**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. Scenario

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. Scenario

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.

Here are the PL/SQL stored procedures for the given scenarios:

**Scenario 1: Safe Fund Transfer Between Accounts**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

BEGIN

-- Check if the from account has sufficient funds

DECLARE

v\_from\_balance NUMBER;

BEGIN

SELECT balance INTO v\_from\_balance

FROM accounts

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

FOR UPDATE;

IF v\_from\_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

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

END SafeTransferFunds;

/

**Scenario 2: Manage Errors When Updating Employee Salaries**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

employee\_not\_found EXCEPTION;

BEGIN

-- Update the salary of the employee

DECLARE

v\_salary employees.salary%TYPE;

BEGIN

SELECT salary INTO v\_salary

FROM employees

WHERE employee\_id = p\_employee\_id

FOR UPDATE;

-- Increase the salary by the given percentage

v\_salary := v\_salary + (v\_salary \* p\_percentage / 100);

UPDATE employees

SET salary = v\_salary

WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE employee\_not\_found;

WHEN employee\_not\_found THEN

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

WHEN OTHERS THEN

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

END;

END UpdateSalary;

/

**Scenario 3: Ensure Data Integrity When Adding a New Customer**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_age IN NUMBER,

p\_address IN VARCHAR2

) IS

duplicate\_customer EXCEPTION;

BEGIN

-- Insert the new customer

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

RAISE duplicate\_customer;

WHEN duplicate\_customer THEN

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

WHEN OTHERS THEN

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

END;

END AddNewCustomer;

/

These stored procedures handle exceptions by checking for specific error conditions, logging appropriate error messages, and ensuring that the transaction is rolled back when necessary.