**Exercise 3: Stored Procedures**

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

**CODE :**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

CURSOR savings\_accounts\_cursor IS

SELECT ACCOUNT\_ID, BALANCE

FROM SAVINGS\_ACCOUNTS

FOR UPDATE OF BALANCE;

current\_account\_id SAVINGS\_ACCOUNTS.ACCOUNT\_ID%TYPE;

current\_balance SAVINGS\_ACCOUNTS.BALANCE%TYPE;

new\_balance SAVINGS\_ACCOUNTS.BALANCE%TYPE;

interest\_rate CONSTANT NUMBER := 0.01; -- 1% interest rate

BEGIN

FOR account\_rec IN savings\_accounts\_cursor LOOP

current\_account\_id := account\_rec.ACCOUNT\_ID;

current\_balance := account\_rec.BALANCE;

-- Calculate new balance by applying the interest rate

new\_balance := current\_balance \* (1 + interest\_rate);

-- Update the account balance

UPDATE SAVINGS\_ACCOUNTS

SET BALANCE = new\_balance

WHERE CURRENT OF savings\_accounts\_cursor;

END LOOP;

-- Commit the changes

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Rollback in case of an error

ROLLBACK;

RAISE;

END ProcessMonthlyInterest;

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

**CODE :** CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN EMPLOYEES.DEPARTMENT\_ID%TYPE,

p\_bonus\_percentage IN NUMBER

) IS

employee\_salary EMPLOYEES.SALARY%TYPE;

bonus\_amount NUMBER;

BEGIN

-- Loop through all employees in the specified department

FOR emp\_rec IN (SELECT EMPLOYEE\_ID, SALARY

FROM EMPLOYEES

WHERE DEPARTMENT\_ID = p\_department\_id) LOOP

-- Calculate the bonus amount

bonus\_amount := emp\_rec.SALARY \* (p\_bonus\_percentage / 100);

-- Update the employee's salary by adding the bonus amount

UPDATE EMPLOYEES

SET SALARY = SALARY + bonus\_amount

WHERE EMPLOYEE\_ID = emp\_rec.EMPLOYEE\_ID;

END LOOP;

-- Commit the changes

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Rollback in case of an error

ROLLBACK;

RAISE;

END UpdateEmployeeBonus;

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

**CODE :**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN ACCOUNTS.ACCOUNT\_ID%TYPE,

p\_to\_account\_id IN ACCOUNTS.ACCOUNT\_ID%TYPE,

p\_amount IN NUMBER

) IS

from\_account\_balance ACCOUNTS.BALANCE%TYPE;

to\_account\_balance ACCOUNTS.BALANCE%TYPE;

-- Custom exception for insufficient funds

insufficient\_funds EXCEPTION;

PRAGMA EXCEPTION\_INIT(insufficient\_funds, -20001);

BEGIN

-- Lock the rows for both accounts to prevent concurrent updates

SELECT BALANCE INTO from\_account\_balance

FROM ACCOUNTS

WHERE ACCOUNT\_ID = p\_from\_account\_id

FOR UPDATE;

SELECT BALANCE INTO to\_account\_balance

FROM ACCOUNTS

WHERE ACCOUNT\_ID = p\_to\_account\_id

FOR UPDATE;

-- Check if the source account has sufficient funds

IF from\_account\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Perform the fund transfer

UPDATE ACCOUNTS

SET BALANCE = BALANCE - p\_amount

WHERE ACCOUNT\_ID = p\_from\_account\_id;

UPDATE ACCOUNTS

SET BALANCE = BALANCE + p\_amount

WHERE ACCOUNT\_ID = p\_to\_account\_id;

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

-- Handle insufficient funds

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in account ' || p\_from\_account\_id);

WHEN NO\_DATA\_FOUND THEN

-- Handle cases where one of the accounts does not exist

RAISE\_APPLICATION\_ERROR(-20002, 'One or both account IDs are invalid.');

WHEN OTHERS THEN

-- Handle any other unexpected errors

ROLLBACK;

RAISE;

END TransferFunds;