Scenario 1: Handle exceptions during fund transfers between accounts

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account\_id NUMBER,

p\_to\_account\_id NUMBER,

p\_amount NUMBER

) AS

BEGIN

BEGIN TRANSACTION;

-- Check if the from account has sufficient funds

IF (SELECT balance FROM accounts WHERE account\_id = p\_from\_account\_id) < p\_amount THEN

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

END IF;

-- Debit the from account

UPDATE accounts

SET balance = balance - p\_amount

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

-- Credit the to account

UPDATE accounts

SET balance = balance + p\_amount

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

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

Scenario 2: Manage errors when updating employee salaries

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id NUMBER,

p\_percentage NUMBER

) AS

BEGIN

BEGIN

UPDATE employees

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

WHERE 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

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

END;

END UpdateSalary;

Scenario 3: Ensure data integrity when adding a new customer

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_address VARCHAR2

) AS

BEGIN

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

END AddNewCustomer;