**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 Accounts.AccountID%TYPE,

p\_to\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

)

AS

v\_from\_balance Accounts.Balance%TYPE;

v\_to\_balance Accounts.Balance%TYPE;

BEGIN

*-- Start a transaction*

BEGIN

*-- Lock the rows to avoid concurrency issues*

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

SELECT Balance INTO v\_to\_balance

FROM Accounts

WHERE AccountID = p\_to\_account\_id

FOR UPDATE;

*-- Check if the source account has enough funds*

IF v\_from\_balance < p\_amount THEN

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

END IF;

*-- Perform the fund transfer*

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 the transaction if no errors occurred*

COMMIT;

EXCEPTION

WHEN OTHERS THEN

*-- Rollback the transaction in case of any error*

ROLLBACK;

*-- Log the error message*

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

*-- Reraise the exception to notify the caller*

RAISE;

END;

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.

The error message will be logged in an ErrorLog table.

Let's create the ErrorLog table first:

CREATE TABLE ErrorLog (

ErrorID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

ErrorMessage VARCHAR2(4000),

ErrorDate DATE

);

UpdateSalary procedure:

CREATE PROCEDURE UpdateSalary (

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_PercentageIncrease IN NUMBER

)

IS

*-- Local variable to hold the number of rows updated*

v\_RowsUpdated NUMBER;

BEGIN

*-- Attempt to update the employee's salary*

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

*-- Get the number of rows updated*

v\_RowsUpdated := SQL%ROWCOUNT;

*-- Check if no rows were updated, meaning the EmployeeID does not exist*

IF v\_RowsUpdated = 0 THEN

*-- Raise an exception if the employee ID does not exist*

RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID ' || p\_EmployeeID || ' does not exist.');

END IF;

EXCEPTION

WHEN OTHERS THEN

*-- Log the error message if an exception occurs*

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (SQLERRM, SYSDATE)

*-- Raise the exception again to propagate it*

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 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

)

IS

BEGIN

*-- Attempt to insert a new customer*

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

*-- Log the error message if a duplicate CustomerID exists*

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

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

*-- Raise the exception again to prevent the insertion*

RAISE\_APPLICATION\_ERROR(-20002, 'Customer ID ' || p\_CustomerID || ' already exists.');

WHEN OTHERS THEN

*-- Log any other error messages*

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (SQLERRM, SYSDATE);

*-- Raise the exception again to propagate it*

RAISE;

END AddNewCustomer;

/

* If a DUP\_VAL\_ON\_INDEX exception is raised (indicating a duplicate CustomerID), we log an error message in the ErrorLog table and raise an application error to prevent the insertion.
* Any other exceptions are also logged in the ErrorLog table, and the exception is propagated.