**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.

**Ans:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

fromAccountID IN Accounts.AccountID%TYPE,

toAccountID IN Accounts.AccountID%TYPE,

amount IN NUMBER

) AS

fromBalance Accounts.Balance%TYPE;

toBalance Accounts.Balance%TYPE;

BEGIN

SELECT Balance INTO fromBalance

FROM Accounts

WHERE AccountID = fromAccountID

FOR UPDATE;

IF fromBalance < amount THEN

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

END IF;

SELECT Balance INTO toBalance

FROM Accounts

WHERE AccountID = toAccountID

FOR UPDATE;

UPDATE Accounts

SET Balance = fromBalance - amount

WHERE AccountID = fromAccountID;

UPDATE Accounts

SET Balance = toBalance + amount

WHERE AccountID = toAccountID;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error occurd during fund transfer: ' || 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.

**Ans:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employeeID IN Employees.EmployeeID%TYPE,

p\_percentage IN NUMBER

) AS

v\_salary Employees.Salary%TYPE;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employeeID;

UPDATE Employees

SET Salary = v\_salary \* (1 + p\_percentage / 100)

WHERE EmployeeID = p\_employeeID;

COMMIT;

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: More than one employee with ID ' || p\_employeeID || ' found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error during salary update: ' || 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.

**Ans:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

customerID IN Customers.CustomerID%TYPE,

c\_name IN Customers.Name%TYPE,

c\_dob IN Customers.DOB%TYPE,

c\_balance IN Customers.Balance%TYPE

) AS

BEGIN

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

VALUES (customerID, c\_name, c\_dob, c\_balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('New customer added successfully in the database.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: A customer with ID ' || customerID || 'is already exists in your database.');

WHEN OTHERS THEN

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

END AddNewCustomer;

/