**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER;

ex\_insufficient\_funds EXCEPTION;

BEGIN

-- Get the balance of the from account

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

-- Check if there are sufficient funds

IF v\_from\_balance < p\_amount THEN

RAISE ex\_insufficient\_funds;

END IF;

-- Deduct the amount from the from account

UPDATE Accounts

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

WHERE AccountID = p\_from\_account\_id;

-- Add the amount to the to account

UPDATE Accounts

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

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN ex\_insufficient\_funds THEN

ROLLBACK;

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

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Account not found.');

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

/

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

v\_current\_salary NUMBER;

v\_new\_salary NUMBER;

BEGIN

-- Attempt to get the current salary of the employee

SELECT Salary INTO v\_current\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

-- Calculate the new salary

v\_new\_salary := v\_current\_salary \* (1 + p\_percentage / 100);

-- Update the salary

UPDATE Employees

SET Salary = v\_new\_salary, LastModified = SYSDATE

WHERE EmployeeID = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID ' || p\_employee\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle the case where the employee ID does not exist

ROLLBACK;

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

WHEN OTHERS THEN

-- Handle any other unexpected errors

ROLLBACK;

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

END UpdateSalary;

/

**Scenario 3:** Ensure data integrity when adding a new customer.

* **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

)

IS

v\_existing\_count NUMBER;

BEGIN

-- Check if a customer with the same ID already exists

SELECT COUNT(\*) INTO v\_existing\_count

FROM Customers

WHERE CustomerID = p\_customer\_id;

IF v\_existing\_count > 0 THEN

-- Customer ID already exists, handle the exception

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_customer\_id || ' already exists. Customer not added.');

ELSE

-- Insert the new customer

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

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully with ID ' || p\_customer\_id);

END IF;

EXCEPTION

WHEN OTHERS THEN

-- Handle any other unexpected errors

ROLLBACK;

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

END AddNewCustomer;

/