# 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.

**Script:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

v\_interest\_rate NUMBER := 0.01;

v\_updated\_accounts NUMBER := 0;

BEGIN

UPDATE Accounts

SET Balance = Balance \* (1 + v\_interest\_rate),

LastModified = SYSDATE

WHERE AccountType = 'Savings';

v\_updated\_accounts := SQL%ROWCOUNT;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% monthly interest to ' || v\_updated\_accounts || ' savings accounts.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END ProcessMonthlyInterest;

/

EXEC ProcessMonthlyInterest;

**Script output:**

EXEC ProcessMonthlyInterest

Applied 1% monthly interest to 1 savings accounts.

PL/SQL procedure successfully completed.

**DBMS output:**

Applied 1% monthly interest to 1 savings accounts.

**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.

**Script:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) AS

v\_updated\_employees NUMBER := 0;

BEGIN

IF p\_bonus\_percentage < 0 OR p\_bonus\_percentage > 100 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Bonus percentage must be between 0 and 100');

END IF;

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percentage/100)

WHERE Department = p\_department;

v\_updated\_employees := SQL%ROWCOUNT;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Applied ' || p\_bonus\_percentage || '% bonus to ' ||

v\_updated\_employees || ' employees in ' || p\_department || ' department.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateEmployeeBonus;

/

EXEC UpdateEmployeeBonus('IT', 10);

**Script output:**

EXEC UpdateEmployeeBonus('IT', 10)

Applied 10% bonus to 1 employees in IT department.

PL/SQL procedure successfully completed.

**DBMS output:**

Applied 10% bonus to 1 employees in IT department.

**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.

**Script:**

CREATE SEQUENCE TransactionSeq

START WITH 100

INCREMENT BY 1

NOCACHE

NOCYCLE;

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER,

p\_result OUT VARCHAR2

) AS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

v\_from\_customer NUMBER;

v\_to\_customer NUMBER;

BEGIN

IF p\_amount <= 0 THEN

p\_result := 'Error: Transfer amount must be positive';

RETURN;

END IF;

BEGIN

SELECT Balance, CustomerID INTO v\_from\_balance, v\_from\_customer

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

SELECT Balance, CustomerID INTO v\_to\_balance, v\_to\_customer

FROM Accounts

WHERE AccountID = p\_to\_account\_id

FOR UPDATE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

p\_result := 'Error: One or both accounts not found';

ROLLBACK;

RETURN;

END;

IF v\_from\_balance < p\_amount THEN

p\_result := 'Error: Insufficient funds in source account';

ROLLBACK;

RETURN;

END IF;

IF v\_from\_customer != v\_to\_customer THEN

p\_result := 'Error: Accounts must belong to the same customer';

ROLLBACK;

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (TransactionSeq.NEXTVAL, p\_from\_account\_id, SYSDATE, p\_amount, 'Transfer Out');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (TransactionSeq.NEXTVAL, p\_to\_account\_id, SYSDATE, p\_amount, 'Transfer In');

COMMIT;

p\_result := 'Success: Transfer completed';

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount || ' from account ' || p\_from\_account\_id ||

' to account ' || p\_to\_account\_id);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

p\_result := 'Error: ' || SUBSTR(SQLERRM, 1, 200);

END TransferFunds;

/

DECLARE

v\_result VARCHAR2(200);

BEGIN

TransferFunds(1, 2, 500, v\_result);

DBMS\_OUTPUT.PUT\_LINE(v\_result);

END;

/

**Script output:**

Error: Accounts must belong to the same customer

PL/SQL procedure successfully completed.

**DBMS output:**

Error: Accounts must belong to the same customer