AddNewCustomer

- **Description:** Inserts a new customer record into the CUSTOMER table.

```
CREATE PROCEDURE sp_AddNewCustomer
   @CustomerID INT,
   @FullName NVARCHAR(100),
   @DOB DATE,
   @Address NVARCHAR(255),
   @Phone VARCHAR(20),
   @Sex VARCHAR(10)
AS
BEGIN
   INSERT INTO CUSTOMER (CustomerID, FullName, DOB, Address, Phone, Sex)
   VALUES (@CustomerID, @FullName, @DOB, @Address, @Phone, @Sex);
END;
```

- Execution:

```
EXEC sp_AddNewCustomer 7, N'Ngo Huy Long', '1990-01-01', N'321 ABC Street, Da lat',
'0992645378', 'Male';
```


Results | Messages

	CustomerID	FullName	Address	Phone	DOB	Sex
1	1	Nguyen Van An	123 Le Loi, Ha Noi	0905123456	1988-03-15	Male
2	2	Tran Thi Bich	45 Nguyen Trai, Da Nang	0905789123	1992-07-09	Female
3	3	Le Hoang Nam	78 Hai Ba Trung, HCM	0912345678	1985-12-22	Male
4	4	Pham Minh Chau	23 Nguyen Van Cu, Hue	0978123987	1998-04-01	Female
5	5	Do Quang Huy	89 Phan Dinh Phung, Ha Noi	0904556677	1990-09-30	Male
6	7	Ngo Huy Long	321 ABC Street, Da lat	0992645378	1990-01-01	Male

4.2. Procedure 2: sp_PerformTransaction

- **Description:** Performs a multi-step operation: updates an account balance and inserts a record into the TRANSACTIONS table, all within a single database transaction.

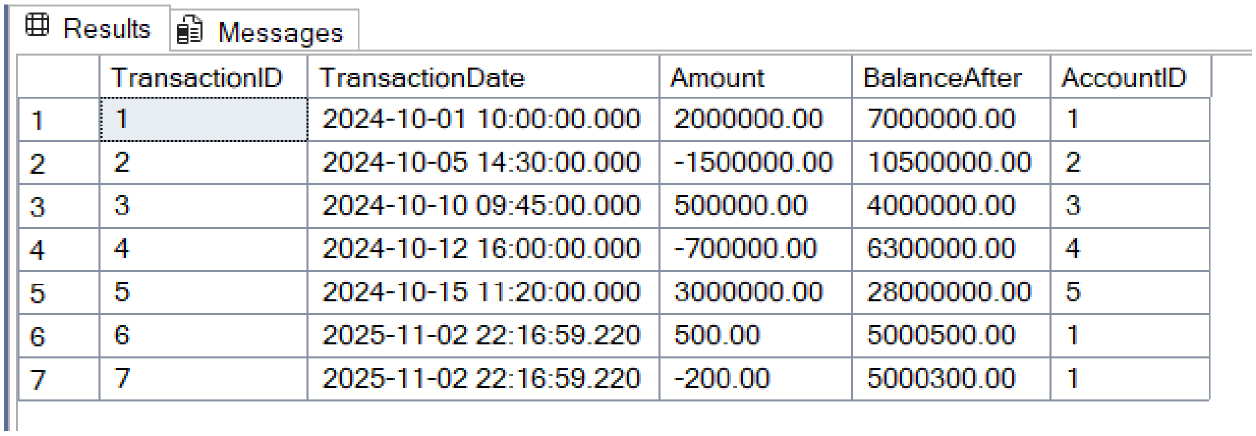
```
CREATE PROCEDURE sp_PerformTransaction
   @AccountID INT,
   @Amount DECIMAL(12,2)
AS
BEGIN
   DECLARE @CurrentBalance DECIMAL(12,2);
   DECLARE @NextTransactionID INT;

   SELECT @CurrentBalance = Balance
   FROM ACCOUNT
   WHERE AccountID = @AccountID;

   IF @CurrentBalance IS NULL
      PRINT 'Account not found.';
   ELSE IF (@CurrentBalance + @Amount) < 0
      PRINT 'Insufficient funds to complete transaction.';
   ELSE
   BEGIN
      UPDATE ACCOUNT
      SET Balance = Balance + @Amount
      WHERE AccountID = @AccountID;

      SELECT @NextTransactionID = ISNULL(MAX(TransactionID), 0) + 1
      FROM TRANSACTIONS;

      INSERT INTO TRANSACTIONS (TransactionID, TransactionDate, Amount,
BalanceAfter, AccountID)
      VALUES (
         @NextTransactionID,
         GETDATE(),
         @Amount,
         (SELECT Balance FROM ACCOUNT WHERE AccountID = @AccountID),
         @AccountID
      );

      PRINT 'Transaction successful.';
   END
END;
```

- Execution (Deposit 500):

```
EXEC sp_PerformTransaction 1, 500.00;
```

- Execution (Withdraw 200):

```
EXEC sp_PerformTransaction 1, -200.00;
```



Results | Messages

	TransactionID	TransactionDate	Amount	BalanceAfter	AccountID
1	1	2024-10-01 10:00:00.000	2000000.00	7000000.00	1
2	2	2024-10-05 14:30:00.000	-1500000.00	10500000.00	2
3	3	2024-10-10 09:45:00.000	500000.00	4000000.00	3
4	4	2024-10-12 16:00:00.000	-700000.00	6300000.00	4
5	5	2024-10-15 11:20:00.000	3000000.00	28000000.00	5
6	6	2025-11-02 22:16:59.220	500.00	5000500.00	1
7	7	2025-11-02 22:16:59.220	-200.00	5000300.00	1

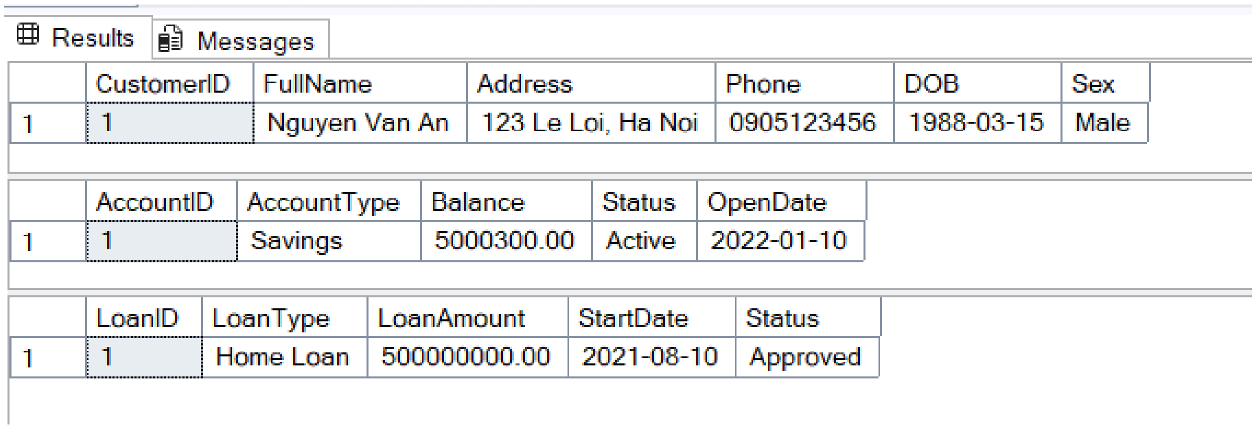
4.3. Procedure 3: sp_GetCustomerBankingSummary

- **Description:** Generates a summary report for a specific customer, returning their personal details, account list, and loan list.

```
CREATE PROCEDURE sp_GetCustomerBankingSummary @CustomerID INT
AS
BEGIN
   -- 1. Customer Details
   SELECT * FROM CUSTOMER WHERE CustomerID = @CustomerID;

   -- 2. Account List
   SELECT AccountID, AccountType, Balance, Status, OpenDate
   FROM ACCOUNT WHERE CustomerID = @CustomerID;

   -- 3. Loan List
   SELECT LoanID, LoanType, LoanAmount, StartDate, Status
   FROM LOAN WHERE CustomerID = @CustomerID;
END;
```

- Execution:

```
EXEC sp_GetCustomerBankingSummary 1;
```



Results | Messages

	CustomerID	FullName	Address	Phone	DOB	Sex
1	1	Nguyen Van An	123 Le Loi, Ha Noi	0905123456	1988-03-15	Male

	AccountID	AccountType	Balance	Status	OpenDate
1	1	Savings	5000300.00	Active	2022-01-10

	LoanID	LoanType	LoanAmount	StartDate	Status
1	1	Home Loan	500000000.00	2021-08-10	Approved

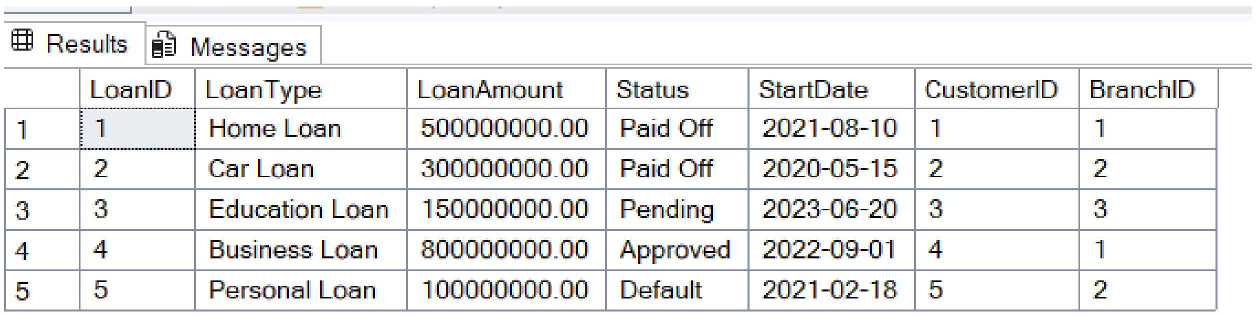
4.4. Procedure 4: sp_UpdateLoanStatus

- **Description:** A procedure to update related data, specifically changing the Status of a loan (e.g., 'Active', 'PaidOff', 'Default').

```
CREATE PROCEDURE sp_UpdateLoanStatus
   @LoanID INT,
   @NewStatus NVARCHAR(20)
AS
BEGIN
   UPDATE LOAN
   SET Status = @NewStatus
   WHERE LoanID = @LoanID;
END;
```

- Execution:

```
EXEC sp_UpdateLoanStatus 1, N'Paid Off';
```



Results | Messages

	LoanID	LoanType	LoanAmount	Status	StartDate	CustomerID	BranchID
1	1	Home Loan	500000000.00	Paid Off	2021-08-10	1	1
2	2	Car Loan	300000000.00	Paid Off	2020-05-15	2	2
3	3	Education Loan	150000000.00	Pending	2023-06-20	3	3
4	4	Business Loan	800000000.00	Approved	2022-09-01	4	1
5	5	Personal Loan	100000000.00	Default	2021-02-18	5	2

5. Triggers

Here are four triggers to enforce business rules and maintain consistency.

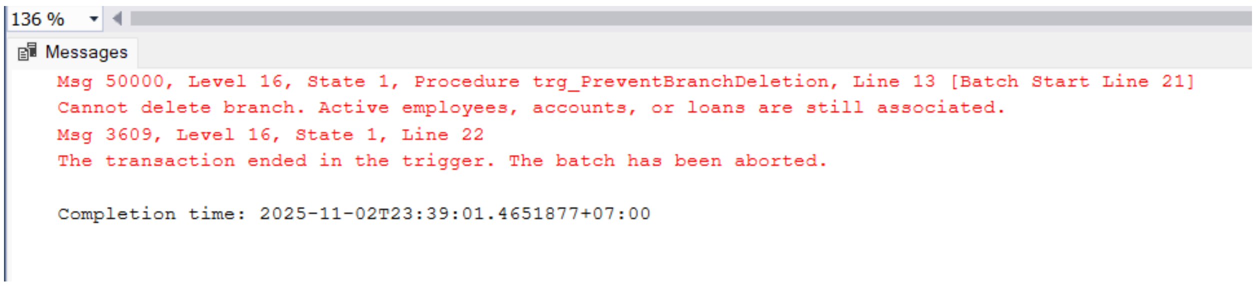
5.1. Trigger 1: trg_PreventBranchDeletion

- **Description:** This trigger prevents the deletion of a BRANCH record (parent record) if there are still associated child records (employees, accounts, or loans) linked to it. This helps ensure referential integrity.

```
CREATE TRIGGER trg_PreventBranchDeletion
ON BRANCH
INSTEAD OF DELETE
AS
BEGIN
   DECLARE @BranchID INT;
   SELECT @BranchID = d.BranchID FROM deleted AS d;

   IF EXISTS (SELECT 1 FROM EMPLOYEE WHERE BranchID = @BranchID) OR
      EXISTS (SELECT 1 FROM ACCOUNT WHERE BranchID = @BranchID) OR
      EXISTS (SELECT 1 FROM LOAN WHERE BranchID = @BranchID)
   BEGIN
       RAISERROR('Cannot delete branch. Active employees, accounts, or loans are still
associated.', 16, 1);
       ROLLBACK TRANSACTION;
   END
   ELSE
   BEGIN
       DELETE FROM BRANCH WHERE BranchID = @BranchID;
   END
END;
```

- Execution: The command fails and returns the error: Cannot delete branch...

```
DELETE FROM BRANCH WHERE BranchID = 1;
```



```
Msg 50000, Level 16, State 1, Procedure trg_PreventBranchDeletion, Line 13 [Batch Start Line 21]
Cannot delete branch. Active employees, accounts, or loans are still associated.
Msg 3609, Level 16, State 1, Line 22
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2025-11-02T23:39:01.4651877+07:00
```

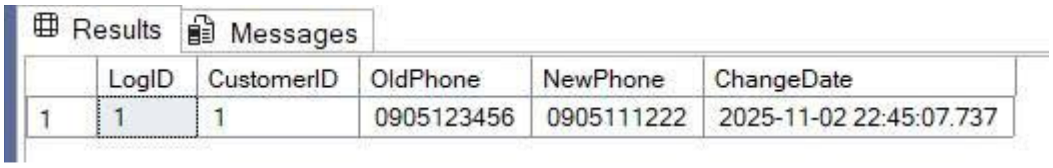
5.2. Trigger 2: trg_LogCustomerPhoneChange

- **Description:** Automatically logs changes to an audit table (Audit_CustomerChanges) whenever a customer's Phone number is updated.

```
CREATE TABLE Audit_CustomerChanges (
   LogID INT IDENTITY PRIMARY KEY,
   CustomerID INT,
   OldPhone VARCHAR(20),
   NewPhone VARCHAR(20),
   ChangeDate DATETIME
);

CREATE TRIGGER trg_LogCustomerPhoneChange
ON CUSTOMER
AFTER UPDATE
AS
BEGIN
   IF UPDATE(Phone)
   BEGIN
      INSERT INTO Audit_CustomerChanges (CustomerID, OldPhone, NewPhone,
ChangeDate)
      SELECT
         i.CustomerID,
         d.Phone,
         i.Phone,
         GETDATE()
      FROM
         inserted AS i
      INNER JOIN
         deleted AS d ON i.CustomerID = d.CustomerID
      WHERE
         i.Phone <> d.Phone;
   END
END;
```

- Execution:

```
UPDATE CUSTOMER SET Phone = '0905111222' WHERE CustomerID = 1;
SELECT * FROM Audit_CustomerChanges; -- check
```



	LogID	CustomerID	OldPhone	NewPhone	ChangeDate
1	1	1	0905123456	0905111222	2025-11-02 22:45:07.737

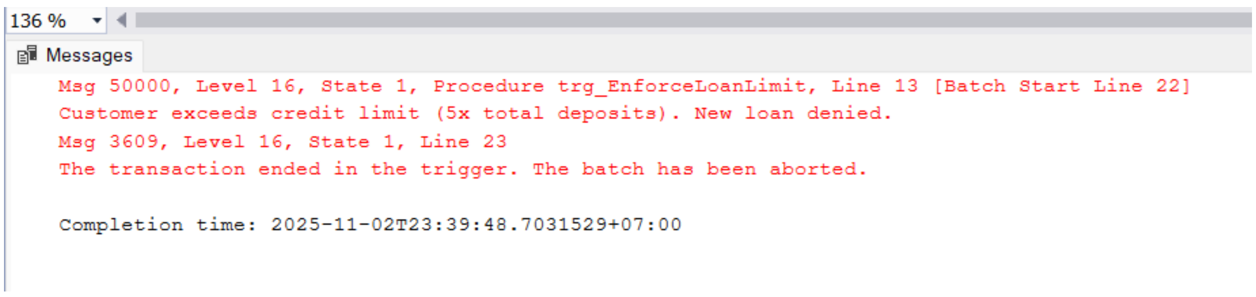
5.3. Trigger 3: trg_EnforceLoanLimit

- **Description:** Enforces a business rule. Prevents a new LOAN insertion if the customer's total loan amount would exceed 5 times their total deposit balance.

```
CREATE TRIGGER trg_EnforceLoanLimit
ON LOAN
INSTEAD OF INSERT
AS
BEGIN
   DECLARE @CustomerID INT, @NewLoanAmount DECIMAL(18, 2);
   DECLARE @TotalBalance DECIMAL(18, 2), @TotalLoans DECIMAL(18, 2);
   SELECT @CustomerID = i.CustomerID, @NewLoanAmount = i.LoanAmount FROM
inserted i;
   SELECT @TotalBalance = ISNULL(SUM(Balance), 0) FROM ACCOUNT WHERE
CustomerID = @CustomerID;
   SELECT @TotalLoans = ISNULL(SUM(LoanAmount), 0) FROM LOAN WHERE
CustomerID = @CustomerID;
   IF (@TotalLoans + @NewLoanAmount) > (@TotalBalance * 5)
   BEGIN
      RAISERROR('Customer exceeds credit limit (5x total deposits). New loan denied.', 16, 1);
      ROLLBACK TRANSACTION;
   END
   ELSE
   BEGIN
      INSERT INTO LOAN (LoanID, LoanType, LoanAmount, Status, StartDate, CustomerID,
BranchID)
      SELECT LoanID, LoanType, LoanAmount, Status, StartDate, CustomerID, BranchID
FROM inserted;
   END
END;
```

- Execution: Fails with error: Customer exceeds credit limit...

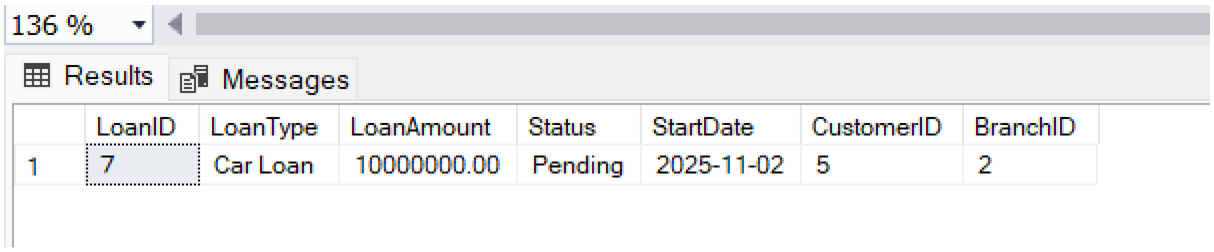
```
INSERT INTO LOAN (LoanID, LoanType, LoanAmount, Status, StartDate, CustomerID,
BranchID)
VALUES (6, 'Personal Loan', 10000000, 'Pending', GETDATE(), 1, 1);
```



```
Msg 50000, Level 16, State 1, Procedure trg_EnforceLoanLimit, Line 13 [Batch Start Line 22]
Customer exceeds credit limit (5x total deposits). New loan denied.
Msg 3609, Level 16, State 1, Line 23
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2025-11-02T23:39:48.7031529+07:00
```

- Execution: The INSERT succeeds and the new loan (LoanID = 7) is added.

```
INSERT INTO LOAN (LoanID, LoanType, LoanAmount, Status, StartDate, CustomerID,
BranchID)
VALUES (7, 'Car Loan', 10000000, 'Pending', GETDATE(), 5, 2);
SELECT * FROM LOAN WHERE LoanID = 7;
```


136 %
Results | Messages

	LoanID	LoanType	LoanAmount	Status	StartDate	CustomerID	BranchID
1	7	Car Loan	10000000.00	Pending	2025-11-02	5	2

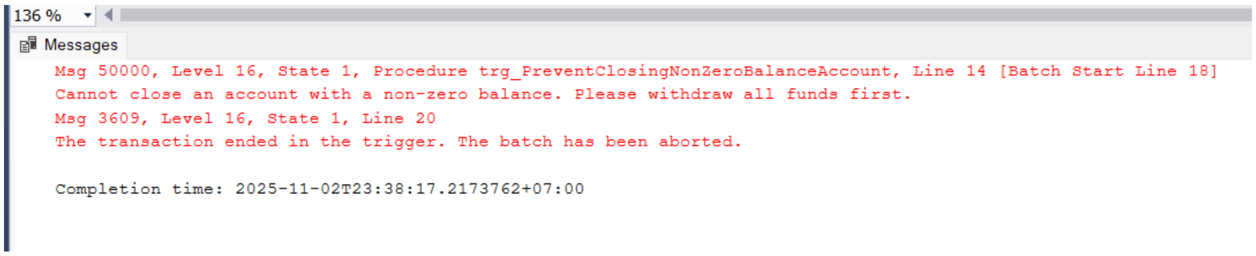
5.4. Trigger 4: trg_PreventClosingNonZeroBalanceAccount

- **Description:** This is the fourth required trigger. It enforces a business rule that prevents an account's Status from being changed to 'Closed' if its Balance is not 0.

```
CREATE TRIGGER trg_PreventClosingNonZeroBalanceAccount
ON ACCOUNT
AFTER UPDATE
AS
BEGIN
   -- Check if an update tried to set Status to 'Closed' on an account with money
   IF EXISTS (
      SELECT 1
      FROM inserted i
      JOIN deleted d ON i.AccountID = d.AccountID
      WHERE i.Status = 'Closed' AND d.Status <> 'Closed' AND i.Balance <> 0
   )
   BEGIN
      RAISERROR('Cannot close an account with a non-zero balance. Please withdraw all funds
first.', 16, 1);
      ROLLBACK TRANSACTION;
   END
END;
```

- Execution: The command fails and returns the error: Cannot close an account with a non-zero balance...

```
-- AccountID = 1 has a balance of 5,000,000.
UPDATE ACCOUNT SET Status = 'Closed' WHERE AccountID = 1;
```



```
136 %
Messages
Msg 50000, Level 16, State 1, Procedure trg_PreventClosingNonZeroBalanceAccount, Line 14 [Batch Start Line 18]
Cannot close an account with a non-zero balance. Please withdraw all funds first.
Msg 3609, Level 16, State 1, Line 20
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2025-11-02T23:38:17.2173762+07:00
```

6. View and Index

6.1. Views

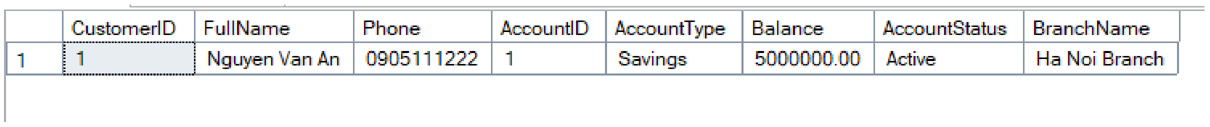
6.1.1. View 1: vw_CustomerAccountDetails

- **Description:** Defines a view to simplify complex queries. This view provides a straightforward way to see customer account details without needing to write the JOINs manually.

```
CREATE VIEW vw_CustomerAccountDetails AS
SELECT
   C.CustomerID, C.FullName, C.Phone,
   A.AccountID, A.AccountType, A.Balance, A.Status AS AccountStatus,
   B.BranchName
FROM CUSTOMER AS C
JOIN ACCOUNT AS A ON C.CustomerID = A.CustomerID
JOIN BRANCH AS B ON A.BranchID = B.BranchID;
```

- Execution:

```
SELECT * FROM vw_CustomerAccountDetails WHERE FullName LIKE N'Nguyen Van A%';
```



	CustomerID	FullName	Phone	AccountID	AccountType	Balance	AccountStatus	BranchName
1	1	Nguyen Van An	0905111222	1	Savings	5000000.00	Active	Ha Noi Branch

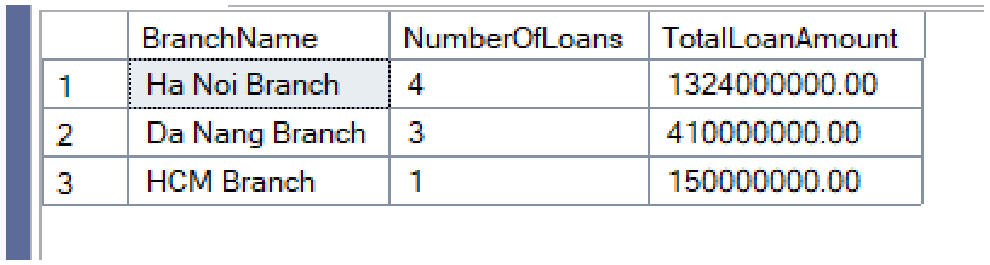
6.1.2. View 2: vw_BranchLoanSummary

- **Description:** This view summarizes the total loan amount and the count of loans for each branch. It is useful for management reports to assess the performance of each branch.

```
CREATE VIEW vw_BranchLoanSummary AS
SELECT B.BranchName,
   COUNT(L.LoanID) AS NumberOfLoans,
   SUM(L.LoanAmount) AS TotalLoanAmount
FROM BRANCH AS B
LEFT JOIN LOAN AS L ON B.BranchID = L.BranchID
GROUP BY B.BranchName;
```

- Execution:

```
SELECT * FROM vw_BranchLoanSummary;
```



	BranchName	NumberOfLoans	TotalLoanAmount
1	Ha Noi Branch	4	1324000000.00
2	Da Nang Branch	3	410000000.00
3	HCM Branch	1	150000000.00

6.2. Views

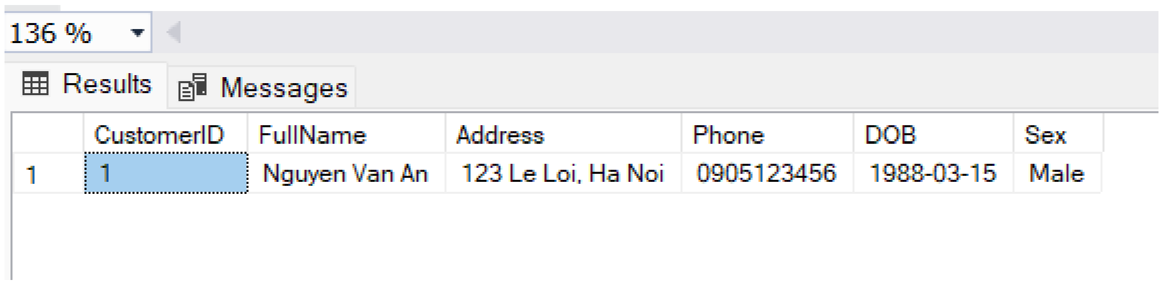
6.2.1. Index 1: idx_Customer_Phone (Single Column)

- **Description:** Creates a **UNIQUE**, non-clustered, single-column index on the **Phone** column of the **CUSTOMER** table. Since a phone number is unique and often used for customer lookups, this index will significantly improve the performance of queries that filter by phone number.

```
CREATE UNIQUE INDEX idx_Customer_Phone
ON CUSTOMER(Phone);
```

- Execution:

```
SELECT * FROM CUSTOMER WHERE Phone = '0905123456';
```



	CustomerID	FullName	Address	Phone	DOB	Sex
1	1	Nguyen Van An	123 Le Loi, Ha Noi	0905123456	1988-03-15	Male

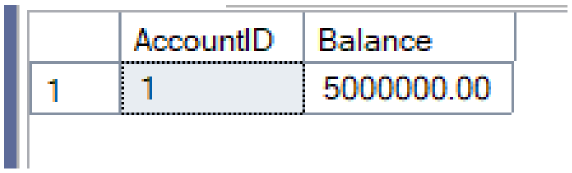
6.2.2. Index 2: idx_Account_CustomerBranch (Composite)

- **Description:** Creates a composite index on the **CustomerID** and **BranchID** columns in the **ACCOUNT** table. This index is designed to improve the performance of queries that frequently filter or join on both the customer and their branch.

```
CREATE INDEX idx_Account_CustomerBranch
ON ACCOUNT(CustomerID, BranchID);
```

- Execution:

```
SELECT AccountID, Balance FROM ACCOUNT
WHERE CustomerID = 1 AND BranchID = 1;
```



	AccountID	Balance
1	1	5000000.00

7. AI Utilization

During the development of Lab 5: Banking System, our group used AI tools to improve code quality, accuracy, and efficiency through Good Prompts, Debugging, and Refinement.

Good Prompts: AI suggested optimized SQL syntax for functions like fn_CalculateCustomerAge and helped structure multi-step logic in sp_PerformTransaction, ensuring clarity and efficiency.

Debugging: When testing sp_PerformTransaction, AI helped locate and fix a logic error in the balance validation condition IF (@CurrentBalance + @Amount) < 0, improving transaction accuracy.

Refinement: AI improved trigger efficiency, such as recommending IF UPDATE(Phone) with i.Phone <> d.Phone in trg_LogCustomerPhoneChange, so it runs only when the phone number changes.

AI served as a support tool, not a generator. All outputs were reviewed, tested, and aligned with our database's business logic.

8. Conclusion and Reflection

Conclusion: Lab 5 enabled our group to apply advanced SQL concepts to the Banking System project. We completed:

- SQL queries (Basic → Intermediate → Advanced)
- 4 functions, 4 procedures, 4 triggers

- 2 views and 2 indexes

These elements simplify data handling, automate operations, and enforce data integrity.

Reflection (Computational Thinking & AI Use):

- *Decomposition*: We broke down complex tasks (e.g., deposit handling) into smaller parts in sp_PerformTransaction.
- *Abstraction*: Views (vw_CustomerAccountDetails) and functions (fn_GetCustomerTotalBalance) simplified complex joins for end users.
- *Pattern Recognition*: We reused logic patterns such as preventing parent deletion (trg_PreventBranchDeletion) and indexing frequent searches (Customer.Phone).
- *Algorithmic Thinking*: Procedures and triggers followed structured steps—validate, update, handle errors—ensuring reliable logic.
- *AI Assistance*: AI supported debugging and optimization. All suggestions were verified to maintain accuracy and originality.

Final Reflection: This lab enhanced our SQL design and teamwork skills. Combining computational thinking with AI refinement helped us build a more effective and reliable database system.