**Exercise 1: Control Structures**

**Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

age NUMBER,

loan\_interest NUMBER(5,2)

);

INSERT INTO customers VALUES (1, 'Alice', 65, 8.5);

INSERT INTO customers VALUES (2, 'Bob', 45, 9.0);

INSERT INTO customers VALUES (3, 'Carol', 70, 7.5);

INSERT INTO customers VALUES (4, 'David', 30, 10.0);

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (SELECT customer\_id, age, loan\_interest FROM customers) LOOP

IF rec.age > 60 THEN

UPDATE customers

SET loan\_interest = loan\_interest - (loan\_interest \* 0.01)

WHERE customer\_id = rec.customer\_id;

END IF;

END LOOP;

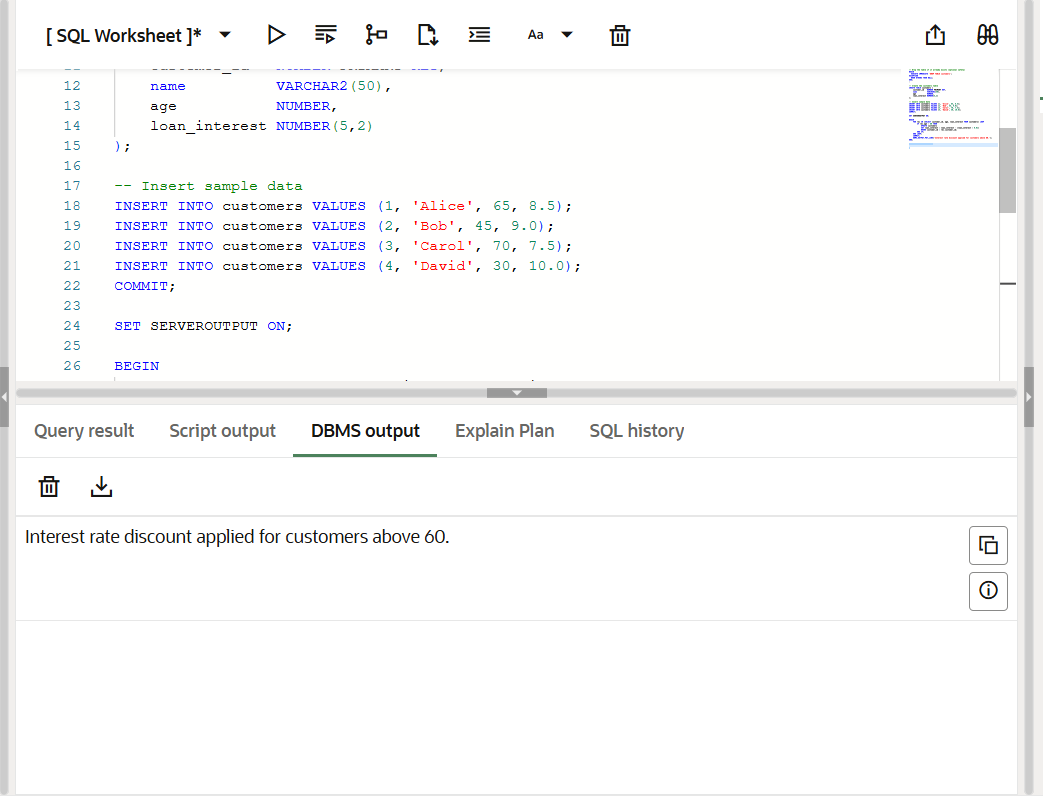
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest rate discount applied for customers above 60.');

END;

SELECT \* FROM customers;

**Output :**



**Scenario 2: A customer can be promoted to VIP status based on their balance**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER,

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

INSERT INTO Customers VALUES (1, 'Alice', 12000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 8000, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 15000, 'FALSE');

INSERT INTO Customers VALUES (4, 'David', 9500, 'FALSE');

INSERT INTO Customers VALUES (5, 'Eva', 20000, 'FALSE');

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, CustomerName, Balance FROM Customers;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || cust\_rec.CustomerName ||

' (ID: ' || cust\_rec.CustomerID ||

') promoted to VIP.');

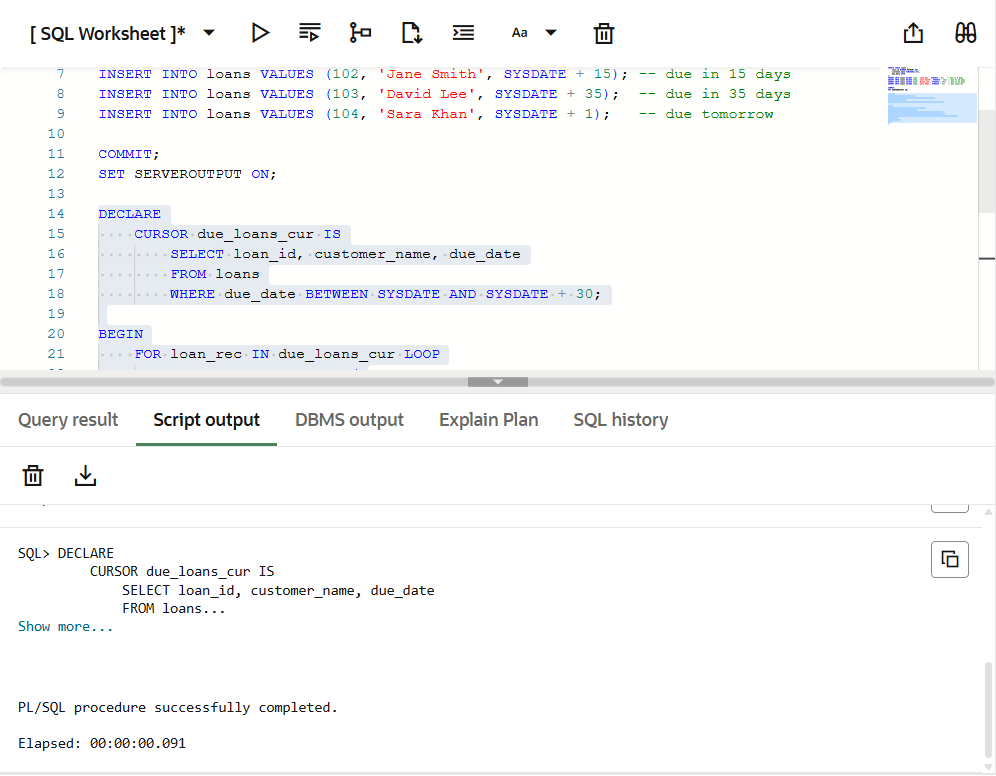
END IF;

END LOOP;

COMMIT;

END;

**Output :**



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

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(100),

due\_date DATE

);

INSERT INTO loans VALUES (101, 'Alice', SYSDATE + 5); -- due in 5 days

INSERT INTO loans VALUES (102, 'Jane Smith', SYSDATE + 15); -- due in 15 days

INSERT INTO loans VALUES (103, 'David Lee', SYSDATE + 35); -- due in 35 days

INSERT INTO loans VALUES (104, 'Sara Khan', SYSDATE + 1); -- due tomorrow

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR due\_loans\_cur IS

SELECT loan\_id, customer\_name, due\_date

FROM loans

WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan\_rec IN due\_loans\_cur LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || loan\_rec.loan\_id ||

' for customer ' || loan\_rec.customer\_name ||

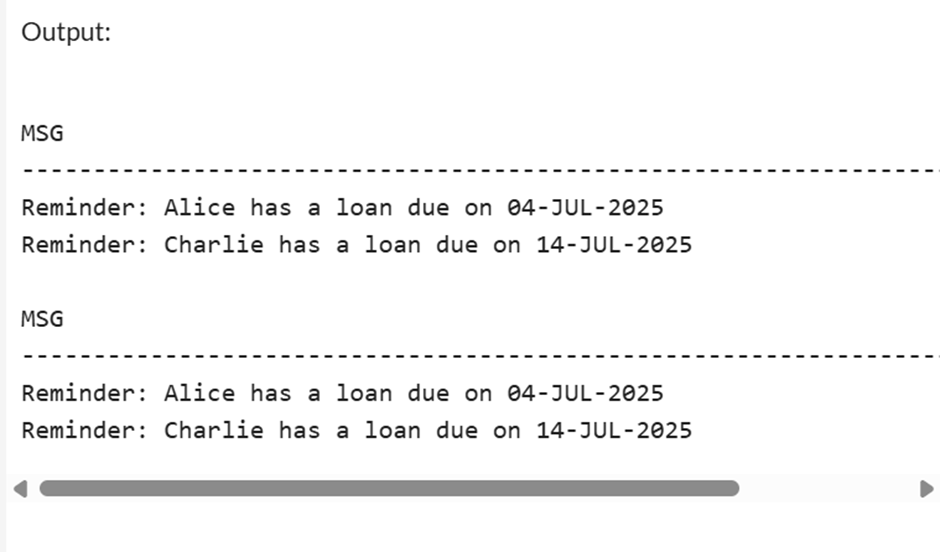
' is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY')

);

END LOOP;

END;

**Output :**



**Exercise 3: Stored Procedures**

**Scenario 1: The bank needs to process monthly interest for all savings accounts.**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

CREATE TABLE ACCOUNTS (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

/

INSERT INTO ACCOUNTS VALUES (101, 1, 'SAVINGS', 1000);

INSERT INTO ACCOUNTS VALUES (102, 1, 'CHECKING', 500);

INSERT INTO ACCOUNTS VALUES (103, 2, 'SAVINGS', 3000);

INSERT INTO ACCOUNTS VALUES (104, 3, 'SAVINGS', 700);

COMMIT;

/

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE ACCOUNTS

SET Balance = Balance + (Balance \* 0.01)

WHERE UPPER(AccountType) = 'SAVINGS';

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' savings accounts updated with interest.');

COMMIT;

END;

/

BEGIN

DBMS\_OUTPUT.ENABLE;

END;

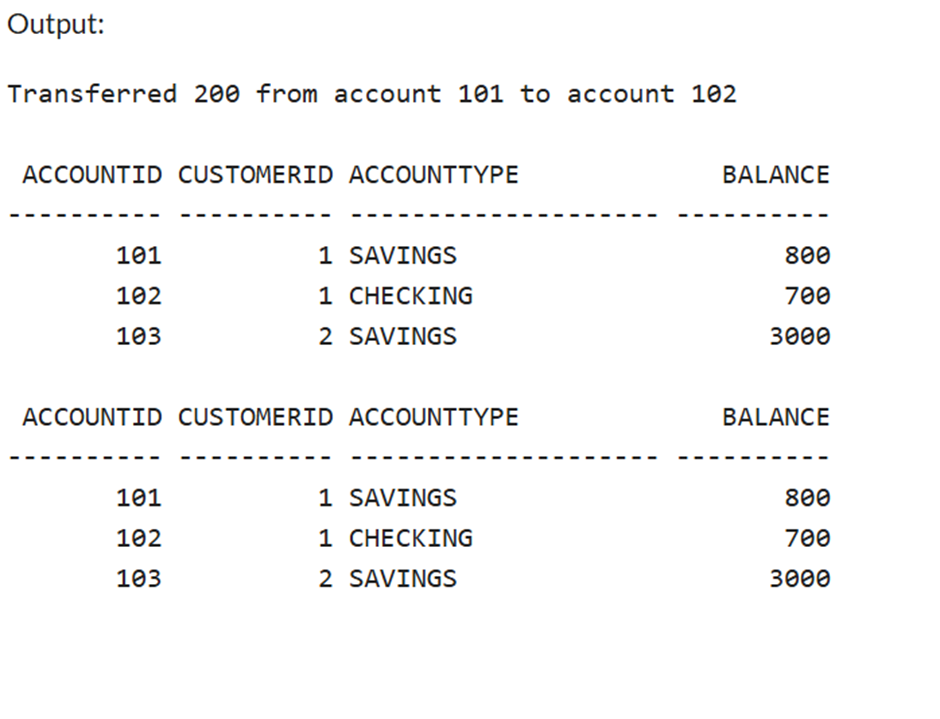
/

BEGIN

ProcessMonthlyInterest;

END;

/



**Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_FromBalance

FROM ACCOUNTS

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

IF v\_FromBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

UPDATE ACCOUNTS

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

UPDATE ACCOUNTS

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

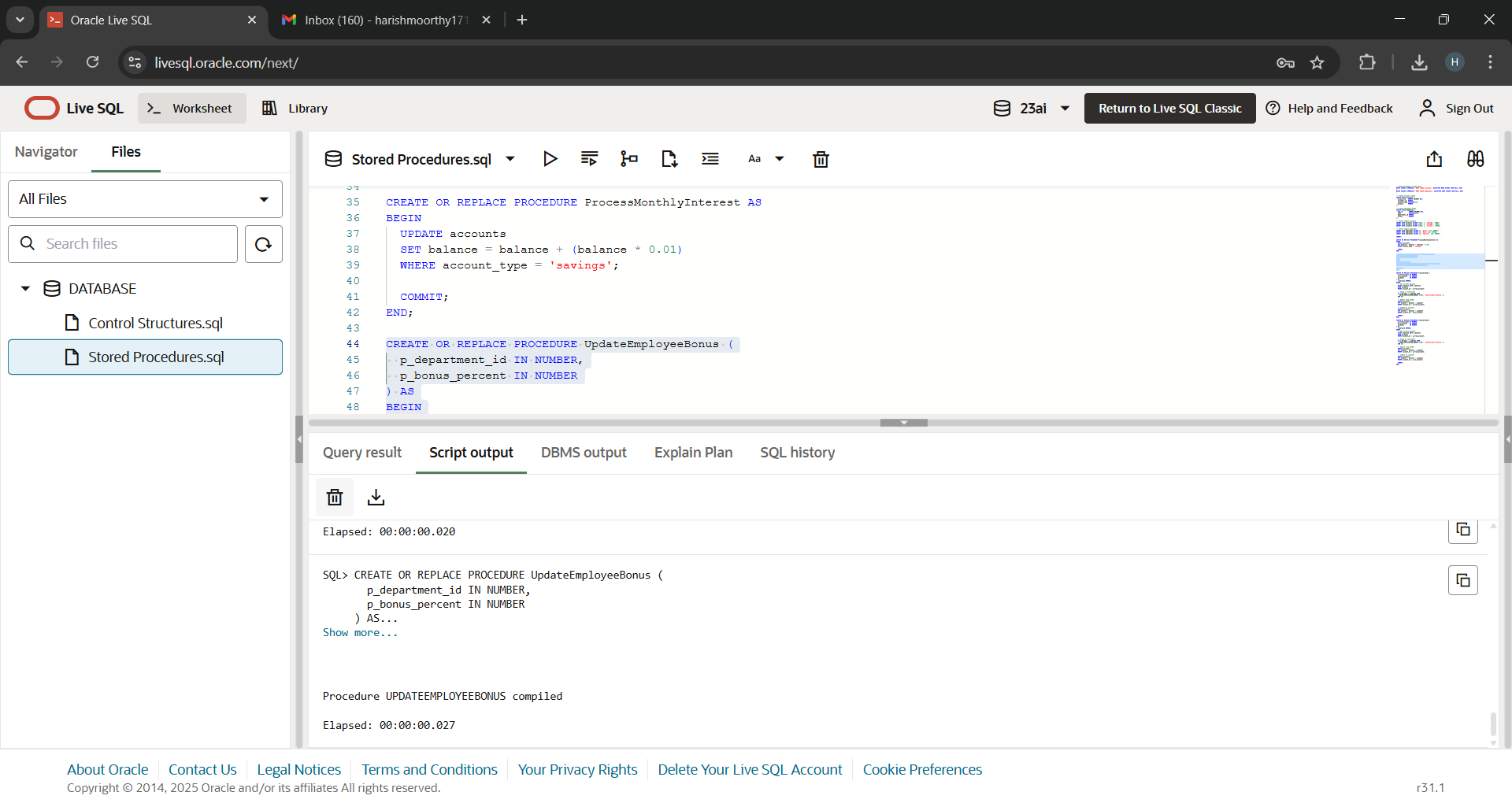
DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_Amount || ' from account ' || p\_FromAccountID || ' to account ' || p\_ToAccountID);

COMMIT;

END;

/

**Output :**



**Scenario 3: Customers should be able to transfer funds between their accounts.**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_FromBalance

FROM ACCOUNTS

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

IF v\_FromBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

UPDATE ACCOUNTS

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

UPDATE ACCOUNTS

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_Amount || ' from account ' || p\_FromAccountID || ' to account ' || p\_ToAccountID);

COMMIT;

END;

/

**Output :**

