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\_balance NUMBER;

BEGIN

-- Check if the from\_account has enough balance

SELECT balance INTO v\_balance

FROM accounts

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

FOR UPDATE;

IF v\_balance < p\_amount THEN

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

END IF;

-- Deduct the amount from the source account

UPDATE accounts

SET balance = balance - p\_amount

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

-- Add the amount to the destination account

UPDATE accounts

SET balance = balance + p\_amount

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

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from account ' || p\_from\_account\_id || ' to account ' || p\_to\_account\_id);

EXCEPTION

WHEN OTHERS THEN

-- Rollback in case of any error

ROLLBACK;

-- Log the error message

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

-- Optionally, log the error to an error log table

-- INSERT INTO error\_log (error\_message) VALUES (SQLERRM);

END SafeTransferFunds;

/

Scenario 2: Manage Errors When Updating Employee Salaries

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_current\_salary NUMBER;

BEGIN

-- Attempt to select the current salary for the employee

SELECT salary INTO v\_current\_salary

FROM employees

WHERE employee\_id = p\_employee\_id

FOR UPDATE;

-- Update the salary by the given percentage

UPDATE employees

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

WHERE employee\_id = p\_employee\_id;

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for employee ID: ' || p\_employee\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

-- Rollback in case of any other error

ROLLBACK;

-- Log the error message

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

-- Optionally, log the error to an error log table

-- INSERT INTO error\_log (error\_message) VALUES (SQLERRM);

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

) AS

BEGIN

-- Attempt to insert a new customer

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

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

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added with ID: ' || p\_customer\_id);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

-- Handle duplicate customer ID

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

-- Optionally, log the error to an error log table

-- INSERT INTO error\_log (error\_message) VALUES ('Duplicate customer ID: ' || p\_customer\_id);

WHEN OTHERS THEN

-- Rollback in case of any other error

ROLLBACK;

-- Log the error message

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

-- Optionally, log the error to an error log table

-- INSERT INTO error\_log (error\_message) VALUES (SQLERRM);

END AddNewCustomer;