**Exercise 1: Control Structures(Mandatory)**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

Solution:

First create an tables required for the pl/sql

-- Create Customers Table

CREATE TABLE Customers (

CustomerID NUMBER,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5),

LoanInterest NUMBER

);

-- Create Loans Table

CREATE TABLE Loans (

LoanID NUMBER,

CustomerID NUMBER,

DueDate DATE

);

-- Sample Customers

INSERT INTO Customers VALUES (1, 'John', 65, 15000, 'FALSE', 5.5);

INSERT INTO Customers VALUES (2, 'Alice', 45, 8000, 'FALSE', 6.0);

INSERT INTO Customers VALUES (3, 'Bob', 70, 20000, 'FALSE', 7.0);

INSERT INTO Loans VALUES (101, 1, DATE\_ADD(CURDATE(), INTERVAL 10 DAY));

INSERT INTO Loans VALUES (102, 2, DATE\_ADD(CURDATE(), INTERVAL 40 DAY));

INSERT INTO Loans VALUES (103, 3, DATE\_ADD(CURDATE(), INTERVAL 20 DAY));

**Scenario 1:**

DELIMITER //

CREATE PROCEDURE ApplyLoanDiscount()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE vCustomerID INT;

DECLARE vAge INT;

DECLARE vLoanInterest DECIMAL(5,2);

DECLARE cur CURSOR FOR SELECT CustomerID, Age, LoanInterest FROM Customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO vCustomerID, vAge, vLoanInterest;

IF done THEN

LEAVE read\_loop;

END IF;

IF vAge > 60 THEN

UPDATE Customers

SET LoanInterest = LoanInterest - 1

WHERE CustomerID = vCustomerID;

END IF;

END LOOP;

CLOSE cur;

END;

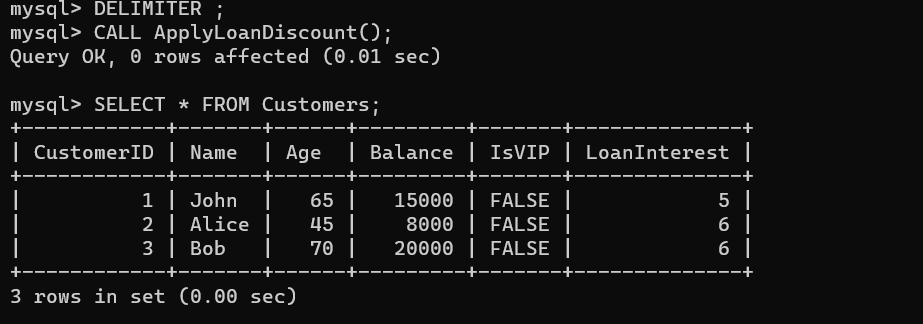
//

DELIMITER ;

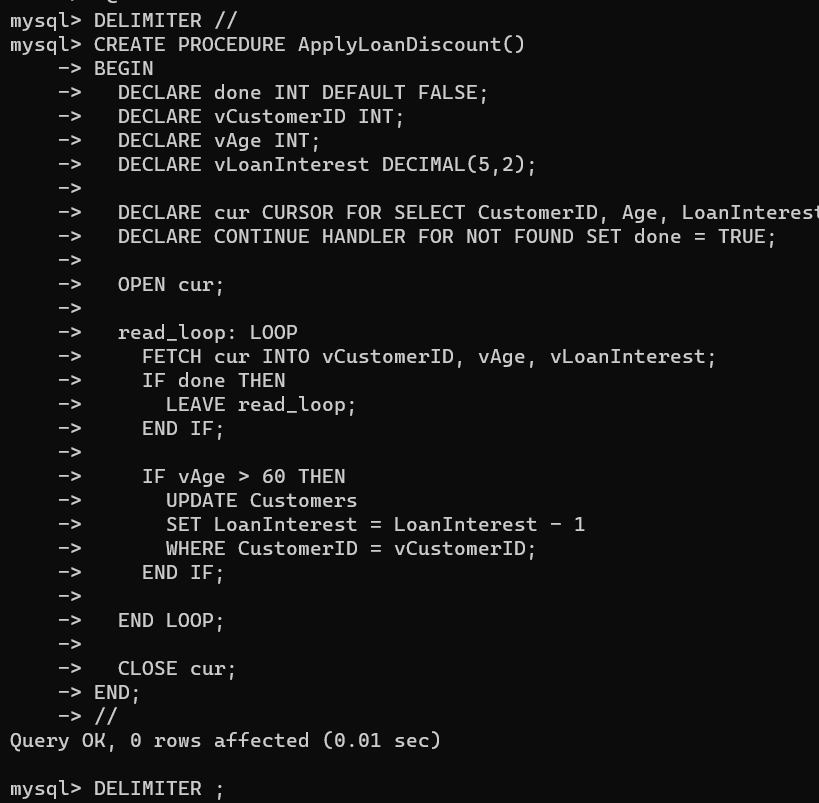
CALL ApplyLoanDiscount();

SELECT \* FROM Customers;

Output:



Sql code:



**Scenario 2:**

CREATE PROCEDURE PromoteVIPCustomers()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE vCustomerID INT;

DECLARE vBalance DECIMAL(10,2);

DECLARE cur CURSOR FOR SELECT CustomerID, Balance FROM Customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO vCustomerID, vBalance;

IF done THEN

LEAVE read\_loop;

END IF;

IF vBalance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = vCustomerID;

END IF;

END LOOP;

CLOSE cur;

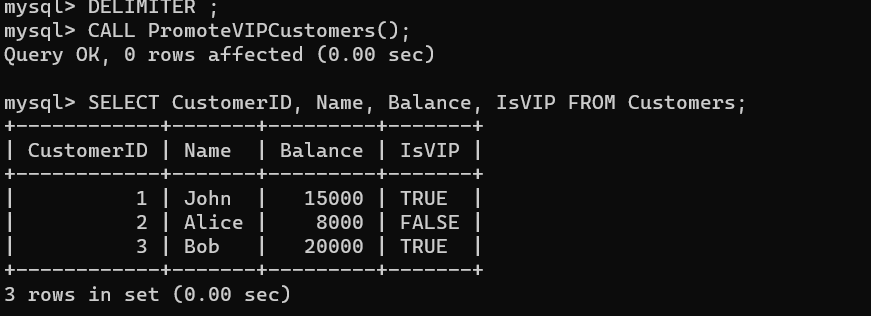
END;

//

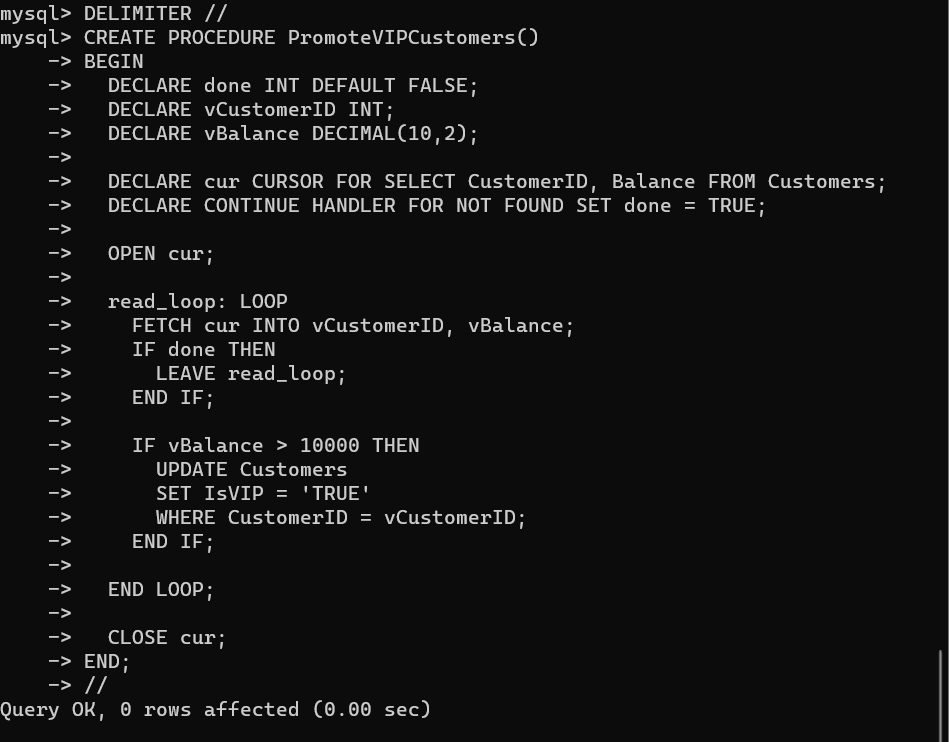
CALL PromoteVIPCustomers();

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

Output:



Sql code:



**Scenario 3:**

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE vLoanID INT;

DECLARE vCustomerName VARCHAR(100);

DECLARE vDueDate DATE;

DECLARE cur CURSOR FOR

SELECT l.LoanID, c.Name, l.DueDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY);

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO vLoanID, vCustomerName, vDueDate;

IF done THEN

LEAVE read\_loop;

END IF;

-- Print reminder message

SELECT CONCAT('Reminder: ', vCustomerName, ', your loan (ID: ', vLoanID, ') is due on ', DATE\_FORMAT(vDueDate, '%d-%b-%Y')) AS ReminderMessage;

END LOOP;

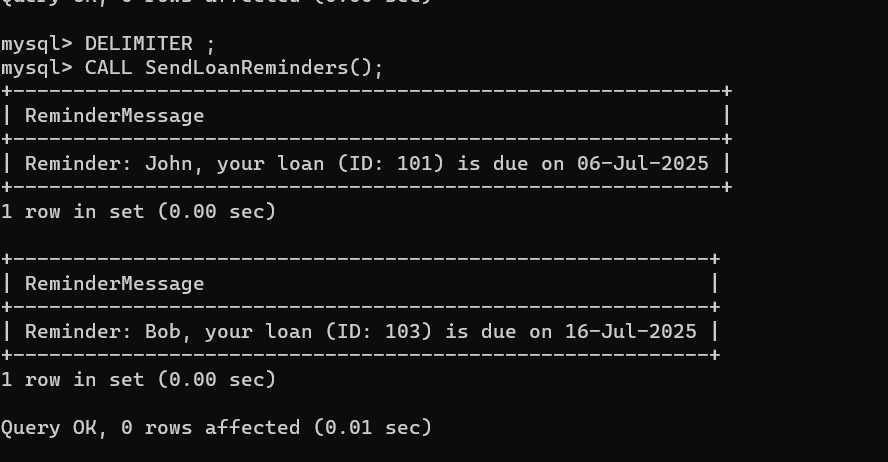
CLOSE cur;

END;

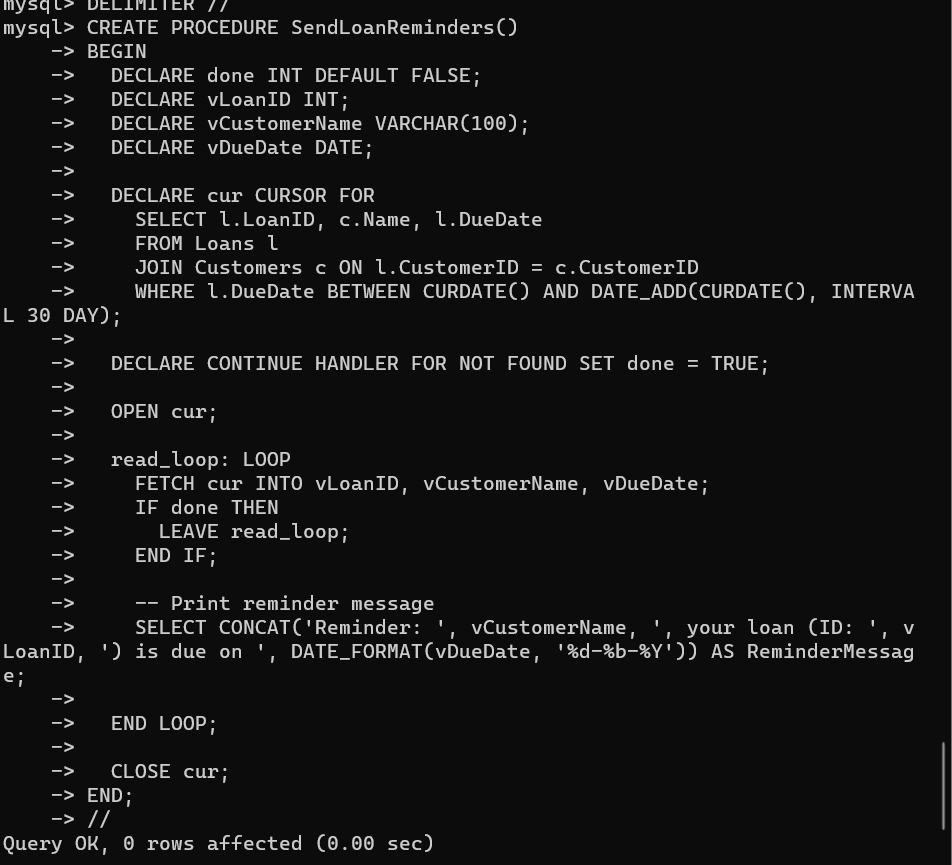
//

CALL SendLoanReminders();

Outputs:



Sql code:



**Exercise 3: Stored Procedures(Mandatory)**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

Solution:

Tables creation:

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

AccountType VARCHAR(20),

Balance DECIMAL(10,2)

);

CREATE TABLE Employees (

EmpID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(50),

Salary DECIMAL(10,2)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

AccountType VARCHAR(20),

Balance DECIMAL(10,2)

);

Insert data into tables:

INSERT INTO Accounts (AccountID, AccountType, Balance) VALUES

(1001, 'savings', 12000.50),

(1002, 'savings', 4500.75),

(1003, 'current', 9800.00),

(1004, 'savings', 15320.00),

(1005, 'current', 6000.00),

(1006, 'savings', 8750.30);

INSERT INTO Employees (EmpID, Name, Department, Salary) VALUES

(2001, 'Esha Nair', 'Sales', 58000.00),

(2002, 'Kabir Anand', 'Marketing', 63000.00),

(2003, 'Ritika Bose', 'Sales', 54000.00),

(2004, 'Yusuf Khan', 'Finance', 72000.00),

(2005, 'Ananya Reddy', 'HR', 51000.00),

(2006, 'Sahil Mehra', 'Marketing', 60500.00);

**Scenario 1:**

DELIMITER //

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'savings';

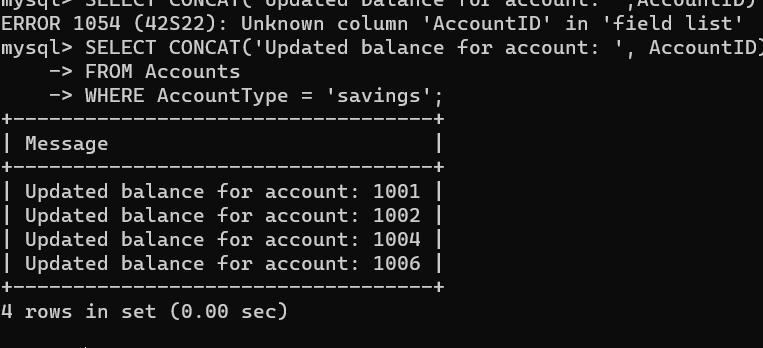
END;

//

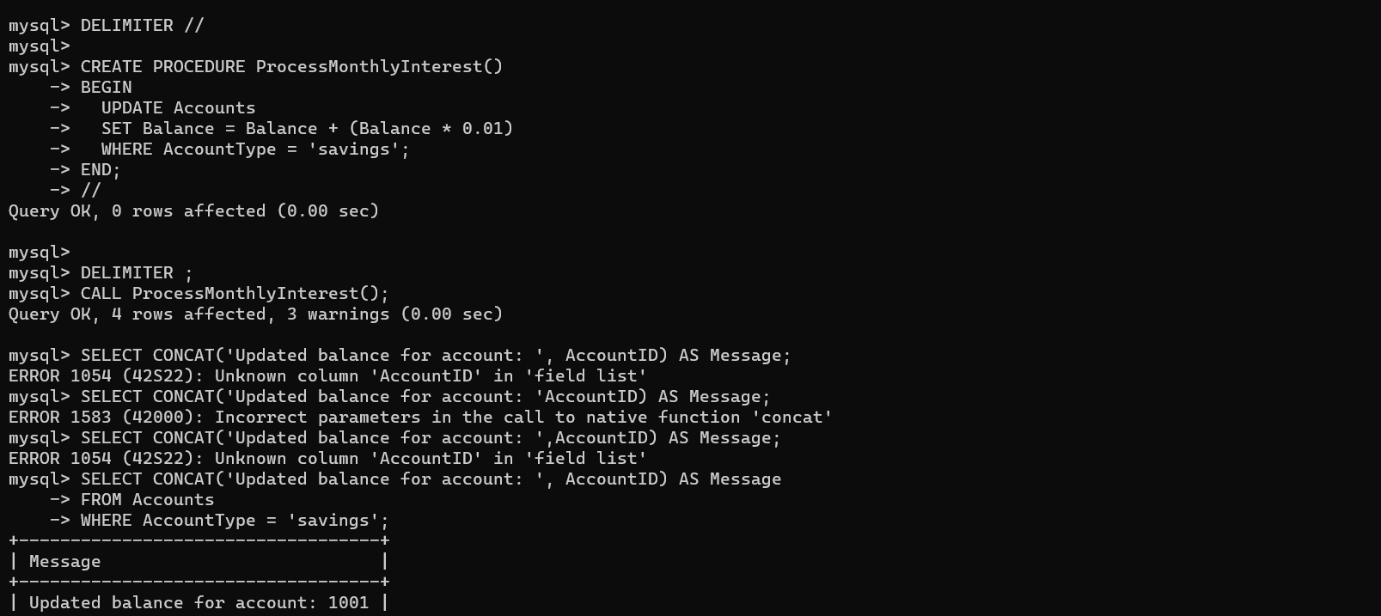
DELIMITER ;

CALL ProcessMonthlyInterest();

Output:



Sql code:



**Scenario 2:**

DELIMITER //

CREATE PROCEDURE UpdateEmployeeBonus(

IN deptName VARCHAR(50),

IN bonusPercent DECIMAL(5,2)

)

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* (bonusPercent / 100))

WHERE Department = deptName;

-- Print updated employees

SELECT CONCAT(Name, ' got a bonus. New salary: ', Salary) AS Message

FROM Employees

WHERE Department = deptName;

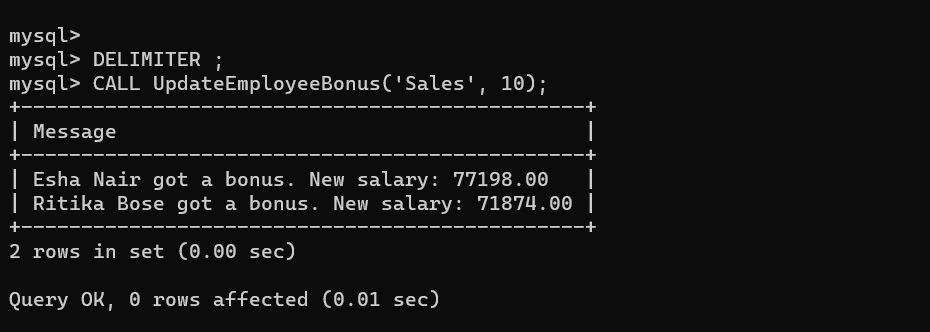
END;

//

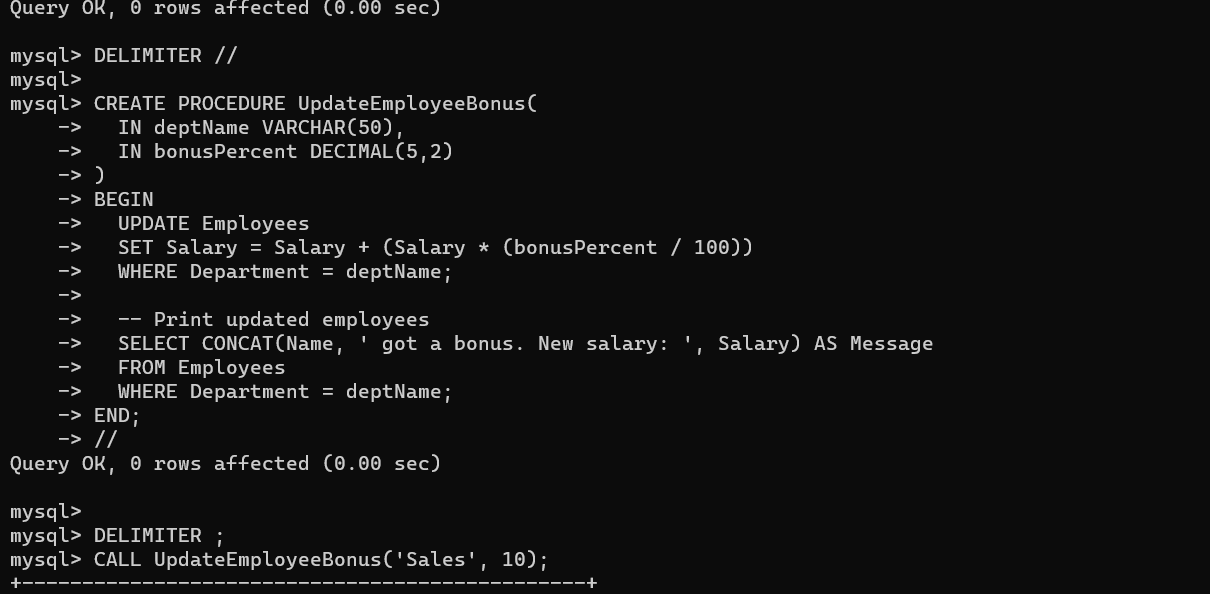
DELIMITER ;

CALL UpdateEmployeeBonus('Sales', 10);

Output:



Sql code:



**Scenario 3:**

DELIMITER //

CREATE PROCEDURE TransferFunds(

IN fromAcc INT,

IN toAcc INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE fromBalance DECIMAL(10,2);

-- Get balance of source account

SELECT Balance INTO fromBalance

FROM Accounts

WHERE AccountID = fromAcc;

-- Check if sufficient balance

IF fromBalance >= amount THEN

-- Deduct amount from source

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = fromAcc;

-- Add amount to destination

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = toAcc;

-- Print transfer confirmation

SELECT CONCAT('Transferred ₹', amount, ' from account ', fromAcc,

' to account ', toAcc) AS Message;

ELSE

-- Raise error if insufficient balance

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient funds';

END IF;

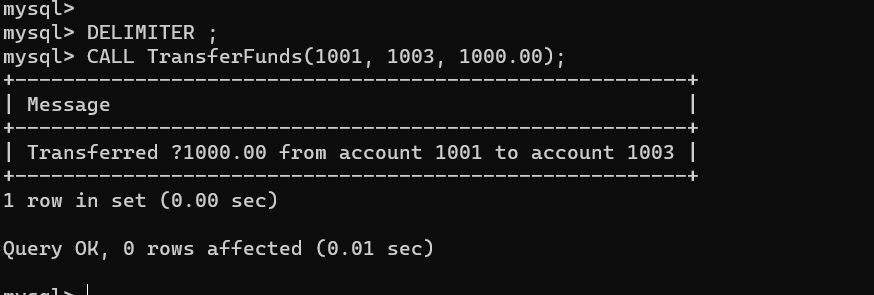
END;

//

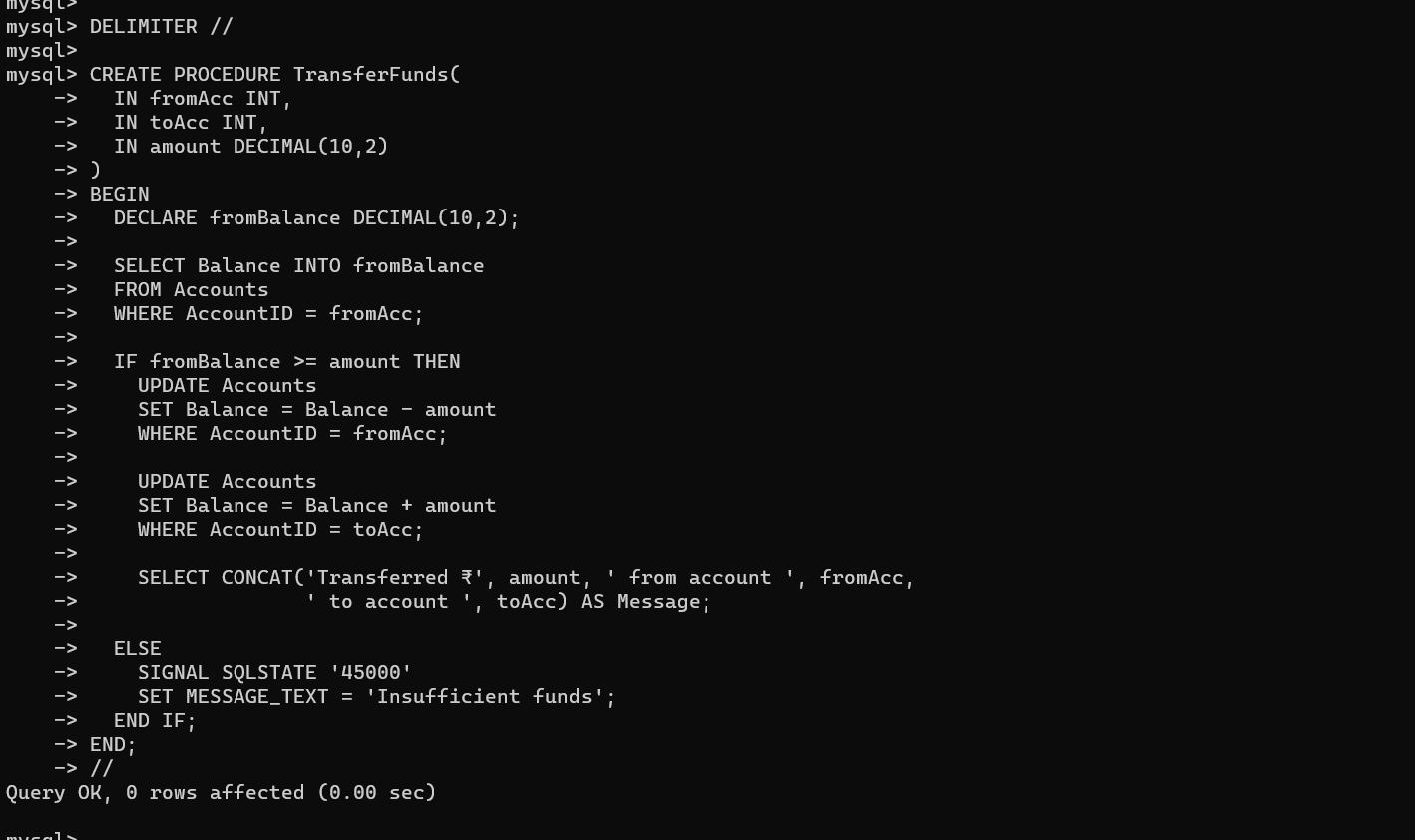
DELIMITER ;

CALL TransferFunds(1001, 1003, 1000.00);

Output:



Sql code:



**Exercise 2: Error Handling(Additional handsOn)**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**Data required for the program**

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

AccountHolder VARCHAR(100),

Balance DECIMAL(10, 2)

);

CREATE TABLE TransferErrors (

ErrorID INT AUTO\_INCREMENT PRIMARY KEY,

ErrorTime DATETIME DEFAULT CURRENT\_TIMESTAMP,

ErrorMessage VARCHAR(255)

);

-- Insert sample accounts

INSERT INTO Accounts (AccountID, AccountHolder, Balance)

VALUES

(101, 'Alice', 1000.00),

(202, 'Bob', 500.00),

(303, 'Charlie', 1500.00);

-- Insert sample employees

INSERT INTO Employees (EmpID, EmpName, Salary)

VALUES

(1, 'John Doe', 50000.00),

(2, 'Jane Smith', 60000.00),

(3, 'Priya Kapoor', 45000.00);

-- Insert sample customers

INSERT INTO Customers (CustomerID, CustomerName, City)

VALUES

(1001, 'Tom Hanks', 'Chennai'),

(1002, 'Jerry Seinfeld', 'Mumbai');

**Scenario 1:**

DELIMITER //

DROP PROCEDURE IF EXISTS SafeTransferFunds;

CREATE PROCEDURE SafeTransferFunds(

IN senderID INT,

IN receiverID INT,

IN amount DECIMAL(10, 2)

)

BEGIN

DECLARE senderBalance DECIMAL(10,2) DEFAULT 0.0;

DECLARE errMsg VARCHAR(255);

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

SET errMsg = CONCAT('Transfer failed from Account ', senderID, ' to ', receiverID);

INSERT INTO TransferErrors(ErrorMessage) VALUES (errMsg);

ROLLBACK;

END;

START TRANSACTION;

-- Get balance of sender

SELECT Balance INTO senderBalance FROM Accounts WHERE AccountID = senderID FOR UPDATE;

-- Show balance for debug

SELECT senderBalance AS 'Fetched Balance';

-- Check for insufficient funds

IF senderBalance < amount THEN

-- Manual rollback before SIGNAL

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient balance for transfer.';

END IF;

-- Transfer money

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = senderID;

UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = receiverID;

COMMIT;

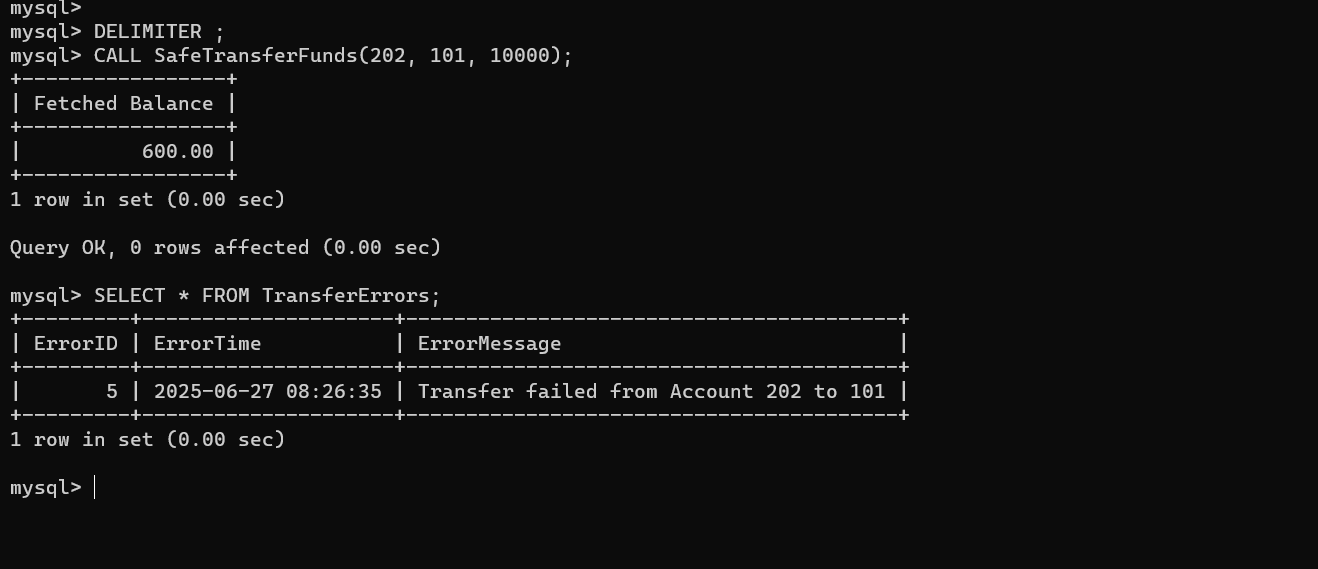
END;

//

DELIMITER ;

CALL SafeTransferFunds(202, 101, 10000);

Output:



**Scenario 2:**

CREATE TABLE Employees (

EmpID INT PRIMARY KEY,

EmpName VARCHAR(100),

Salary DECIMAL(10, 2)

);

CREATE TABLE SalaryUpdateErrors (

ErrorID INT AUTO\_INCREMENT PRIMARY KEY,

ErrorTime DATETIME DEFAULT CURRENT\_TIMESTAMP,

ErrorDetail VARCHAR(255)

);

INSERT INTO Employees (EmpID, EmpName, Salary)

VALUES

(1, 'John Doe', 50000.00),

(2, 'Anjali Sharma', 60000.00),

(3, 'Michael Lee', 70000.00);

DELIMITER //

DROP PROCEDURE IF EXISTS UpdateSalary;

CREATE PROCEDURE UpdateSalary(

IN empID INT,

IN percentIncrease DECIMAL(5,2)

)

BEGIN

DECLARE currentSalary DECIMAL(10,2);

DECLARE errorDetail VARCHAR(255);

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

SET errorDetail = CONCAT('Salary update failed for Employee ID ', empID);

INSERT INTO SalaryUpdateErrors(ErrorDetail) VALUES (errorDetail);

END;

-- Check if employee exists and fetch salary

SELECT Salary INTO currentSalary FROM Employees WHERE EmpID = empID;

-- Optional Debug

SELECT currentSalary AS 'Current Salary';

-- Update salary

UPDATE Employees

SET Salary = Salary + (Salary \* percentIncrease / 100)

WHERE EmpID = empID;

END;

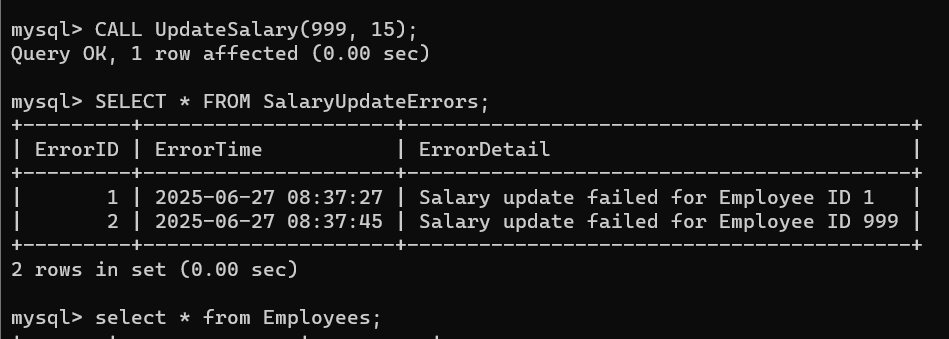
//

DELIMITER ;

CALL UpdateSalary(1, 10);

CALL UpdateSalary(999, 15); -- Should trigger error

Output:



**Scenario 3:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

City VARCHAR(100)

);

CREATE TABLE CustomerInsertErrors (

LogID INT AUTO\_INCREMENT PRIMARY KEY,

LogTime DATETIME DEFAULT CURRENT\_TIMESTAMP,

LogMessage VARCHAR(255)

);

INSERT INTO Customers (CustomerID, CustomerName, City)

VALUES

(1001, 'Amit Mehra', 'Mumbai'),

(1002, 'Sara Ali', 'Delhi'),

(1003, 'Vikram Rao', 'Chennai');

DELIMITER //

DROP PROCEDURE IF EXISTS AddNewCustomer;

CREATE PROCEDURE AddNewCustomer(

IN newID INT,

IN newName VARCHAR(100),

IN newCity VARCHAR(100)

)

BEGIN

DECLARE existingCount INT DEFAULT 0;

DECLARE logMsg VARCHAR(255);

-- Check if customer already exists

SELECT COUNT(\*) INTO existingCount FROM Customers WHERE CustomerID = newID;

IF existingCount > 0 THEN

SET logMsg = CONCAT('Insert failed: Customer ID ', newID, ' already exists.');

INSERT INTO CustomerInsertErrors(LogMessage) VALUES (logMsg);

ELSE

INSERT INTO Customers(CustomerID, CustomerName, City)

VALUES (newID, newName, newCity);

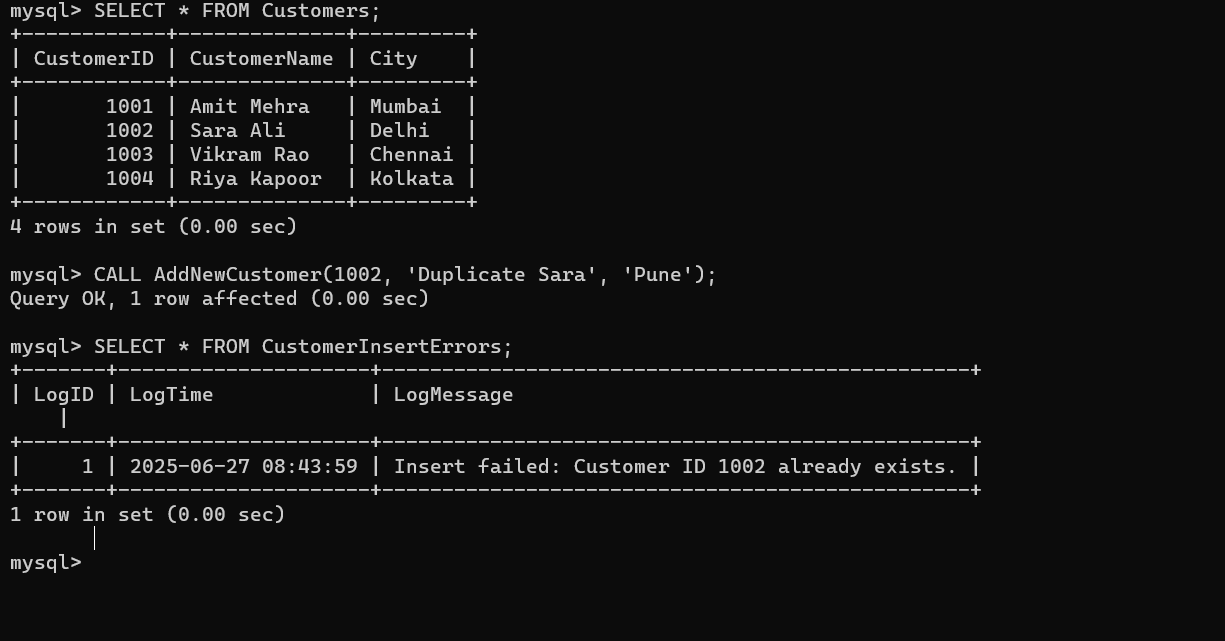
END IF;

END;

//

DELIMITER ;

Output:



**Exercise 4: Functions(additional handsOn)**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

Required data:

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

DOB DATE

);

INSERT INTO Customers (CustomerID, Name, DOB) VALUES

(1, 'John Doe', '1990-05-15'),

(2, 'Jane Smith', '1985-10-22'),

(3, 'Alice Johnson', '2000-01-10');

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount DECIMAL(10,2),

InterestRate DECIMAL(5,2),

DurationYears INT

);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, DurationYears) VALUES

(1, 1, 500000.00, 7.5, 5),

(2, 2, 300000.00, 8.0, 3),

(3, 3, 200000.00, 6.5, 4);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

Balance DECIMAL(10,2)

);

INSERT INTO Accounts (AccountID, CustomerID, Balance) VALUES

(101, 1, 15000.00),

(102, 2, 3000.00),

(103, 3, 7000.00);

**Scenario 1:**

DELIMITER //

CREATE FUNCTION CalculateAge(dob DATE)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE age INT;

SET age = TIMESTAMPDIFF(YEAR, dob, CURDATE());

RETURN age;

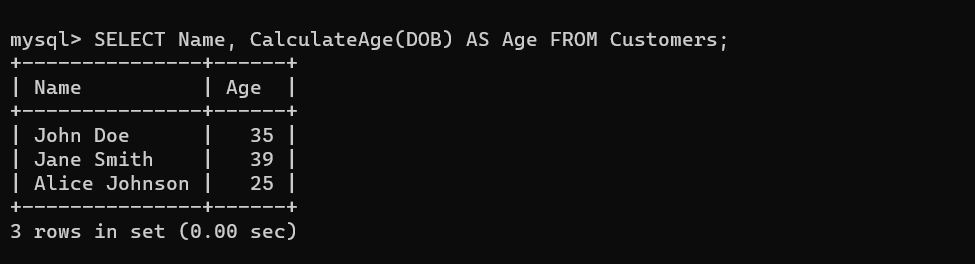
END;

//

DELIMITER ;

SELECT Name, CalculateAge(DOB) AS Age FROM Customers;

Output:



**Scenario 2:**

DELIMITER //

CREATE FUNCTION CalculateMonthlyInstallment(

loanAmount DECIMAL(10,2),

annualInterestRate DECIMAL(5,2),

durationYears INT

)

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE r DECIMAL(10,6);

DECLARE n INT;

DECLARE emi DECIMAL(10,2);

SET r = annualInterestRate / 12 / 100;

SET n = durationYears \* 12;

IF r = 0 THEN

SET emi = loanAmount / n;

ELSE

SET emi = loanAmount \* r \* POWER((1 + r), n) / (POWER((1 + r), n) - 1);

END IF;

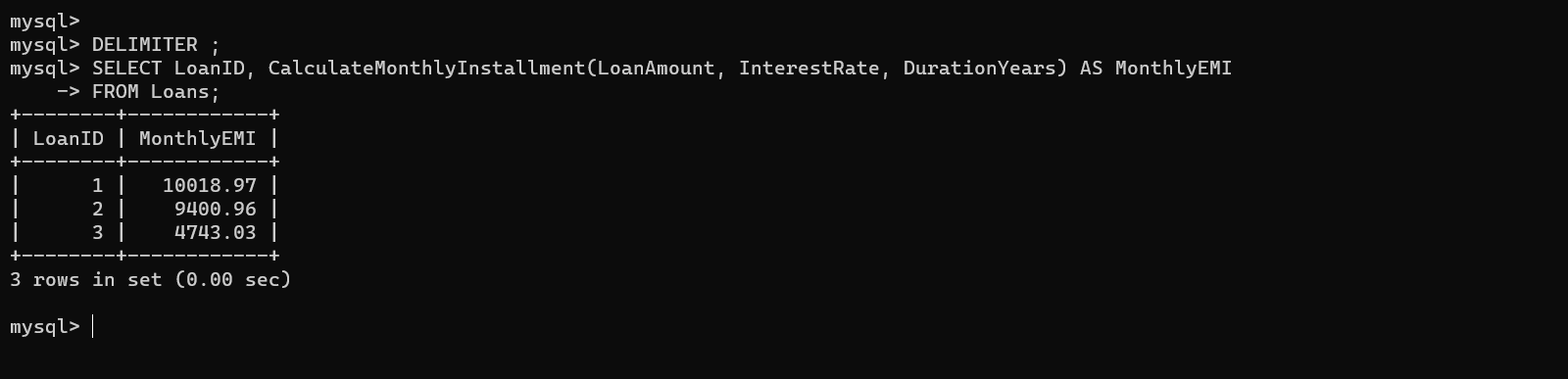
RETURN ROUND(emi, 2);

END;

//

DELIMITER ;

Output:



**Scenario 3:**

DELIMITER //

CREATE FUNCTION HasSufficientBalance(accID INT, requiredAmount DECIMAL(10,2))

RETURNS BOOLEAN

DETERMINISTIC

BEGIN

DECLARE currentBalance DECIMAL(10,2);

SELECT Balance INTO currentBalance

FROM Accounts

WHERE AccountID = accID;

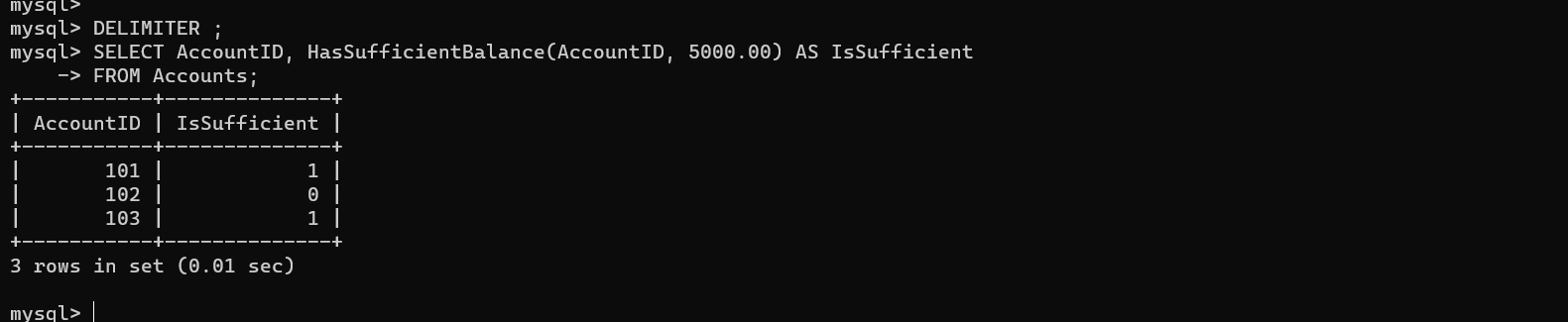
RETURN currentBalance >= requiredAmount;

END;

//

DELIMITER ;

Output:



**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

* + **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**Scenario 2:** Maintain an audit log for all transactions.

* + **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**Scenario 3:** Enforce business rules on deposits and withdrawals.

* + **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

**Scenario 1:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

DOB DATE,

LastModified DATE

);

INSERT INTO Customers (CustomerID, Name, DOB, LastModified) VALUES

(1, 'John Doe', '1990-05-15', '2024-01-01'),

(2, 'Jane Smith', '1985-10-22', '2024-01-01');

DELIMITER //

CREATE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

SET NEW.LastModified = CURDATE();

END;

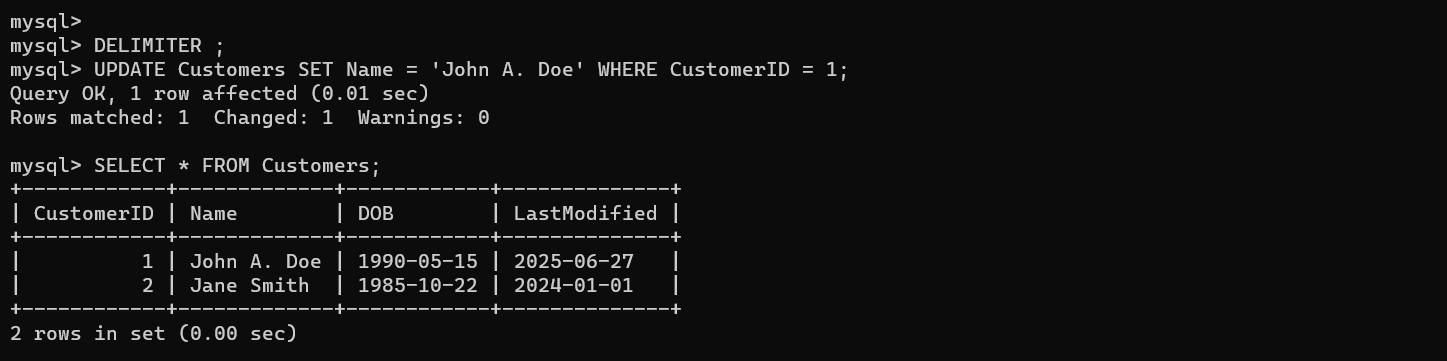
//

DELIMITER ;

UPDATE Customers SET Name = 'John A. Doe' WHERE CustomerID = 1;

SELECT \* FROM Customers;

Output:



**Scenario 2:**

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

Type ENUM('Deposit', 'Withdrawal'),

Amount DECIMAL(10,2),

TransactionDate DATE

);

CREATE TABLE AuditLog (

LogID INT AUTO\_INCREMENT PRIMARY KEY,

TransactionID INT,

Action VARCHAR(50),

LoggedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER //

CREATE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, Action)

VALUES (NEW.TransactionID, CONCAT('Inserted ', NEW.Type));

END;

//

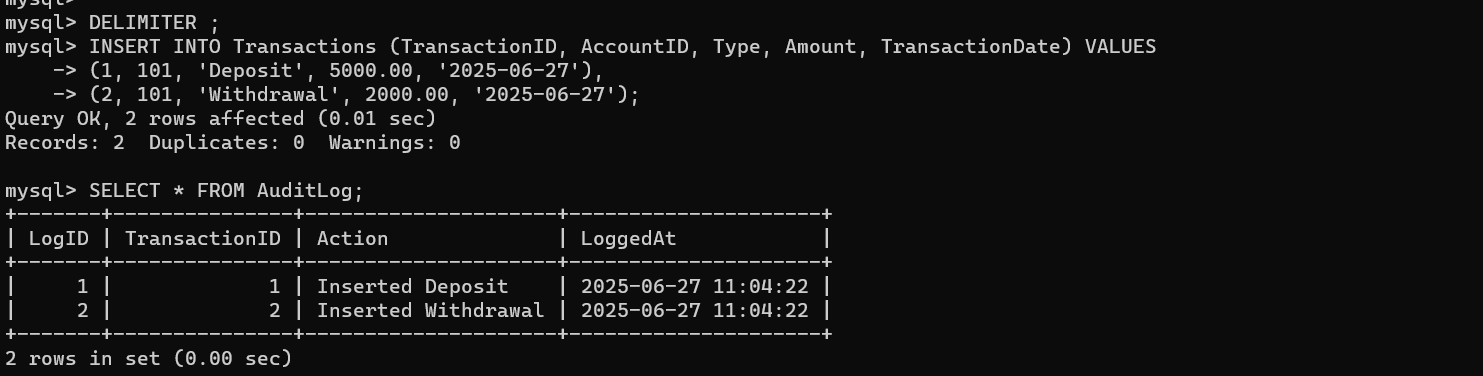
DELIMITER ;

INSERT INTO Transactions (TransactionID, AccountID, Type, Amount, TransactionDate) VALUES

(1, 101, 'Deposit', 5000.00, CURDATE()),

(2, 101, 'Withdrawal', 2000.00, CURDATE());

Output:



**Scenario 3:**

UPDATE Accounts SET Balance = 10000.00 WHERE AccountID = 101;

DELIMITER //

CREATE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

DECLARE currentBalance DECIMAL(10,2);

SELECT Balance INTO currentBalance

FROM Accounts

WHERE AccountID = NEW.AccountID;

IF NEW.Type = 'Withdrawal' AND NEW.Amount > currentBalance THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Withdrawal exceeds account balance';

END IF;

IF NEW.Type = 'Deposit' AND NEW.Amount <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Deposit amount must be positive';

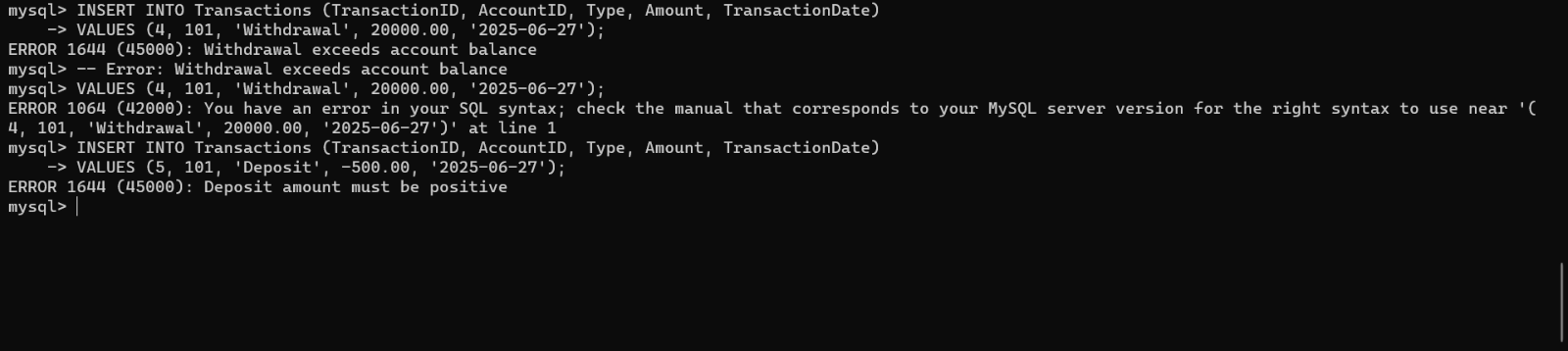
END IF;

END;

//

DELIMITER ;

Output:



**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

* + **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

**Scenario 2:** Apply annual fee to all accounts.

* + **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

**Scenario 3:** Update the interest rate for all loans based on a new policy.

* + **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

**Scenario 1:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

CustomerID INT,

Type ENUM('Deposit', 'Withdrawal'),

Amount DECIMAL(10,2),

TransactionDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe');

INSERT INTO Customers VALUES (2, 'Jane Smith');

INSERT INTO Transactions VALUES (101, 1, 'Deposit', 5000.00, CURDATE());

INSERT INTO Transactions VALUES (102, 1, 'Withdrawal', 2000.00, CURDATE());

INSERT INTO Transactions VALUES (103, 2, 'Deposit', 3000.00, CURDATE());

DELIMITER //

CREATE PROCEDURE GenerateMonthlyStatements()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_customer\_id INT;

DECLARE v\_name VARCHAR(100);

DECLARE v\_type ENUM('Deposit', 'Withdrawal');

DECLARE v\_amount DECIMAL(10,2);

DECLARE v\_date DATE;

DECLARE cur CURSOR FOR

SELECT c.CustomerID, c.Name, t.Type, t.Amount, t.TransactionDate

FROM Customers c

JOIN Transactions t ON c.CustomerID = t.CustomerID

WHERE MONTH(t.TransactionDate) = MONTH(CURDATE())

AND YEAR(t.TransactionDate) = YEAR(CURDATE());

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_customer\_id, v\_name, v\_type, v\_amount, v\_date;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT('Customer: ', v\_name, ' | ', v\_type, ' of Rs.', v\_amount, ' on ', v\_date) AS Statement;

END LOOP;

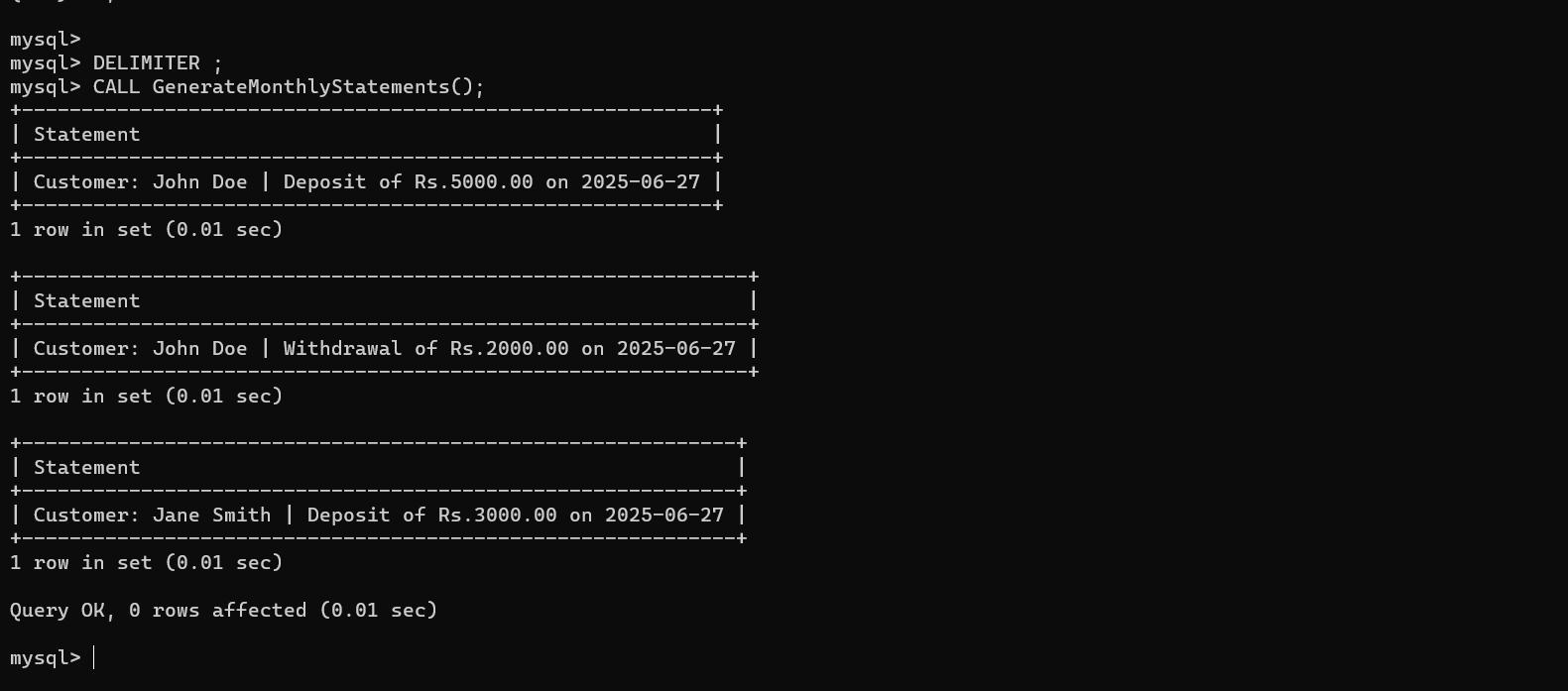
CLOSE cur;

END;

//

DELIMITER ;

Output:



**Scenario 2:**

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

Balance DECIMAL(10,2)

);

INSERT INTO Accounts (AccountID, CustomerID, Balance) VALUES

(201, 1, 10000.00),

(202, 2, 400.00),

(203, 3, 7000.00),

(204, 4, 500.00);

DELIMITER //

CREATE PROCEDURE ApplyAnnualFee()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_account\_id INT;

DECLARE v\_balance DECIMAL(10,2);

DECLARE fee DECIMAL(10,2) DEFAULT 500.00;

DECLARE cur CURSOR FOR

SELECT AccountID, Balance FROM Accounts;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_account\_id, v\_balance;

IF done THEN

LEAVE read\_loop;

END IF;

IF v\_balance >= fee THEN

UPDATE Accounts

SET Balance = Balance - fee

WHERE AccountID = v\_account\_id;

SELECT CONCAT('Annual fee applied to Account ', v\_account\_id) AS Message;

ELSE

SELECT CONCAT('Insufficient balance in Account ', v\_account\_id) AS Message;

END IF;

END LOOP;

CLOSE cur;

END;

//

DELIMITER ;

Output:



**Scenario 3:**

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

InterestRate DECIMAL(5,2)

);

INSERT INTO Loans (LoanID, CustomerID, InterestRate) VALUES

(301, 1, 6.5),

(302, 2, 7.5),

(303, 3, 5.75),

(304, 4, 8.0);

DELIMITER //

CREATE PROCEDURE UpdateLoanInterestRates()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_loan\_id INT;

DECLARE v\_rate DECIMAL(5,2);

DECLARE cur CURSOR FOR

SELECT LoanID, InterestRate FROM Loans;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_loan\_id, v\_rate;

IF done THEN

LEAVE read\_loop;

END IF;

IF v\_rate < 7 THEN

UPDATE Loans

SET InterestRate = v\_rate + 1

WHERE LoanID = v\_loan\_id;

SELECT CONCAT('Loan ', v\_loan\_id, ': Rate increased to ', v\_rate + 1) AS Result;

ELSE

UPDATE Loans

SET InterestRate = v\_rate + 0.5

WHERE LoanID = v\_loan\_id;

SELECT CONCAT('Loan ', v\_loan\_id, ': Rate increased to ', v\_rate + 0.5) AS Result;

END IF;

END LOOP;

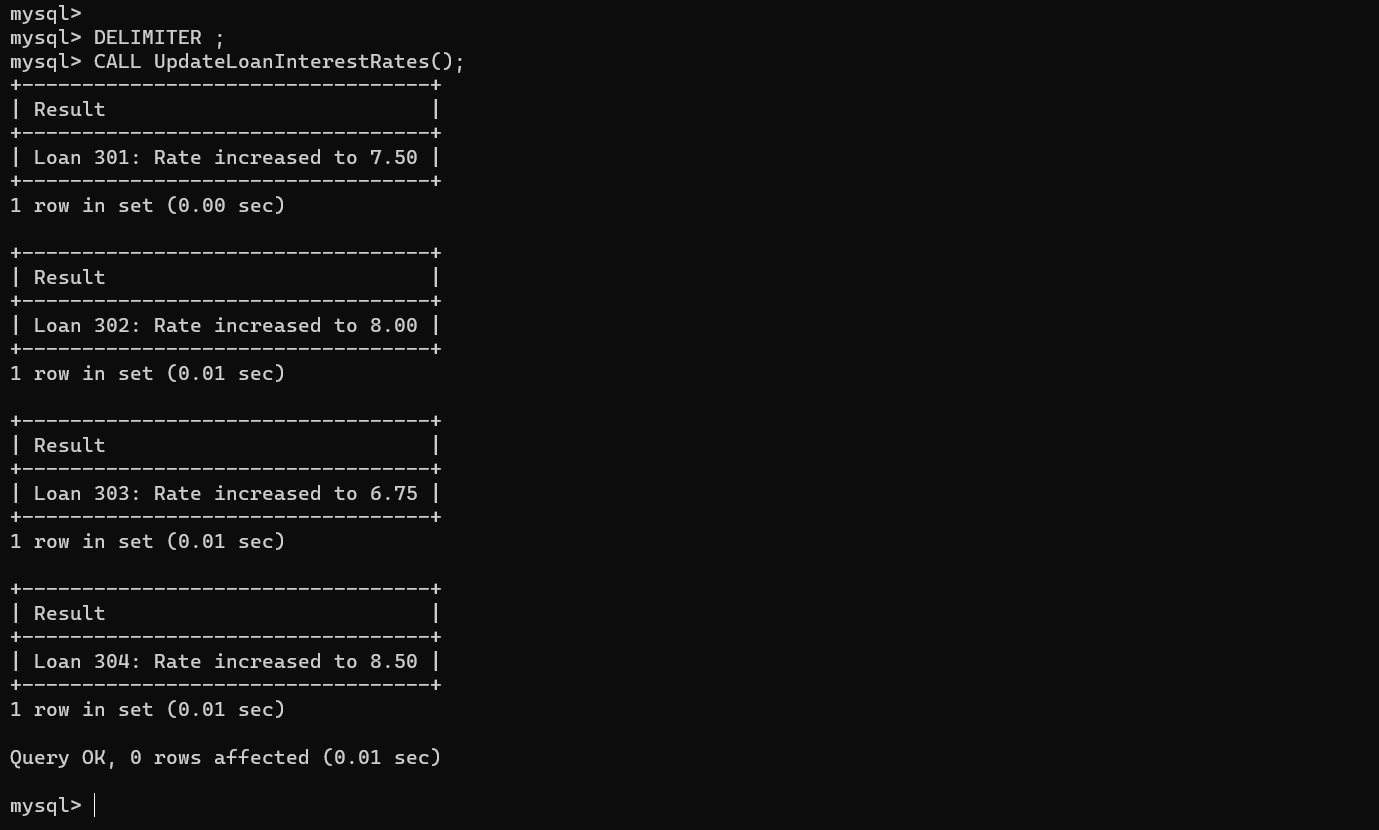
CLOSE cur;

END;

//

DELIMITER ;

Output:



**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

* + **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**Scenario 2:** Create a package to manage employee data.

* + **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Scenario 3:** Group all account-related operations into a package.

* + **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Scenario 1:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Email VARCHAR2(100)

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER(10,2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', 'john@example.com');

INSERT INTO Accounts VALUES (101, 1, 10000.00);

INSERT INTO Accounts VALUES (102, 1, 5000.00);

DELIMITER //

CREATE PROCEDURE Customer\_Add(

IN p\_id INT, IN p\_name VARCHAR(100), IN p\_email VARCHAR(100)

)

BEGIN

INSERT INTO Customers (CustomerID, Name, Email)

VALUES (p\_id, p\_name, p\_email);

END;

//

DELIMITER ;

DELIMITER //

CREATE PROCEDURE Customer\_Update(

IN p\_id INT, IN p\_name VARCHAR(100), IN p\_email VARCHAR(100)

)

BEGIN

UPDATE Customers SET Name = p\_name, Email = p\_email WHERE CustomerID = p\_id;

END;

//

DELIMITER ;

DELIMITER //

CREATE FUNCTION Customer\_GetBalance(p\_id INT)

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE total DECIMAL(10,2);

SELECT IFNULL(SUM(Balance), 0) INTO total FROM Accounts WHERE CustomerID = p\_id;

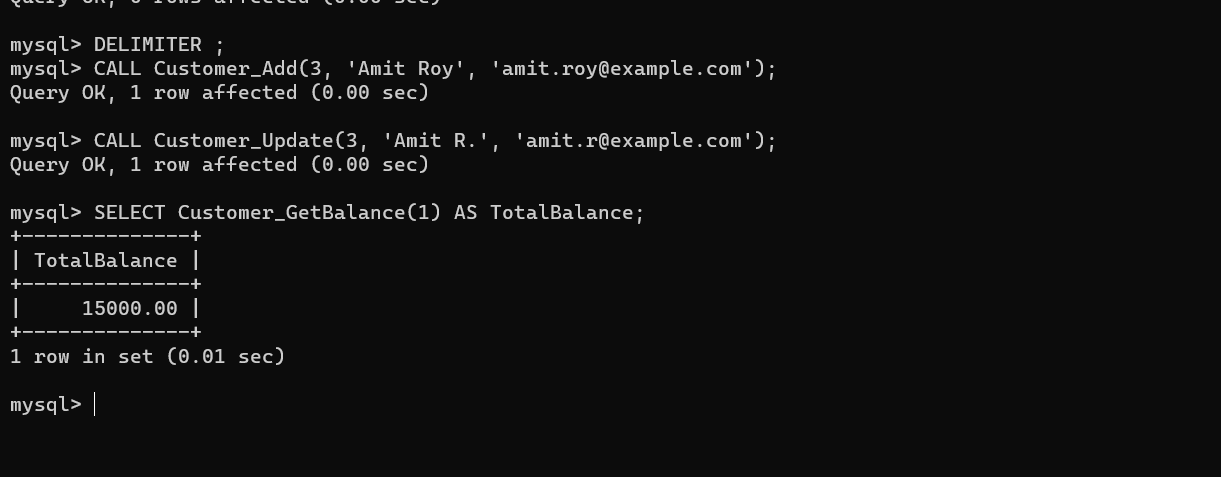
RETURN total;

END;

//

DELIMITER ;

Output:



**Scenario 2:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Position VARCHAR(100),

Salary DECIMAL(10,2)

);

INSERT INTO Employees VALUES (1, 'Abhay Kumar', 'Manager', 50000);

DELIMITER //

CREATE PROCEDURE Employee\_Hire(

IN p\_id INT, IN p\_name VARCHAR(100), IN p\_position VARCHAR(100), IN p\_salary DECIMAL(10,2)

)

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_position, p\_salary);

END;

//

DELIMITER ;

DELIMITER //

CREATE PROCEDURE Employee\_Update(

IN p\_id INT, IN p\_name VARCHAR(100), IN p\_position VARCHAR(100), IN p\_salary DECIMAL(10,2)

)

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary

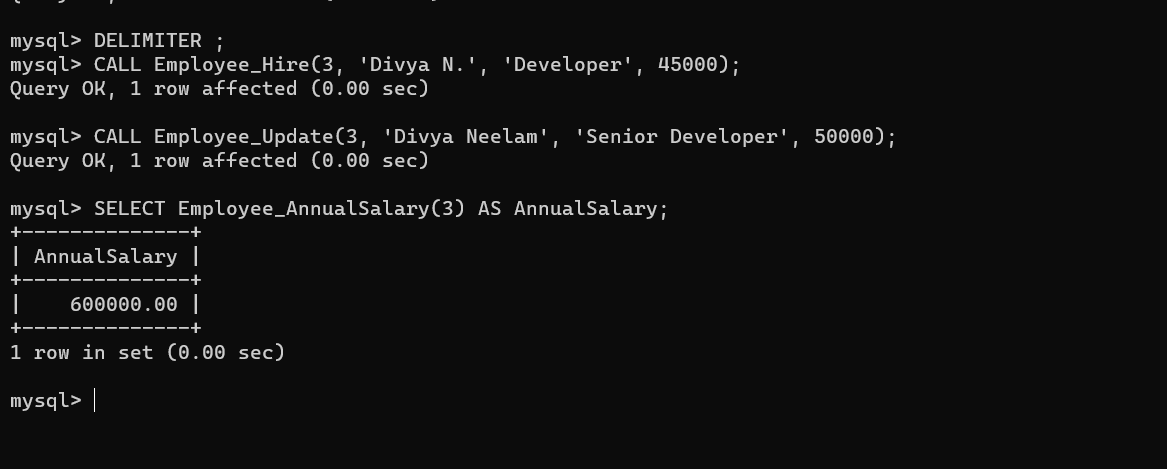
WHERE EmployeeID = p\_id;

END;

//

DELIMITER ;

Output:



**Scenario 3:**

DELIMITER //

CREATE PROCEDURE Account\_Open(

IN p\_acc\_id INT, IN p\_cust\_id INT, IN p\_balance DECIMAL(10,2)

)

BEGIN

INSERT INTO Accounts VALUES (p\_acc\_id, p\_cust\_id, p\_balance);

END;

//

DELIMITER ;

DELIMITER //

CREATE FUNCTION Account\_TotalBalance(p\_cust\_id INT)

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE total DECIMAL(10,2);

SELECT IFNULL(SUM(Balance), 0) INTO total FROM Accounts WHERE CustomerID = p\_cust\_id;

RETURN total;

END;

//

DELIMITER ;

DELIMITER //

CREATE PROCEDURE Account\_Close(IN p\_acc\_id INT)

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_acc\_id;

END;

//

DELIMITER ;

Output:

