**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.

**Query:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

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;

insufficient\_funds EXCEPTION;

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

BEGIN

SAVEPOINT transfer\_start;

IF p\_amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Transfer amount must be greater than zero.');

END IF;

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

SELECT Balance INTO v\_to\_balance

FROM Accounts

WHERE AccountID = p\_to\_account\_id;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK TO transfer\_start;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds for the transfer.');

WHEN OTHERS THEN

ROLLBACK TO transfer\_start;

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

END SafeTransferFunds;

**To get the output:**

**Query:**

BEGIN

SafeTransferFunds(1, 2, 100);

END;

**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.

**Query:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_current\_salary NUMBER;

employee\_not\_found EXCEPTION;

PRAGMA EXCEPTION\_INIT(employee\_not\_found, -20001);

BEGIN

IF p\_percentage <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Percentage must be greater than zero.');

END IF;

BEGIN

SELECT Salary INTO v\_current\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

END UpdateSalary;

**To get the output:**

**Query:**

BEGIN

UpdateSalary(1, 10);

END;

**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.

**Query:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_last\_modified IN DATE

) AS

duplicate\_id EXCEPTION;

PRAGMA EXCEPTION\_INIT(duplicate\_id, -1);

BEGIN

BEGIN

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

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

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: CustomerID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

ROLLBACK;

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

END;

END AddNewCustomer;

**To get the Output:**

**Query:**

BEGIN

AddNewCustomer(1, 'Alice Johnson', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

**END;**