**Exercise 3: Stored Procedures**

**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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

AS

BEGIN

-- Update the balance for all savings accounts by applying a 1% interest rate

UPDATE Accounts

SET Balance = Balance \* 1.01, -- Apply 1% interest

LastModified = SYSDATE

WHERE AccountType = 'Savings';

-- Commit the transaction to make the changes persistent

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest has been processed for all savings accounts.');

EXCEPTION

WHEN OTHERS THEN

-- Handle any unexpected errors

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END ProcessMonthlyInterest;

/

**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

)

AS

BEGIN

-- Update the salary for employees in the specified department by applying the bonus percentage

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percentage / 100), -- Apply the bonus percentage

LastModified = SYSDATE

WHERE Department = p\_department;

-- Commit the transaction to make the changes persistent

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salaries updated with a bonus of ' || p\_bonus\_percentage || '% for department ' || p\_department);

EXCEPTION

WHEN OTHERS THEN

-- Handle any unexpected errors

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END UpdateEmployeeBonus;

/

**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.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

AS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

ex\_insufficient\_funds EXCEPTION;

BEGIN

-- Check if the source account has sufficient balance

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE ex\_insufficient\_funds;

END IF;

-- Check if the destination account exists

BEGIN

SELECT Balance INTO v\_to\_balance

FROM Accounts

WHERE AccountID = p\_to\_account\_id

FOR UPDATE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Destination account ' || p\_to\_account\_id || ' does not exist.');

END;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

-- Add the amount to the destination account

UPDATE Accounts

SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Successfully transferred ' || p\_amount || ' from account ' || p\_from\_account\_id || ' to account ' || p\_to\_account\_id);

EXCEPTION

WHEN ex\_insufficient\_funds THEN

-- Handle insufficient funds error

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account ' || p\_from\_account\_id);

WHEN OTHERS THEN

-- Handle any other unexpected errors

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END TransferFunds;

/