**Exercise 2: Error Handling**

**Scenario 1: Handle exceptions during fund transfers between accounts**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance accounts.balance%TYPE;

v\_to\_balance accounts.balance%TYPE;

BEGIN

-- Start transaction

SAVEPOINT sp\_transfer;

-- Check the balance of the from\_account

SELECT balance INTO v\_from\_balance

FROM accounts

WHERE accountid = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Update the from\_account balance

UPDATE accounts

SET balance = balance - p\_amount

WHERE accountid = p\_from\_account\_id;

-- Update the to\_account balance

UPDATE accounts

SET balance = balance + p\_amount

WHERE accountid = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK TO sp\_transfer;

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

-- Log the error in a log table (optional)

-- INSERT INTO error\_log (message, created\_at) VALUES (SQLERRM, SYSDATE);

END SafeTransferFunds;

/

**Scenario 2: Manage errors when updating employee salaries**

**Stored Procedure: UpdateSalary**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_salary employees.salary%TYPE;

BEGIN

-- Start transaction

SAVEPOINT sp\_salary;

-- Check if the employee exists and get the current salary

SELECT salary INTO v\_salary

FROM employees

WHERE employeeid = p\_employee\_id

FOR UPDATE;

-- Update the salary

UPDATE employees

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

WHERE employeeid = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK TO sp\_salary;

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

-- Log the error in a log table (optional)

-- INSERT INTO error\_log (message, created\_at) VALUES ('Employee ID not found', SYSDATE);

WHEN OTHERS THEN

ROLLBACK TO sp\_salary;

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

-- Log the error in a log table (optional)

-- INSERT INTO error\_log (message, created\_at) VALUES (SQLERRM, SYSDATE);

END UpdateSalary;

/

**Scenario 3: Ensure data integrity when adding a new customer**

**Stored Procedure: AddNewCustomer**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

BEGIN

-- Start transaction

SAVEPOINT sp\_add\_customer;

-- Insert the new customer

INSERT INTO customers (customerid, name, dob, balance, lastmodified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK TO sp\_add\_customer;

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with the same ID already exists.');

-- Log the error in a log table (optional)

-- INSERT INTO error\_log (message, created\_at) VALUES ('Customer ID duplicate', SYSDATE);

WHEN OTHERS THEN

ROLLBACK TO sp\_add\_customer;

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

-- Log the error in a log table (optional)

-- INSERT INTO error\_log (message, created\_at) VALUES (SQLERRM, SYSDATE);

END AddNewCustomer;

/