**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 NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

BEGIN

BEGIN

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id;

SELECT Balance INTO v\_to\_balance FROM Accounts WHERE AccountID = p\_to\_account\_id;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds for transfer.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer completed successfully.');

EXCEPTION

WHEN OTHERS THEN

-- Rollback transaction in case of error

ROLLBACK;

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

END;

END;

/

* Test Case:

BEGIN

SafeTransferFunds(

p\_from\_account\_id => 1, -- Example from account ID

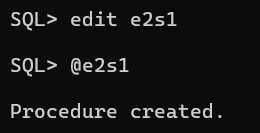
p\_to\_account\_id => 2, -- Example to account ID

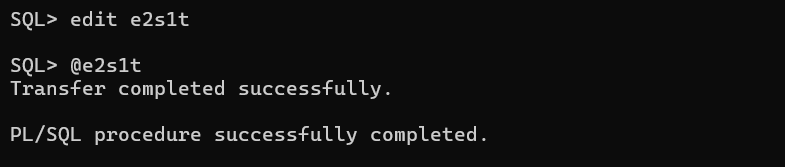
p\_amount => 100 -- Example amount to transfer

);

END;

/





**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 NUMBER,

p\_percentage IN NUMBER

) AS

v\_current\_salary NUMBER;

BEGIN

BEGIN

SELECT Salary INTO v\_current\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

/

* Test Case:

BEGIN

UpdateSalary(

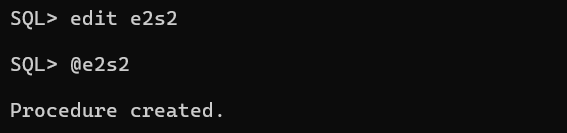
p\_employee\_id => 1, -- Example employee ID

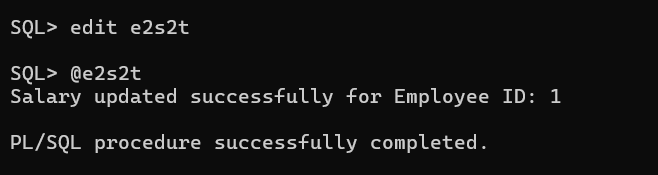
p\_percentage => 10 -- Example percentage increase

);

END;

/





**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_last\_modified IN DATE

) AS

BEGIN

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer with this ID already exists.');

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

/

* Test Case:

BEGIN

AddNewCustomer(

p\_customer\_id => 1001,

p\_name => 'Alice Williams',

p\_dob => TO\_DATE('1990-01-01', 'YYYY-MM-DD'),

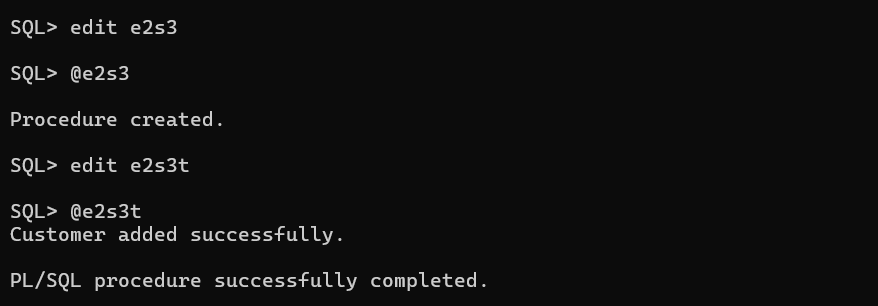
p\_balance => 1500,

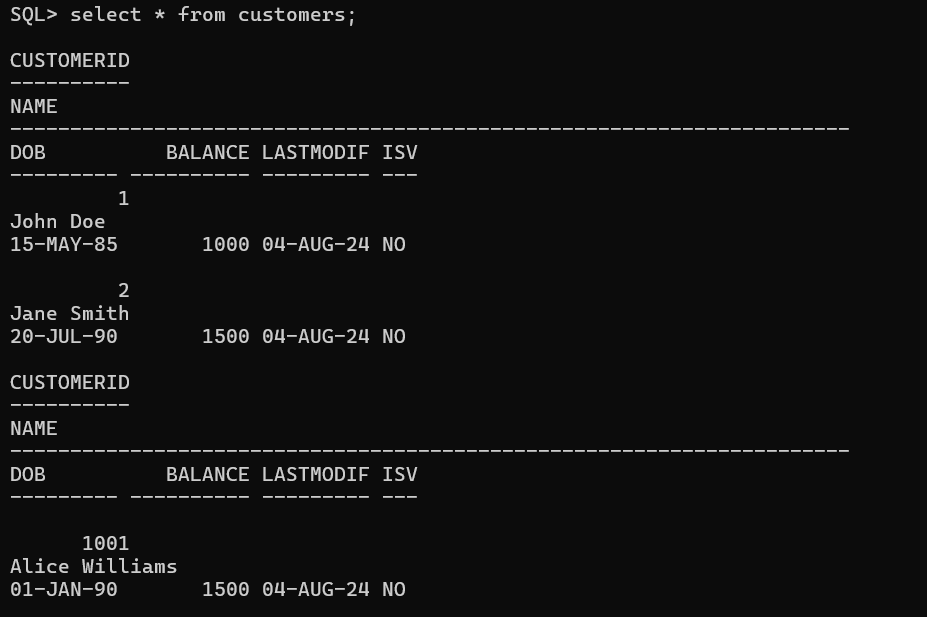
p\_last\_modified => SYSDATE

);

END;

**/**

****

****