**EXERCISE 2: ERROR HANDLING**

**Scenario 1: SafeTransferFunds - Handle Exceptions During Fund Transfers**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_FromAccountID IN Accounts.AccountID%TYPE,

p\_ToAccountID IN Accounts.AccountID%TYPE,

p\_Amount IN NUMBER

) IS

v\_FromBalance Accounts.Balance%TYPE;

v\_ToBalance Accounts.Balance%TYPE;

BEGIN

-- Lock the accounts for update to prevent race conditions

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

SELECT Balance INTO v\_ToBalance

FROM Accounts

WHERE AccountID = p\_ToAccountID

FOR UPDATE;

-- Check if the from account has enough balance

IF v\_FromBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

-- Perform the fund transfer

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

-- Commit the transaction

COMMIT;

-- Log success message

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_Amount || ' from account ' || p\_FromAccountID || ' to account ' || p\_ToAccountID || ' completed successfully.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: One of the account IDs does not exist.');

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

/

**Scenario 2: UpdateSalary - Manage Errors When Updating Employee Salaries**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_Percentage IN NUMBER

) IS

v\_CurrentSalary Employees.Salary%TYPE;

BEGIN

-- Fetch the current salary of the employee

SELECT Salary INTO v\_CurrentSalary

FROM Employees

WHERE EmployeeID = p\_EmployeeID

FOR UPDATE;

-- Calculate the new salary

v\_CurrentSalary := v\_CurrentSalary \* (1 + p\_Percentage / 100);

-- Update the employee's salary

UPDATE Employees

SET Salary = v\_CurrentSalary

WHERE EmployeeID = p\_EmployeeID;

-- Commit the transaction

COMMIT;

-- Log success message

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for employee ID ' || p\_EmployeeID || '. New salary: ' || v\_CurrentSalary);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' does not exist.');

WHEN OTHERS THEN

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

END UpdateSalary;

/

**Scenario 3: AddNewCustomer - Ensure Data Integrity When Adding a New Customer**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID IN Customers.CustomerID%TYPE,

p\_Name IN Customers.Name%TYPE,

p\_DOB IN Customers.DOB%TYPE,

p\_Balance IN Customers.Balance%TYPE

) IS

BEGIN

-- Attempt to insert the new customer

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

-- Commit the transaction

COMMIT;

-- Log success message

DBMS\_OUTPUT.PUT\_LINE('New customer added successfully with ID ' || p\_CustomerID);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: A customer with ID ' || p\_CustomerID || ' already exists.');

WHEN OTHERS THEN

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

END AddNewCustomer;

/