**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 the TransactionErrors table

CREATE TABLE TransactionErrors (

ErrorID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

ErrorDate DATE,

ErrorMessage VARCHAR2(4000)

);

-- Create the SafeTransferFunds procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_FromAccountID IN Accounts.AccountID%TYPE,

p\_ToAccountID IN Accounts.AccountID%TYPE,

p\_Amount IN NUMBER

) AS

v\_FromBalance Accounts.Balance%TYPE;

BEGIN

-- Get the balance of the from account

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

-- Check if there are sufficient funds

IF v\_FromBalance < p\_Amount THEN

-- Log the error and raise an exception if insufficient funds

INSERT INTO TransactionErrors (ErrorDate, ErrorMessage)

VALUES (SYSDATE, 'Insufficient funds in account ID ' || p\_FromAccountID || '. Attempted transfer amount: ' || p\_Amount);

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

ELSE

-- Proceed with the transfer

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add the amount to the destination account

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

-- Commit the transaction

COMMIT;

-- Optionally log successful transfer

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from Account ' || p\_FromAccountID || ' to Account ' || p\_ToAccountID || ' of amount ' || p\_Amount);

END IF;

EXCEPTION

WHEN OTHERS THEN

-- Rollback the transaction in case of any error

ROLLBACK;

-- Log the error message

INSERT INTO TransactionErrors (ErrorDate, ErrorMessage)

VALUES (SYSDATE, 'Error during fund transfer: ' || SQLERRM);

-- Propagate the exception

RAISE;

END SafeTransferFunds;

/

BEGIN

SafeTransferFunds(1, 2, 500);

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.

-- Create the SalaryUpdateErrors table if not already created

CREATE TABLE SalaryUpdateErrors (

ErrorID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

ErrorDate DATE,

ErrorMessage VARCHAR2(4000)

);

-- Create or replace the UpdateSalary procedure

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_Percentage IN NUMBER

) AS

v\_CurrentSalary Employees.Salary%TYPE;

BEGIN

-- Fetch the current salary of the employee

SELECT Salary INTO v\_CurrentSalary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

-- Update the salary by the given percentage

UPDATE Employees

SET Salary = v\_CurrentSalary + (v\_CurrentSalary \* p\_Percentage / 100)

WHERE EmployeeID = p\_EmployeeID;

-- Commit the transaction

COMMIT;

-- Optionally log successful update

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID ' || p\_EmployeeID || ' by ' || p\_Percentage || '%');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle the case where no employee with the given ID exists

INSERT INTO SalaryUpdateErrors (ErrorDate, ErrorMessage)

VALUES (SYSDATE, 'Employee ID ' || p\_EmployeeID || ' does not exist.');

-- Raise an application error

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID does not exist.');

WHEN OTHERS THEN

-- Handle any other exceptions

INSERT INTO SalaryUpdateErrors (ErrorDate, ErrorMessage)

VALUES (SYSDATE, 'Error updating salary for Employee ID ' || p\_EmployeeID || ': ' || SQLERRM);

-- Rollback the transaction

ROLLBACK;

-- Propagate the exception

RAISE;

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 TABLE CustomerErrors (

ErrorID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

ErrorDate DATE,

ErrorMessage VARCHAR2(4000)

);

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

) AS

BEGIN

-- Attempt to insert the new customer record

BEGIN

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

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

-- Commit the transaction

COMMIT;

-- Optionally log successful insertion

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully: ID ' || p\_CustomerID || ', Name ' || p\_Name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

-- Handle the case where the customer ID already exists

INSERT INTO CustomerErrors (ErrorDate, ErrorMessage)

VALUES (SYSDATE, 'Customer ID ' || p\_CustomerID || ' already exists. Insert failed.');

-- Raise an application error

RAISE\_APPLICATION\_ERROR(-20003, 'Customer ID already exists.');

WHEN OTHERS THEN

-- Handle any other exceptions

INSERT INTO CustomerErrors (ErrorDate, ErrorMessage)

VALUES (SYSDATE, 'Error adding customer ID ' || p\_CustomerID || ': ' || SQLERRM);

-- Rollback the transaction

ROLLBACK;

-- Propagate the exception

RAISE;

END;

END AddNewCustomer;

/