**Module 3 - PL/SQL Programming**

**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.

**OUERY:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds.');

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

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;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Transfer successful: ' || p\_amount || ' transferred from account ' || p\_from\_account\_id || ' to ' || p\_to\_account\_id);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

END;

/

**EXEC SafeTransferFunds(1, 2, 100);**



**EXEC SafeTransferFunds(1, 2, 999999);**



**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\_old\_salary NUMBER;

BEGIN

    SELECT Salary INTO v\_old\_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 Employee ID ' || p\_employee\_id);

    DBMS\_OUTPUT.PUT\_LINE('   Old Salary: ' || v\_old\_salary);

    DBMS\_OUTPUT.PUT\_LINE('   New Salary: ' || (v\_old\_salary + (v\_old\_salary \* p\_percentage / 100)));

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

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

    WHEN OTHERS THEN

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

END;

**EXEC UpdateSalary(1, 10)**



**EXEC UpdateSalary(9999, 15)**



**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

) AS

    v\_dummy NUMBER;

BEGIN

    SELECT 1 INTO v\_dummy

    FROM Customers

    WHERE CustomerID = p\_customer\_id;

    DBMS\_OUTPUT.PUT\_LINE(' Customer ID ' || p\_customer\_id || ' already exists. Insertion aborted.');

    INSERT INTO Error\_Log (Error\_Message)

    VALUES ('Attempted to insert duplicate Customer ID: ' || p\_customer\_id);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

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

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

        DBMS\_OUTPUT.PUT\_LINE(' New customer added successfully: ' || p\_name);

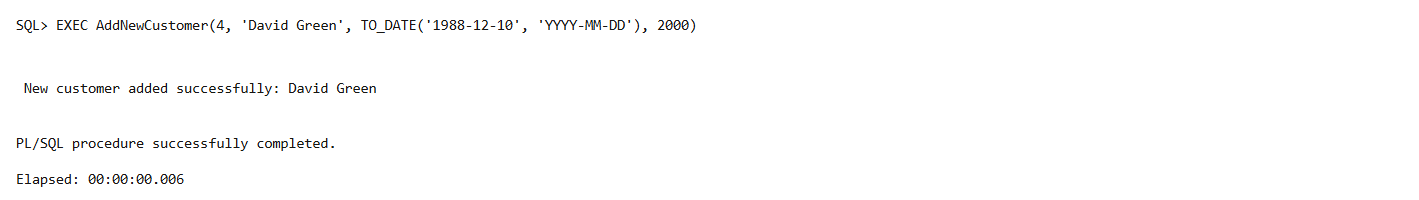
    WHEN OTHERS THEN

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

        RAISE;

END;

**EXEC AddNewCustomer(4, 'David Green', TO\_DATE('1988-12-10', 'YYYY-MM-DD'), 2000);**



**EXEC AddNewCustomer(1, 'Fake John', TO\_DATE('1995-11-11', 'YYYY-MM-DD'), 500);**

