**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 IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

AS

v\_balance NUMBER;

insufficient EXCEPTION; -- fixed spelling

BEGIN

-- Step 1: Lock and check source account balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account

FOR UPDATE;

-- Step 2: Check for sufficient balance

IF v\_balance < p\_amount THEN

RAISE insufficient;

END IF;

-- Step 3: Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

-- Step 4: Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

-- Step 5: Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_amount || ' from Account ' ||

p\_from\_account || ' to Account ' || p\_to\_account || ' successful.');

EXCEPTION

WHEN insufficient THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds');

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

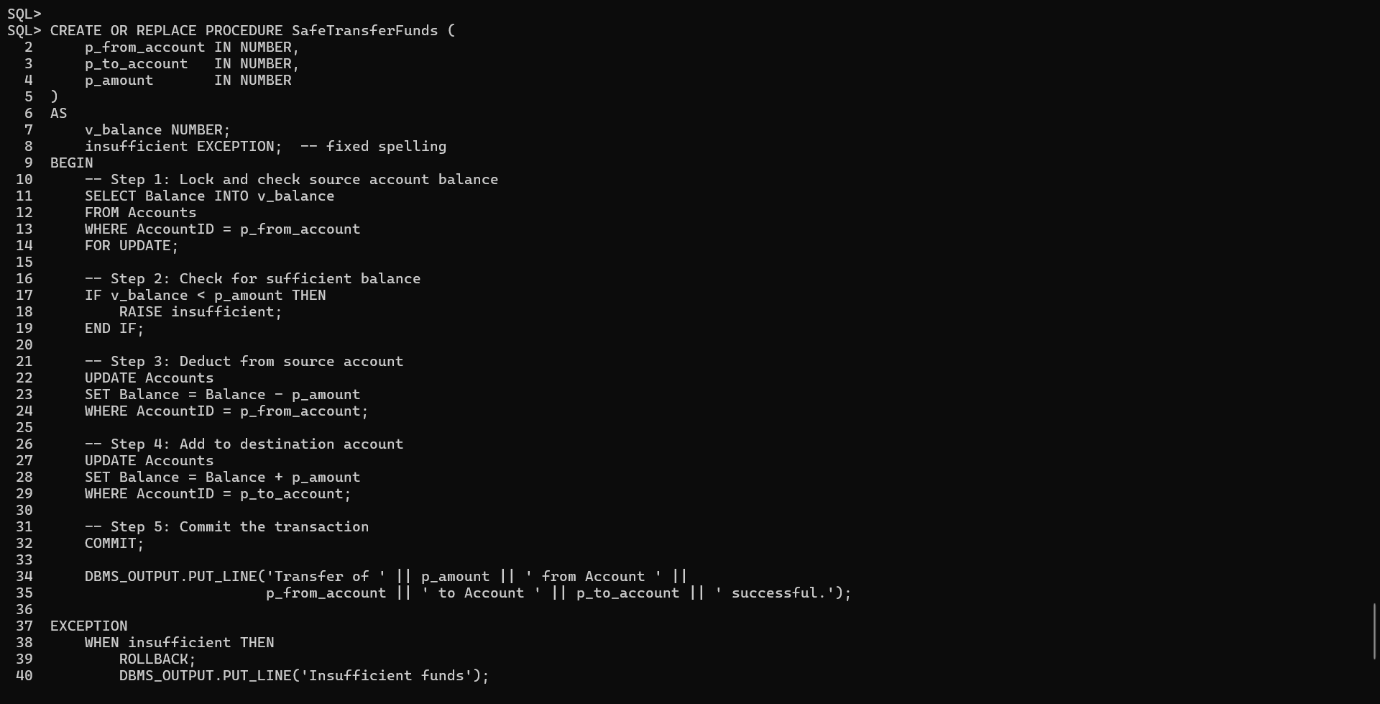
BEGIN

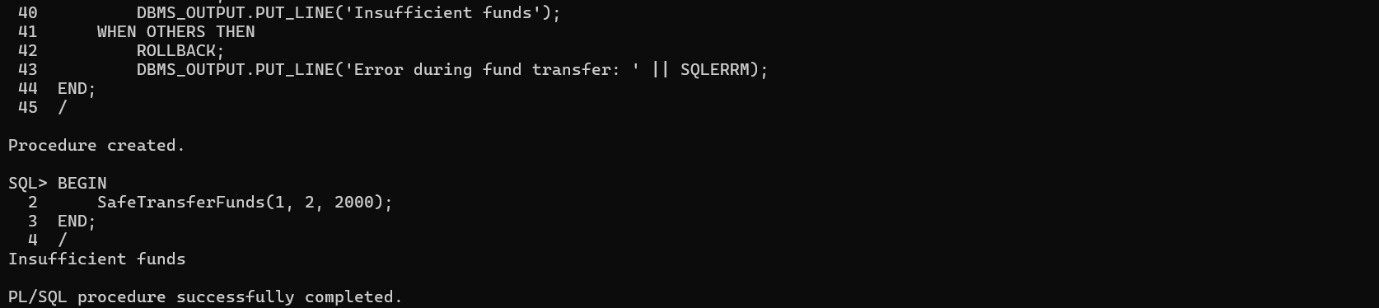
SafeTransferFunds(1, 2, 2000);

END;

/

Output:-





**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 GetCustomerName (

p\_customer\_id IN NUMBER

)

AS

v\_name Customers.Name%TYPE;

BEGIN

-- Try to fetch the name for the given CustomerID

SELECT Name INTO v\_name

FROM Customers

WHERE CustomerID = p\_customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_name);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No customer found with ID: ' || p\_customer\_id);

WHEN OTHERS THEN

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

END;

/

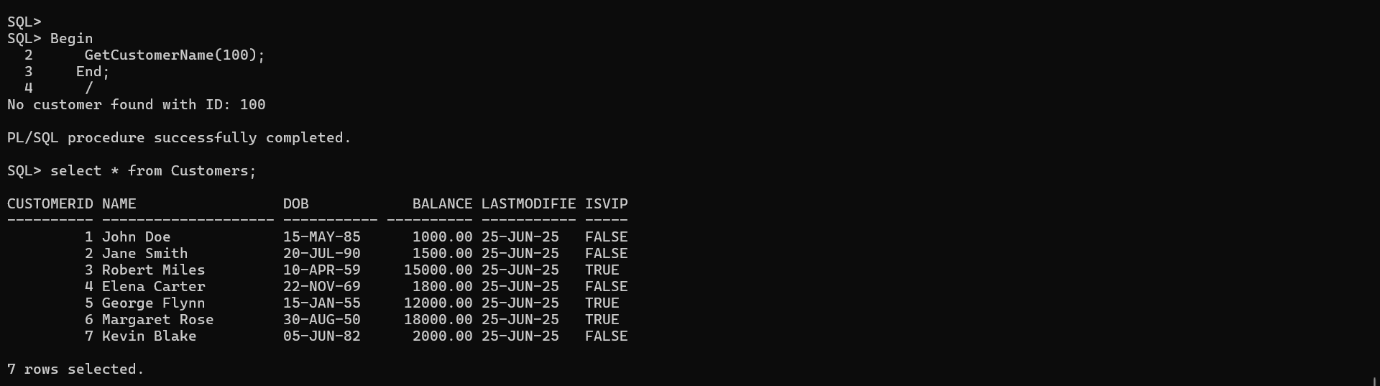
Begin

GetCustomerName(100);

End;

/

**Output :-**



**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.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

)

AS

dup\_customer EXCEPTION;

PRAGMA EXCEPTION\_INIT(dup\_customer, -00001); -- ORA-00001: unique constraint violated

BEGIN

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

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

DBMS\_OUTPUT.PUT\_LINE('Customer inserted successfully: ' || p\_name);

EXCEPTION

WHEN dup\_customer THEN

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

WHEN OTHERS THEN

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

END;

/

**Output :-**

