**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.account\_id%TYPE,

p\_to\_account\_id IN accounts.account\_id%TYPE,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_balance accounts.balance%TYPE;

BEGIN

-- Check balance of the from account

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Deduct the amount from the from account

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account\_id;

-- Add the amount to the to account

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

-- Handle insufficient funds error

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);

WHEN OTHERS THEN

-- Handle any other unexpected errors

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Unable to transfer funds from account ' || p\_from\_account\_id ||

' to account ' || p\_to\_account\_id || '. Error: ' || 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

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN employees.employee\_id%TYPE,

p\_percentage IN NUMBER

) IS

employee\_not\_found EXCEPTION;

v\_current\_salary employees.salary%TYPE;

BEGIN

-- Check if employee exists

SELECT salary INTO v\_current\_salary

FROM employees

WHERE employee\_id = p\_employee\_id;

-- Increase the salary by the given percentage

UPDATE employees

SET salary = salary \* (1 + p\_percentage / 100)

WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle case where employee does not exist

RAISE employee\_not\_found;

WHEN employee\_not\_found THEN

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

WHEN OTHERS THEN

-- Handle any other unexpected errors

DBMS\_OUTPUT.PUT\_LINE('Error: Unable to update salary for employee ID ' || p\_employee\_id || '. Error: ' || 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.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN customers.customer\_id%TYPE,

p\_name IN customers.name%TYPE,

p\_age IN customers.age%TYPE,

p\_balance IN customers.balance%TYPE

) IS

duplicate\_customer EXCEPTION;

BEGIN

-- Attempt to insert a new customer

INSERT INTO customers (customer\_id, name, age, balance)

VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

-- Handle case where customer with same ID already exists

ROLLBACK;

RAISE duplicate\_customer;

WHEN duplicate\_customer THEN

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

WHEN OTHERS THEN

-- Handle any other unexpected errors

DBMS\_OUTPUT.PUT\_LINE('Error: Unable to add new customer with ID ' || p\_customer\_id || '. Error: ' || SQLERRM);

END AddNewCustomer;

/