# PLSQL\_Exercises

### Exercise 1: Control Structures

#### Scenario 1:

DELIMITER $$

CREATE PROCEDURE ApplyInterestDiscount() BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_custID INT;

DECLARE v\_dob DATE;

DECLARE cur CURSOR FOR SELECT CustomerID, DOB FROM Customers; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE

OPEN cur; read\_loop: LOOP

FETCH cur INTO v\_custID, v\_dob;

IF done THEN

LEAVE read\_loop;

END IF

IF TIMESTAMPDIFF(YEAR, v\_dob, CURDATE()) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1 WHERE CustomerID = v\_custID;

SELECT CONCAT('Applied 1% discount to CustomerID: ', v\_custID) AS Message; END IF;

END LOOP;

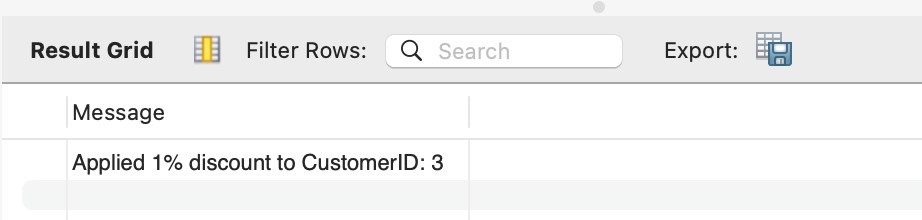
CLOSE cur;

END$$ DELIMITER ;

CALL ApplyInterestDiscount(); USE BankDB;

CALL ApplyInterestDiscount();

# output



##### Scenario 2:

DROP PROCEDURE IF EXISTS UpdateVIPStatus; DELIMITER $$

CREATE PROCEDURE UpdateVIPStatus() BEGIN

UPDATE Customers

SET IsVIP = CASE WHEN Balance > 10000 THEN TRUE ELSE FALSE END;

END$$ DELIMITER ;

SET SQL\_SAFE\_UPDATES = 0;

CALL UpdateVIPStatus(); DELIMITER $$

CREATE PROCEDURE UpdateVIPStatus()

BEGIN

UPDATE Customers

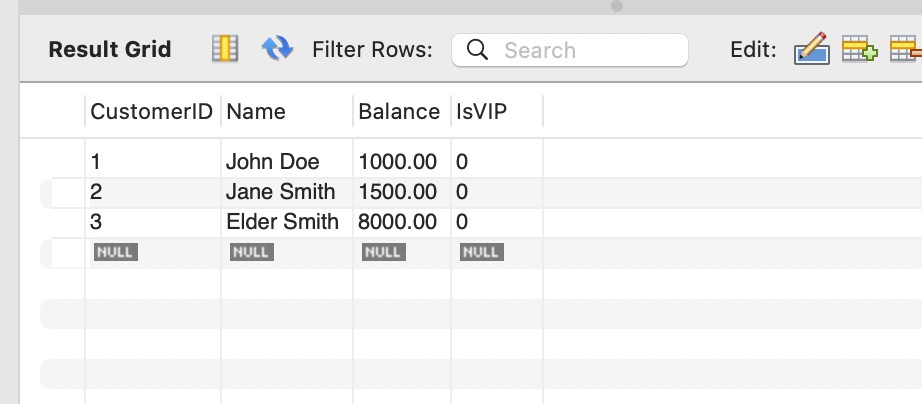
SET IsVIP = CASE WHEN Balance > 10000 THEN TRUE ELSE FALSE END;

END$$ DELIMITER ;

CALL UpdateVIPStatus();

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

# output



### Scenario 3:

DROP PROCEDURE IF EXISTS UpdateVIPStatus; DELIMITER $$

CREATE PROCEDURE UpdateVIPStatus() BEGIN

-- Update IsVIP flag: TRUE if balance > 10000, else FALSE UPDATE Customers

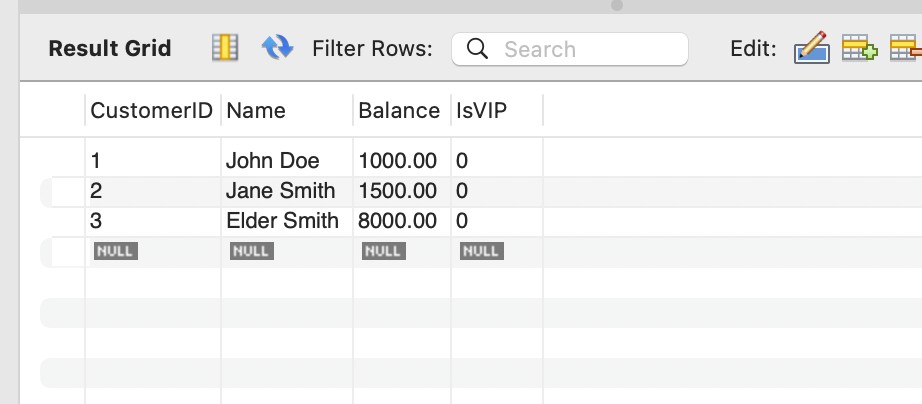
SET IsVIP = CASE WHEN Balance > 10000 THEN TRUE ELSE FALSE END;

END$$ DELIMITER ;

CALL UpdateVIPStatus();

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

Output



## Exercise 2: Error Handling

##### Scenario 1:

DROP PROCEDURE IF EXISTS SafeTransferFunds; DELIMITER $$

CREATE PROCEDURE SafeTransferFunds( IN p\_fromAccount INT,

IN p\_toAccount INT,

IN p\_amount DECIMAL(10,2)

)

BEGIN

DECLARE insufficient\_funds BOOL DEFAULT FALSE; DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

ROLLBACK;

SELECT 'Error: Transfer failed. Transaction rolled back.' AS ErrorMessage; END;

START TRANSACTION;

-- Check if source has sufficient funds

IF (SELECT Balance FROM Accounts WHERE AccountID = p\_fromAccount) < p\_amount THEN SET insufficient\_funds = TRUE;

ELSE

-- Deduct from source account UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_fromAccount;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount WHERE AccountID = p\_toAccount; COMMIT;

END IF;

IF insufficient\_funds THEN ROLLBACK;

SELECT 'Error: Insufficient funds in source account.' AS ErrorMessage; ELSE

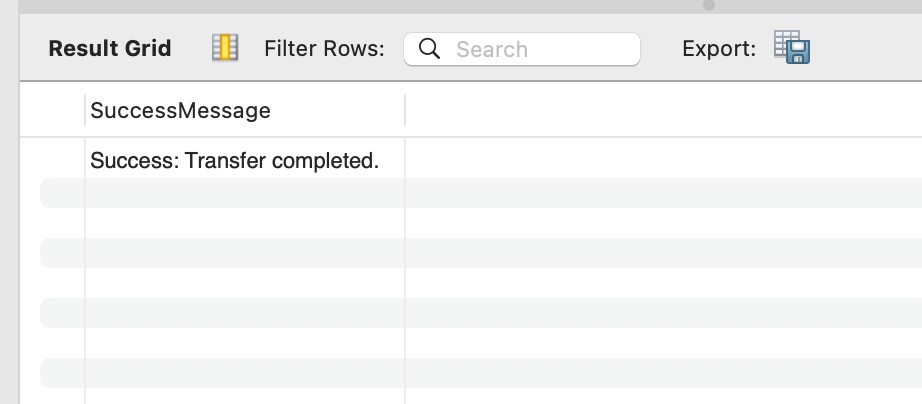
SELECT 'Success: Transfer completed.' AS SuccessMessage; END IF;

END$$

DELIMITER ;

CALL SafeTransferFunds(1, 2, 500.00);

# output



##### Scenario 2:

DROP PROCEDURE IF EXISTS UpdateSalary; DELIMITER $$

CREATE PROCEDURE UpdateSalary(

IN p\_empID INT,

IN p\_percent DECIMAL(5,2)

)

BEGIN

-- Declare variables first DECLARE v\_exists INT;

-- Then declare handlers

DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN

ROLLBACK;

SELECT CONCAT('Error: Could not update salary for employee ID ', p\_empID) AS ErrorMessage;

END;

-- Check if employee exists

SELECT COUNT(\*) INTO v\_exists FROM Employees WHERE EmployeeID = p\_empID; IF v\_exists = 0 THEN

SELECT CONCAT('Error: Employee ID ', p\_empID, ' does not exist.') AS ErrorMessage; ELSE

START TRANSACTION;

-- Update salary UPDATE Employees

SET Salary = Salary + (Salary \* p\_percent / 100)

WHERE EmployeeID = p\_empID; COMMIT;

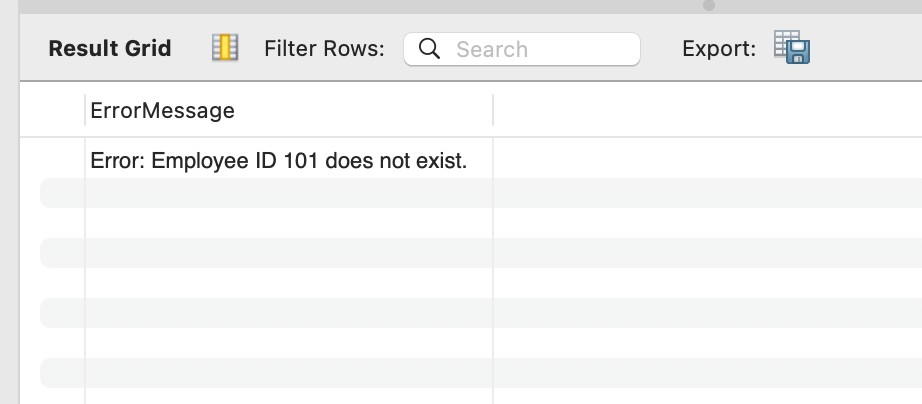
SELECT CONCAT('Success: Salary updated for employee ID ', p\_empID) AS SuccessMessage;

END IF;

END$$ DELIMITER ;

CALL UpdateSalary(101, 10);

# output



##### Scenario 3:

DROP PROCEDURE IF EXISTS AddNewCustomer; DELIMITER $$

CREATE PROCEDURE AddNewCustomer(

IN p\_customerID INT,

IN p\_name VARCHAR(100),

IN p\_dob DATE,

IN p\_balance DECIMAL(10,2)

)

BEGIN

-- Declare variables DECLARE v\_exists INT;

-- Exception handler

DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN

SELECT CONCAT('Error: Could not insert customer ID ', p\_customerID) AS ErrorMessage; END;

-- Check if the customer already exists

SELECT COUNT(\*) INTO v\_exists FROM Customers WHERE CustomerID = p\_customerID; IF v\_exists > 0 THEN

SELECT CONCAT('Error: Customer with ID ', p\_customerID, ' already exists.') AS ErrorMessage;

ELSE

-- Insert the new customer

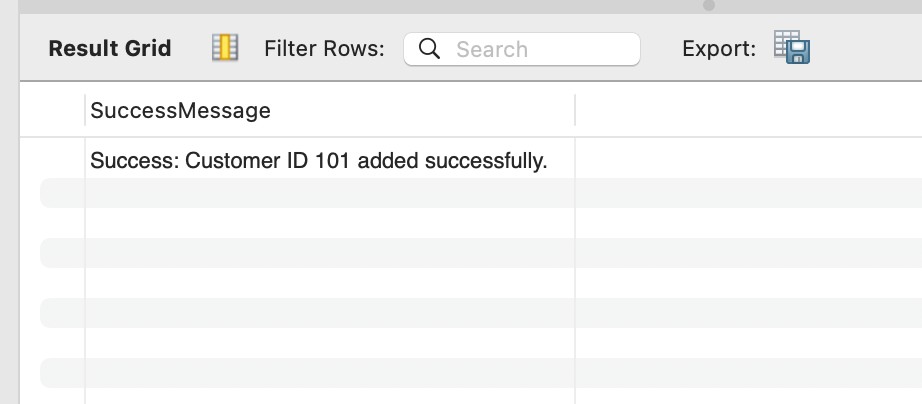
INSERT INTO Customers (CustomerID, Name, DOB, Balance) VALUES (p\_customerID, p\_name, p\_dob, p\_balance);

SELECT CONCAT('Success: Customer ID ', p\_customerID, ' added successfully.') AS SuccessMessage;

END IF;

END$$ DELIMITER ;

# output



#### Exercise 3: Stored Procedures Scenario 1:

DROP PROCEDURE IF EXISTS ProcessMonthlyInterest; DELIMITER $$

CREATE PROCEDURE ProcessMonthlyInterest() BEGIN

-- Update all savings accounts with 1% interest UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

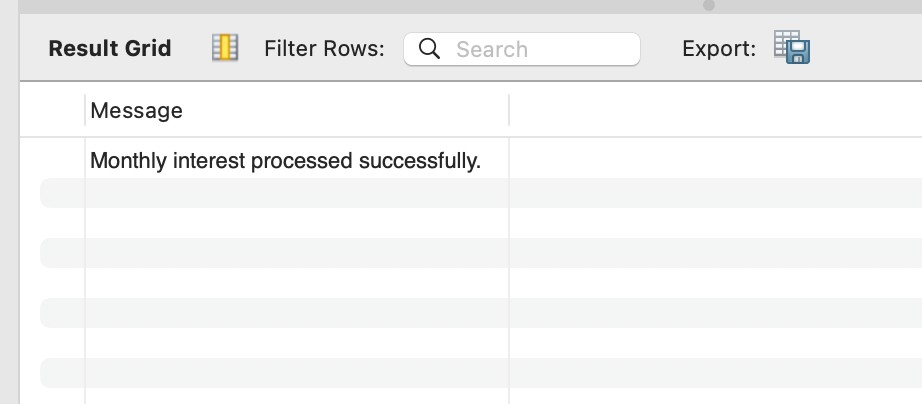
WHERE AccountType = 'Savings';

SELECT 'Monthly interest processed successfully.' AS Message; END$$

DELIMITER ;

CALL ProcessMonthlyInterest();

output



##### Scenario 2:

DROP PROCEDURE IF EXISTS UpdateEmployeeBonus; DELIMITER $$

CREATE PROCEDURE UpdateEmployeeBonus(

IN p\_department VARCHAR(100), IN p\_bonusPercent DECIMAL(5,2)

)

BEGIN

-- Apply bonus to employees in the specified department UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonusPercent / 100)

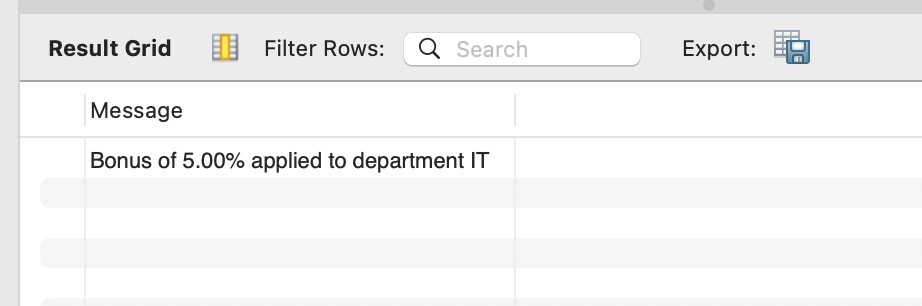
WHERE Department = p\_department;

SELECT CONCAT('Bonus of ', p\_bonusPercent, '% applied to department ', p\_department) AS Message;

END$$ DELIMITER ;

CALL UpdateEmployeeBonus('IT', 5.0);

# output



#### Scenario 3:

DROP PROCEDURE IF EXISTS TransferFunds; DELIMITER $$

CREATE PROCEDURE TransferFunds( IN p\_fromAccount INT,

IN p\_toAccount INT,

IN p\_amount DECIMAL(10,2)

)

BEGIN

DECLARE v\_balance DECIMAL(10,2);

-- Get balance of source account

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_fromAccount;

IF v\_balance < p\_amount THEN

SELECT 'Error: Insufficient balance.' AS ErrorMessage; ELSE

START TRANSACTION;

-- Deduct from source UPDATE Accounts

SET Balance = Balance - p\_amount WHERE AccountID = p\_fromAccount;

-- Add to destination UPDATE Accounts

SET Balance = Balance + p\_amount WHERE AccountID = p\_toAccount;

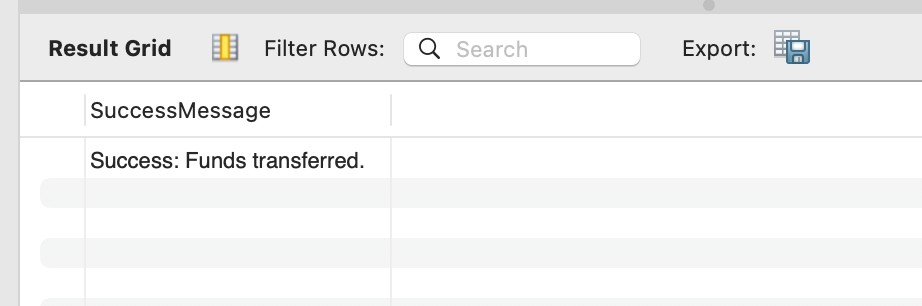
COMMIT;

SELECT 'Success: Funds transferred.' AS SuccessMessage; END IF;

END$$

DELIMITER ;

# output



#### Exercise 4: Functions

##### Scenario 1:

DROP FUNCTION IF EXISTS CalculateAge; DELIMITER $$

CREATE FUNCTION CalculateAge(p\_dob DATE) RETURNS INT

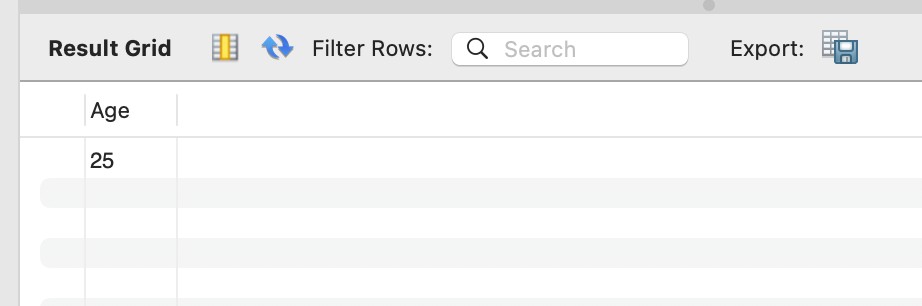
###### DETERMINISTIC BEGIN

RETURN TIMESTAMPDIFF(YEAR, p\_dob, CURDATE());

###### END$$ DELIMITER ;

SELECT CalculateAge('2000-05-21') AS Age;

# output



##### Scenario 2:

DROP FUNCTION IF EXISTS CalculateMonthlyInstallment; DELIMITER $$

CREATE FUNCTION CalculateMonthlyInstallment(

p\_amount DECIMAL(10,2), p\_annualRate DECIMAL(5,2), p\_years INT

)

RETURNS DECIMAL(10,2) DETERMINISTIC

BEGIN

DECLARE r DECIMAL(10,6); -- monthly rate DECLARE n INT; -- total number of months DECLARE emi DECIMAL(10,2);

SET r = p\_annualRate / (12 \* 100); SET n = p\_years \* 12;

SET emi = (p\_amount \* r \* POW(1 + r, n)) / (POW(1 + r, n) - 1);

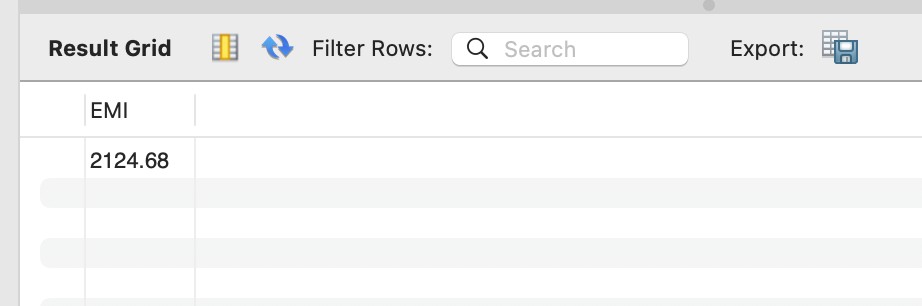
RETURN emi;

END$$

DELIMITER ;

SELECT CalculateMonthlyInstallment(100000, 10, 5) AS EMI;

# output



Scenario 3:

DROP FUNCTION IF EXISTS HasSufficientBalance; DELIMITER $$

CREATE FUNCTION HasSufficientBalance( p\_accountID INT,

p\_amount DECIMAL(10,2)

)

###### RETURNS BOOLEAN DETERMINISTIC BEGIN

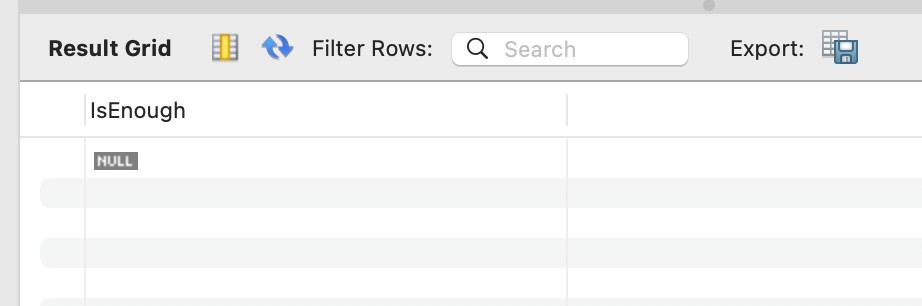
DECLARE v\_balance DECIMAL(10,2); SELECT Balance INTO v\_balance FROM Accounts

WHERE AccountID = p\_accountID; RETURN v\_balance >= p\_amount;

###### END$$ DELIMITER ;

SELECT HasSufficientBalance(101, 500.00) AS IsEnough;

# output



### Exercise 5: Triggers

Scenario 1

DROP TRIGGER IF EXISTS UpdateCustomerLastModified; DELIMITER $$

CREATE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers FOR EACH ROW

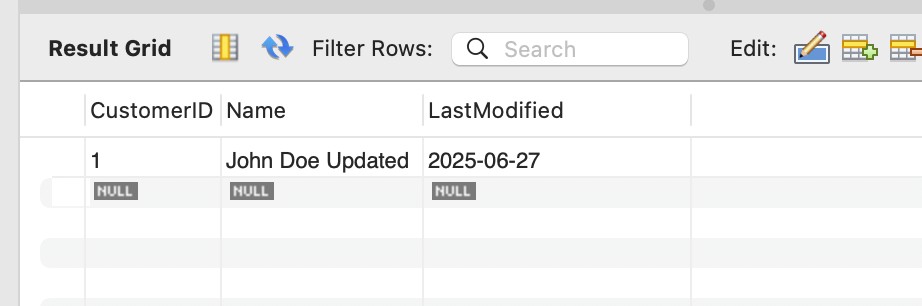
BEGIN

SET NEW.LastModified = NOW(); END$$

DELIMITER ;

SELECT CustomerID, LastModified FROM Customers WHERE CustomerID = 101;

output



### Scenario 2:

DELIMITER $$

CREATE TRIGGER LogTransaction AFTER INSERT ON Transactions FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, Amount, TransactionDate, Action)

VALUES (NEW.TransactionID, NEW.AccountID, NEW.Amount, NEW.TransactionDate, 'INSERT'); END$$

DELIMITER ;

CREATE TABLE AuditLog (

AuditID INT AUTO\_INCREMENT PRIMARY KEY,

TransactionID INT, AccountID INT,

Amount DECIMAL(15,2),

TransactionDate DATE, Action VARCHAR(20),

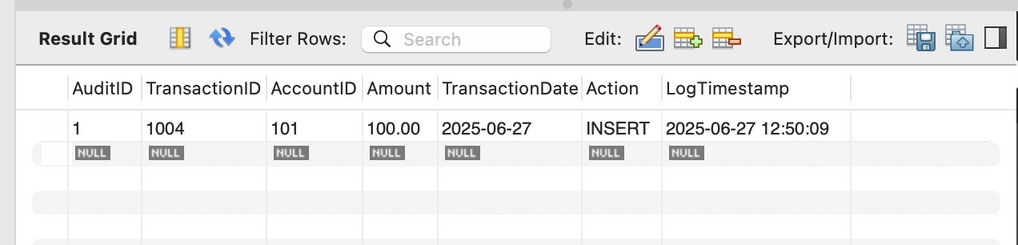
LogTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionDate) VALUES (1004, 101, 100.00, CURDATE());

SELECT \* FROM AuditLog WHERE TransactionID = 1004;

# output



### Scenario 3:

DELIMITER $$

CREATE TRIGGER CheckTransactionRules BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

DECLARE current\_balance DECIMAL(15,2);

-- Get current balance from Accounts table

SELECT Balance INTO current\_balance FROM Accounts WHERE AccountID = NEW.AccountID;

-- Check deposit amount must be positive IF NEW.Amount <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Error: Deposit or withdrawal amount must be positive';

END IF;

-- If withdrawal (assume negative amount means withdrawal) IF NEW.Amount < 0 THEN

-- Ensure withdrawal does not exceed current balance IF current\_balance < ABS(NEW.Amount) THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Error: Insufficient balance for withdrawal';

END IF;

END IF;

END$$

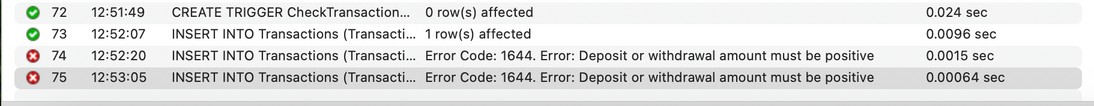
DELIMITER ;

INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionDate) VALUES (1005, 101, 200.00, CURDATE());

INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionDate) VALUES (1006, 101, -50.00, CURDATE());

INSERT INTO Transactions (TransactionID, AccountID, Amount, TransactionDate) VALUES (1007, 101, -100000.00, CURDATE());

# output



### Exercise 6: Cursors

##### Scenario 1:

DELIMITER //

CREATE PROCEDURE GenerateMonthlyStatements() BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_CustomerID INT; DECLARE v\_TransactionID INT; DECLARE v\_Amount DECIMAL(10,2);

DECLARE v\_TransactionDate DATE; DECLARE cur\_transactions CURSOR FOR

SELECT a.CustomerID, t.TransactionID, t.Amount, t.TransactionDate FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

WHERE DATE\_FORMAT(t.TransactionDate, '%Y-%m') = DATE\_FORMAT(CURDATE(), '%Y-

%m');

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur\_transactions; read\_loop: LOOP

FETCH cur\_transactions INTO v\_CustomerID, v\_TransactionID, v\_Amount, v\_TransactionDate;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT('Customer ', v\_CustomerID, ' Transaction ', v\_TransactionID,

' Amount: ', v\_Amount,

' Date: ', DATE\_FORMAT(v\_TransactionDate, '%d-%b-%Y')) AS Statement; END LOOP;

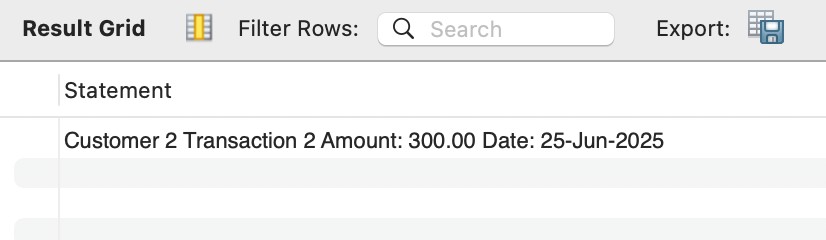
CLOSE cur\_transactions;

END;

// DELIMITER ;

DROP PROCEDURE IF EXISTS GenerateMonthlyStatements; CALL GenerateMonthlyStatements();

# output



##### Scenario 2:

DELIMITER //

CREATE PROCEDURE ApplyAnnualFee() BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_AccountID INT; DECLARE v\_Balance DECIMAL(10,2);

DECLARE cur\_accounts CURSOR FOR

SELECT AccountID, Balance FROM Accounts;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur\_accounts; read\_loop: LOOP

FETCH cur\_accounts INTO v\_AccountID, v\_Balance; IF done THEN

LEAVE read\_loop;

END IF;

-- Deduct fee if balance is sufficient

IF v\_Balance >= 100 THEN UPDATE Accounts

SET Balance = Balance - 100 WHERE AccountID = v\_AccountID;

ELSE

-- Optional: handle insufficient balance scenario

SELECT CONCAT('Account ', v\_AccountID, ' has insufficient balance for annual fee deduction.') AS WarningMessage;

END IF;

END LOOP;

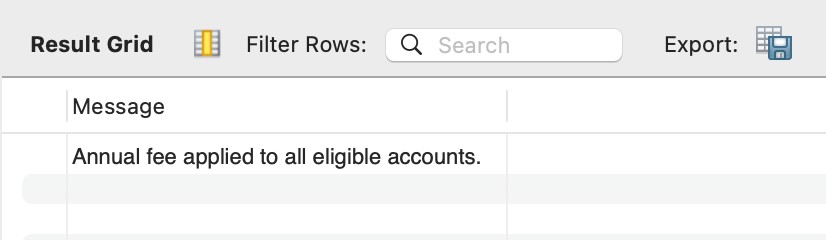
CLOSE cur\_accounts;

SELECT 'Annual fee applied to all eligible accounts.' AS Message; END;

// DELIMITER ;

CALL ApplyAnnualFee();

# output



### Scenario 3:

DELIMITER //

CREATE PROCEDURE UpdateLoanInterestRates() BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_LoanID INT;

DECLARE v\_InterestRate DECIMAL(5,2); DECLARE v\_LoanAmount DECIMAL(10,2); DECLARE cur\_loans CURSOR FOR

SELECT LoanID, InterestRate, LoanAmount FROM Loans; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur\_loans; read\_loop: LOOP

FETCH cur\_loans INTO v\_LoanID, v\_InterestRate, v\_LoanAmount; IF done THEN

LEAVE read\_loop;

END IF;

IF v\_LoanAmount > 50000 THEN

UPDATE Loans

SET InterestRate = v\_InterestRate + 0.5 WHERE LoanID = v\_LoanID;

ELSE

UPDATE Loans

SET InterestRate = v\_InterestRate + 0.2 WHERE LoanID = v\_LoanID;

END IF;

END LOOP;

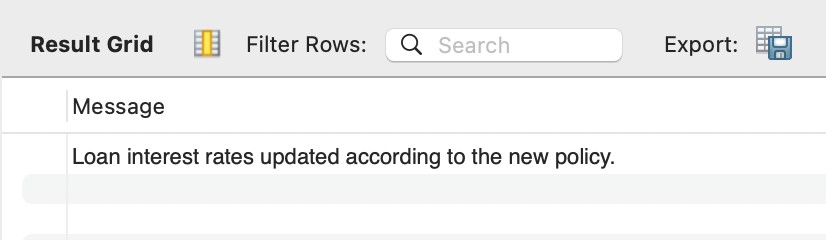
CLOSE cur\_loans;

SELECT 'Loan interest rates updated according to the new policy.' AS Message; END;

// DELIMITER ;

CALL UpdateLoanInterestRates();

# output



#### Exercise 7: Packages

##### Scenario 1:

DELIMITER //

CREATE PROCEDURE CustomerManagement\_AddCustomer ( IN p\_Name VARCHAR(100),

IN p\_Email VARCHAR(100),

IN p\_Balance DECIMAL(10,2)

)

BEGIN

INSERT INTO Customers (Name, Email, Balance) VALUES (p\_Name, p\_Email, p\_Balance);

END;

// DELIMITER ;

DELIMITER //

CREATE PROCEDURE CustomerManagement\_UpdateCustomer ( IN p\_CustomerID INT,

IN p\_Email VARCHAR(100)

)

BEGIN

UPDATE Customers SET Email = p\_Email

WHERE CustomerID = p\_CustomerID;

END;

// DELIMITER ;

DELIMITER //

CREATE FUNCTION CustomerManagement\_GetBalance(p\_CustomerID INT) RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE v\_balance DECIMAL(10,2);

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_CustomerID; RETURN v\_balance;

END;

// DELIMITER ; SELECT

TABLE\_NAME,

CONSTRAINT\_NAME, COLUMN\_NAME,

REFERENCED\_TABLE\_NAME, REFERENCED\_COLUMN\_NAME FROM

INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE WHERE

REFERENCED\_TABLE\_NAME = 'Customers'

AND REFERENCED\_COLUMN\_NAME = 'CustomerID' AND CONSTRAINT\_SCHEMA = 'BankDB';

ALTER TABLE Loans DROP FOREIGN KEY loans\_ibfk\_1;

ALTER TABLE Customers MODIFY CustomerID INT NOT NULL AUTO\_INCREMENT; ALTER TABLE Loans

ADD CONSTRAINT loans\_ibfk\_1 FOREIGN KEY (CustomerID)

REFERENCES Customers(CustomerID);

CALL CustomerManagement\_AddCustomer('Hemavathy', ['hema@email.com',](mailto:%27hema@email.com) 6000); SELECT \* FROM Customers WHERE Email = ['hema@email.com';](mailto:%27hema@email.com)

# output

## Scenario 2:

DROP PROCEDURE IF EXISTS EmployeeManagement\_HireEmployee;

DROP PROCEDURE IF EXISTS EmployeeManagement\_UpdateEmployee;

DROP FUNCTION IF EXISTS EmployeeManagement\_CalculateAnnualSalary; DELIMITER $$

CREATE FUNCTION EmployeeManagement\_CalculateAnnualSalary(

p\_empID INT

) RETURNS DECIMAL(15,2) READS SQL DATA

BEGIN

DECLARE v\_salary DECIMAL(10,2);

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_empID; RETURN v\_salary \* 12;

END $$ DELIMITER ;

SELECT EmployeeManagement\_CalculateAnnualSalary(101) AS AnnualSalary;

SELECT Salary FROM Employees WHERE EmployeeID = 101; DELIMITER $$

CREATE FUNCTION EmployeeManagement\_CalculateAnnualSalary(p\_empID INT)

RETURNS DECIMAL(15,2) DETERMINISTIC

READS SQL DATA BEGIN

DECLARE v\_salary DECIMAL(10,2);

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_empID; IF v\_salary IS NULL THEN

RETURN 0; -- or NULL if you prefer ELSE

RETURN v\_salary \* 12; END IF;

END$$ DELIMITER ;

DROP FUNCTION IF EXISTS EmployeeManagement\_CalculateAnnualSalary; DELIMITER $$

CREATE FUNCTION EmployeeManagement\_CalculateAnnualSalary(p\_empID INT)

RETURNS DECIMAL(15,2) DETERMINISTIC

READS SQL DATA BEGIN

DECLARE v\_salary DECIMAL(10,2);

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_empID; IF v\_salary IS NULL THEN

RETURN 0; -- or NULL if you prefer

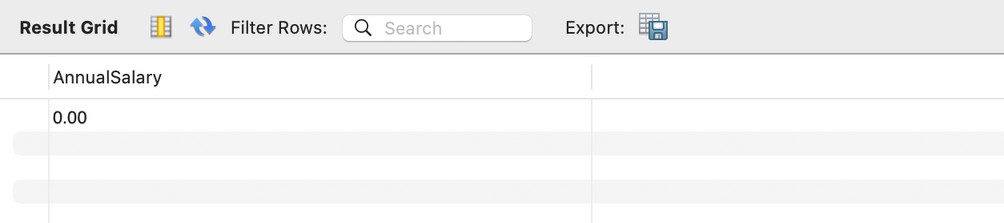
ELSE

RETURN v\_salary \* 12; END IF;

END$$ DELIMITER ;

SELECT EmployeeManagement\_CalculateAnnualSalary(101) AS AnnualSalary;

# output



### Scenario 3:

DELIMITER $$

CREATE PROCEDURE AccountOperations\_OpenAccount( IN p\_CustomerID INT,

IN p\_AccountType VARCHAR(50),

IN p\_InitialDeposit DECIMAL(15,2)

)

BEGIN

INSERT INTO Accounts (CustomerID, AccountType, Balance, OpenDate) VALUES (p\_CustomerID, p\_AccountType, p\_InitialDeposit, CURDATE()); SELECT 'Account opened successfully.' AS Message;

END$$ DELIMITER ;

-- Procedure to close an account DELIMITER $$

CREATE PROCEDURE AccountOperations\_CloseAccount( IN p\_AccountID INT

)

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID; SELECT 'Account closed successfully.' AS Message;

END$$ DELIMITER ; DELIMITER $$

CREATE FUNCTION AccountOperations\_GetTotalBalance( p\_CustomerID INT

)

RETURNS DECIMAL(15,2) DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE total\_balance DECIMAL(15,2);

SELECT IFNULL(SUM(Balance), 0) INTO total\_balance FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN total\_balance; END$$

DELIMITER ;

CALL AccountOperations\_OpenAccount(101, 'Savings', 1000.00); ALTER TABLE Accounts ADD COLUMN OpenDate DATE; DELIMITER $$

CREATE PROCEDURE AccountOperations\_OpenAccount( IN p\_CustomerID INT,

IN p\_AccountType VARCHAR(50), IN p\_InitialDeposit DECIMAL(15,2)

)

BEGIN

INSERT INTO Accounts (CustomerID, AccountType, Balance) VALUES (p\_CustomerID, p\_AccountType, p\_InitialDeposit); SELECT 'Account opened successfully.' AS Message;

END$$ DELIMITER ;

-- Step 1: Drop the foreign key constraint temporarily

ALTER TABLE Transactions DROP FOREIGN KEY transactions\_ibfk\_1;

-- Step 2: Drop the primary key on Accounts.AccountID ALTER TABLE Accounts DROP PRIMARY KEY;

-- Step 3: Modify AccountID to AUTO\_INCREMENT PRIMARY KEY

ALTER TABLE Accounts MODIFY AccountID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY;

-- Step 4: Recreate the foreign key constraint ALTER TABLE Transactions

ADD CONSTRAINT transactions\_ibfk\_1

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID);

INSERT INTO Accounts (CustomerID, AccountType, Balance, OpenDate) VALUES (101, 'Savings', 1000.00, CURDATE());

ALTER TABLE Transactions MODIFY TransactionID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY;

INSERT INTO Transactions (AccountID, Amount, TransactionDate) VALUES (LAST\_INSERT\_ID(), 500.00, CURDATE());

SHOW CREATE TABLE Transactions;

ALTER TABLE Transactions MODIFY TransactionID INT NOT NULL AUTO\_INCREMENT; INSERT INTO Transactions (AccountID, Amount, TransactionDate, TransactionType) VALUES (101, 500.00, CURDATE(), 'Deposit');

SELECT \* FROM Transactions WHERE AccountID = 101 ORDER BY TransactionID DESC LIMIT 1;

output

