**⇒ Mandatory Hands-On**

**Exercise 1: Control Structures**

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

SET SERVEROUTPUT ON;

DECLARE

CURSOR cust\_cursor IS

SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID;

v\_CustomerID Customers.CustomerID%TYPE;

v\_DOB Customers.DOB%TYPE;

v\_LoanID Loans.LoanID%TYPE;

v\_InterestRate Loans.InterestRate%TYPE;

v\_Age NUMBER;

BEGIN

FOR rec IN cust\_cursor LOOP

v\_CustomerID := rec.CustomerID;

v\_DOB := rec.DOB;

v\_LoanID := rec.LoanID;

v\_InterestRate := rec.InterestRate;

v\_Age := FLOOR(MONTHS\_BETWEEN(SYSDATE, v\_DOB) / 12);

IF v\_Age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = v\_LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied for Customer ID: ' || v\_CustomerID || ' | Age: ' || v\_Age);

ELSE

DBMS\_OUTPUT.PUT\_LINE('No discount for Customer ID: ' || v\_CustomerID || ' | Age: ' || v\_Age);

END IF;

END LOOP;

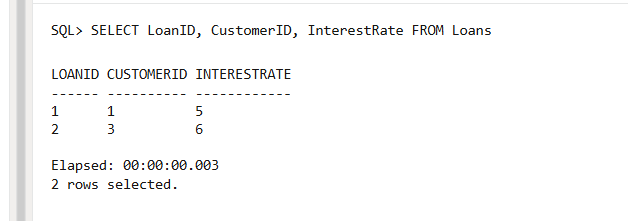
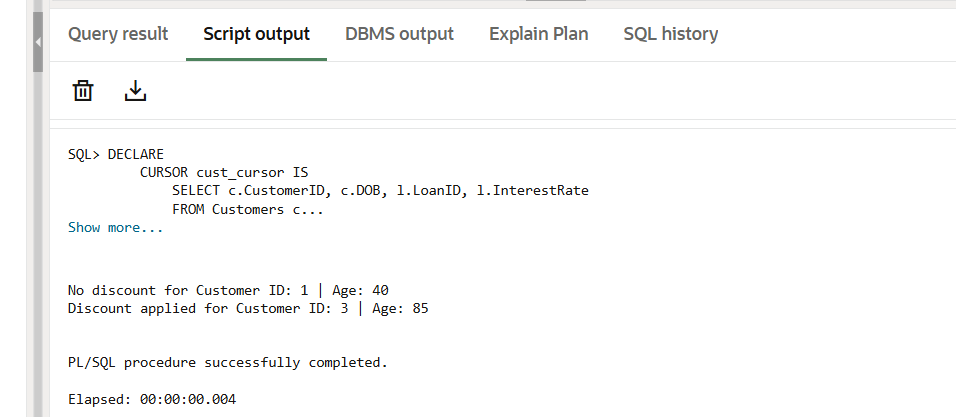
COMMIT;

END;

/

SELECT LoanID, CustomerID, InterestRate FROM Loans;

Result:



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

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

UPDATE Customers SET IsVIP = 'FALSE';

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID, Balance FROM Customers;

v\_CustomerID Customers.CustomerID%TYPE;

v\_Balance Customers.Balance%TYPE;

BEGIN

FOR rec IN vip\_cursor LOOP

v\_CustomerID := rec.CustomerID;

v\_Balance := rec.Balance;

IF v\_Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = v\_CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Marked VIP: Customer ID = ' || v\_CustomerID || ' | Balance = ' || v\_Balance);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Not VIP: Customer ID = ' || v\_CustomerID || ' | Balance = ' || v\_Balance);

END IF;

END LOOP;

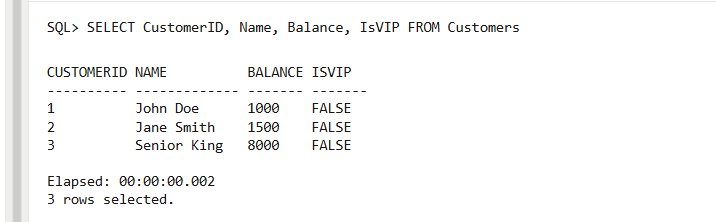
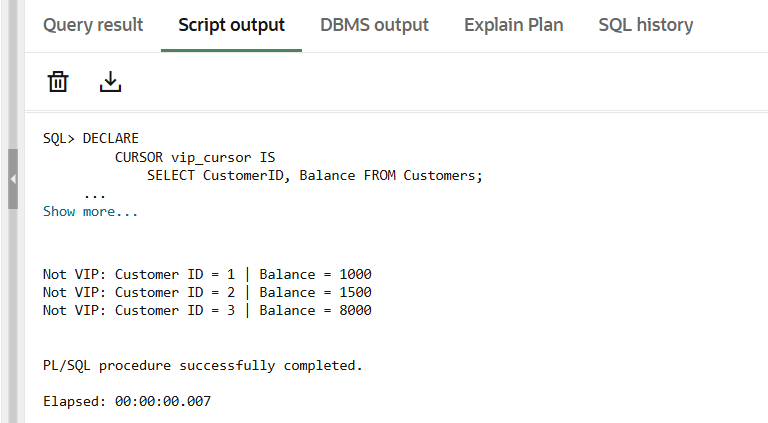
COMMIT;

END;

/

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

Result:



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

SET SERVEROUTPUT ON;

DECLARE

CURSOR loan\_cursor IS

SELECT CustomerID, EndDate

FROM Loans;

v\_CustomerID Loans.CustomerID%TYPE;

v\_EndDate Loans.EndDate%TYPE;

BEGIN

FOR rec IN loan\_cursor LOOP

v\_CustomerID := rec.CustomerID;

v\_EndDate := rec.EndDate;

IF v\_EndDate <= SYSDATE + 30 THEN

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ID ' || v\_CustomerID || ' has a loan due on ' || TO\_CHAR(v\_EndDate, 'YYYY-MM-DD'));

ELSE

DBMS\_OUTPUT.PUT\_LINE('No reminder: Customer ID ' || v\_CustomerID || ' | Due on ' || TO\_CHAR(v\_EndDate, 'YYYY-MM-DD'));

END IF;

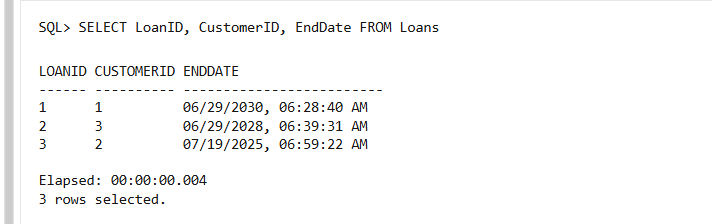
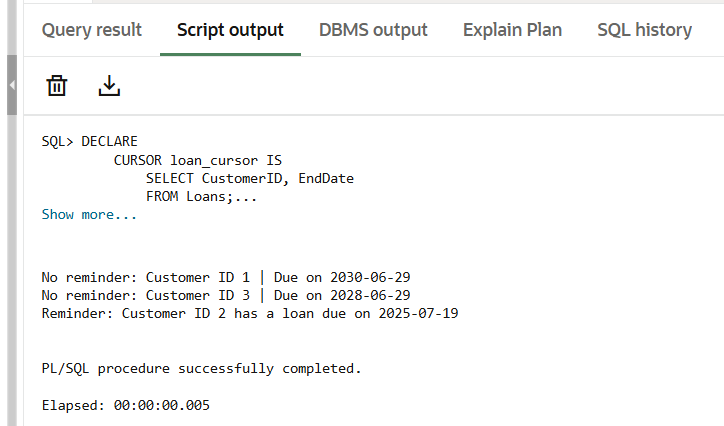
END LOOP;

END;

/

SELECT LoanID, CustomerID, EndDate FROM Loans;

Result:



**Exercise 3: Stored Procedures**

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

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID);

END LOOP;

COMMIT;

END;

/

SET SERVEROUTPUT ON;

BEGIN

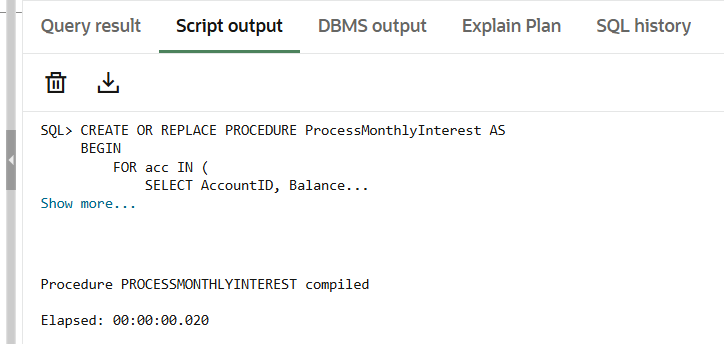
ProcessMonthlyInterest;

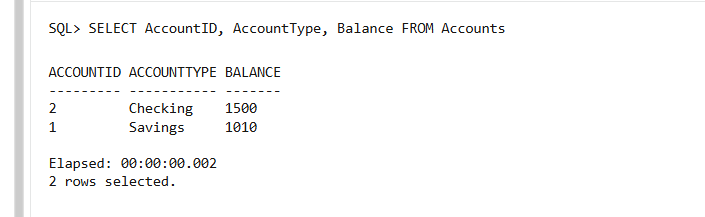
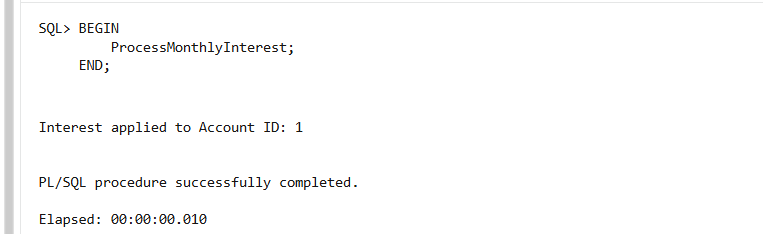
END;

/

SELECT AccountID, AccountType, Balance FROM Accounts;

Result:





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

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) AS

BEGIN

FOR emp IN (

SELECT EmployeeID, Salary

FROM Employees

WHERE Department = p\_Department

) LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* (p\_BonusPercent / 100))

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.EmployeeID);

END LOOP;

COMMIT;

END;

/

SET SERVEROUTPUT ON;

BEGIN

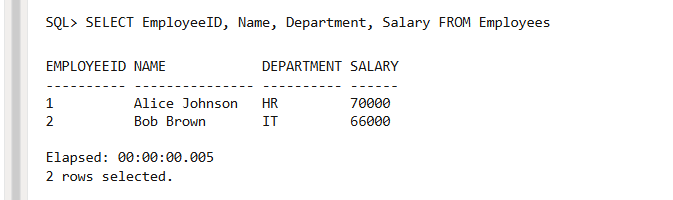
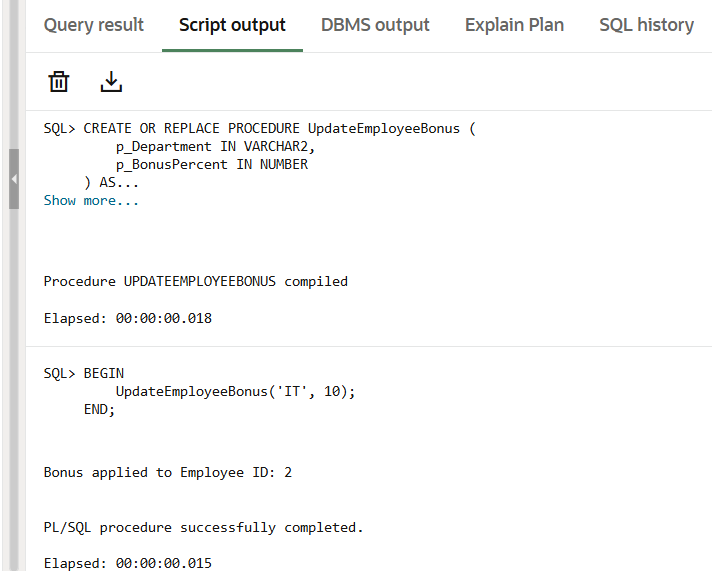
UpdateEmployeeBonus('IT', 10);

END;

/

SELECT EmployeeID, Name, Department, Salary FROM Employees;

Result:



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

) AS

v\_FromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

IF v\_FromBalance >= p\_Amount THEN

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;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ' || p\_FromAccountID);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('One of the accounts does not exist.');

WHEN OTHERS THEN

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

END;

/

SET SERVEROUTPUT ON;

BEGIN

TransferFunds(2, 1, 200);

END;

/

SELECT AccountID, Balance FROM Accounts;

Result:

