**PLSQL Exercises**

**Exercise 1: Control Structures**

## **Scenario1:**

The bank wants to apply a discount to loan interest rates for customers above 60 years old. Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

## **CODE:**

### BEGIN

FOR rec IN (SELECT CustomerID, Age, LoanInterestRate FROM Customers)

LOOP

IF rec.Age > 60 THEN

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - (LoanInterestRate \* 0.01)

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

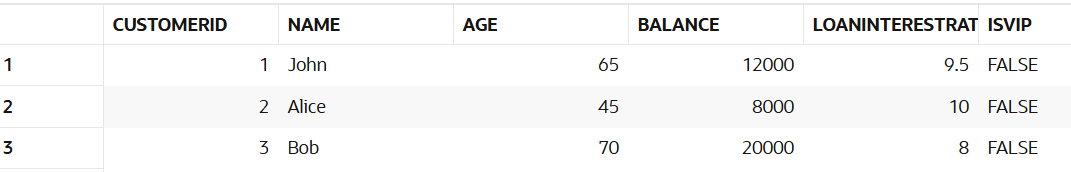
COMMIT;

END;

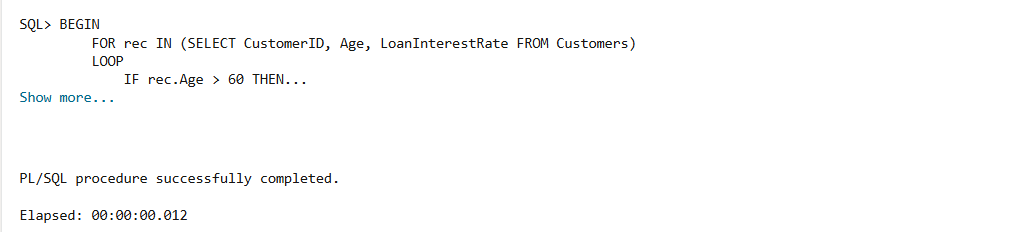
SELECT \* FROM Customers;

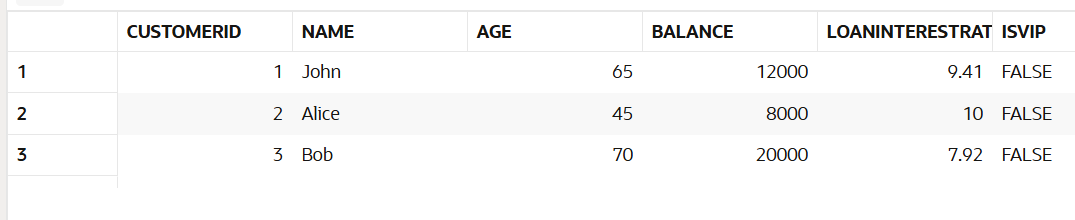
**OUTPUT:**

* **Before PL/SQL Block Execution:**

****

* **After PL/SQL Block Execution:**

****



## **Scenario2:**

A customer can be promoted to VIP status based on their balance. Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

## **CODE:**

### BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers)

LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

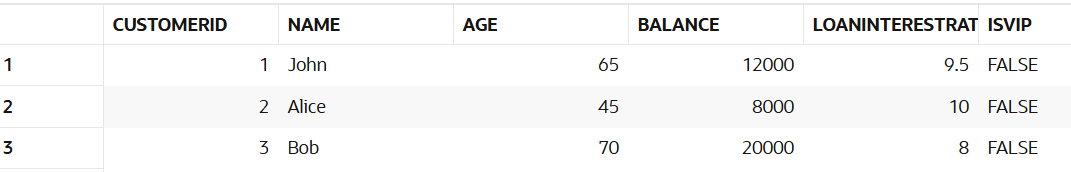
COMMIT;

END;

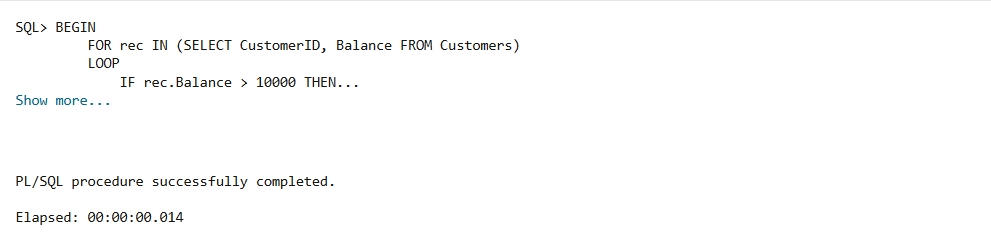
SELECT \* FROM Customers;

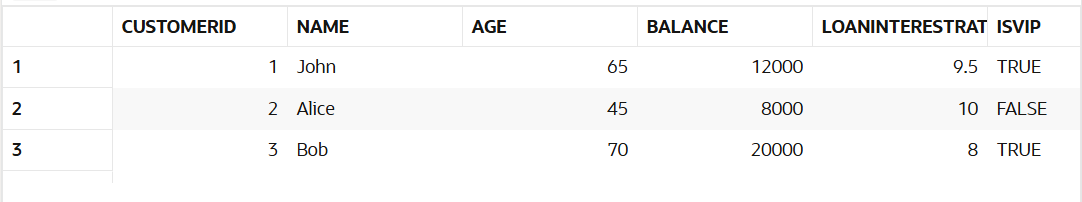
**OUTPUT:**

* **Before PL/SQL Block Execution:**

****

* **After PL/SQL Block Execution:**

****



## **Scenario3:**

The bank wants to send reminders to customers whose loans are due within the next 30 day. Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

## **CODE:**

DECLARE

    v\_due\_date DATE;

    v\_name     VARCHAR2(100);

BEGIN

    FOR rec IN (

        SELECT c.CustomerID, c.Name, l.DueDate

        FROM Customers c

        JOIN Loans l ON c.CustomerID = l.CustomerID

        WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

    )

    LOOP

        DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for ' || rec.Name ||

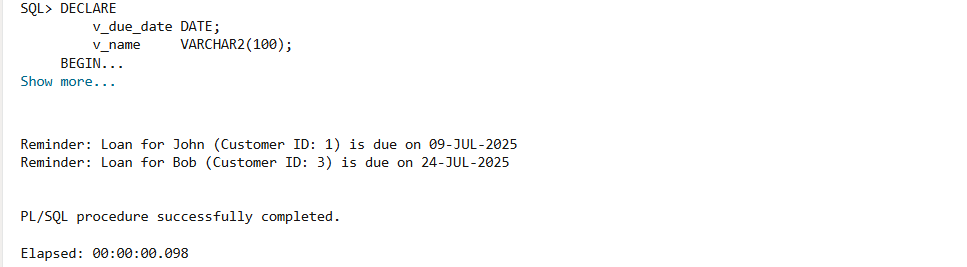
                             ' (Customer ID: ' || rec.CustomerID ||

                             ') is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY'));

    END LOOP;

END;

**OUTPUT:**



**Exercise 3: Stored Procedure**

## **Scenario1:**

The bank needs to process monthly interest for all savings accounts. Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

## **CODE:**

### 

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01); -- 1% interest

COMMIT;

END;

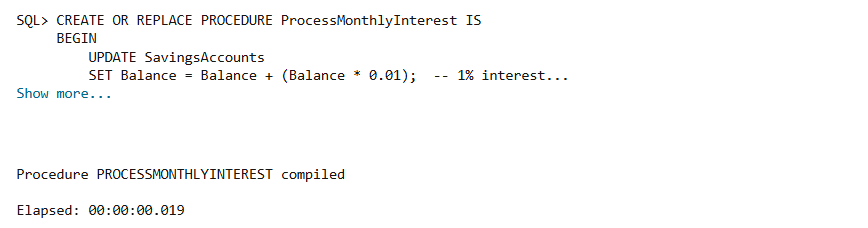
BEGIN

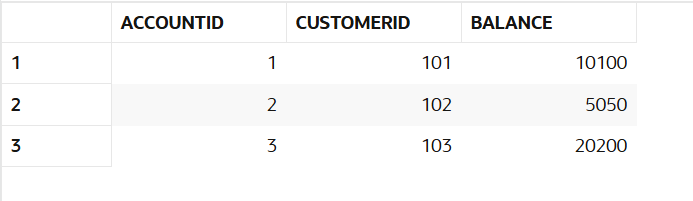
ProcessMonthlyInterest;

END;

SELECT \* FROM SavingsAccounts;

**OUTPUT:**

****



## **Scenario2:**

The bank wants to implement a bonus scheme for employees based on their performance. Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

## **CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_pct / 100)

WHERE Department = p\_department;

COMMIT;

END;

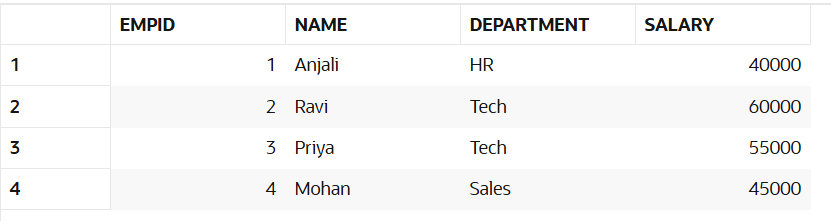
BEGIN

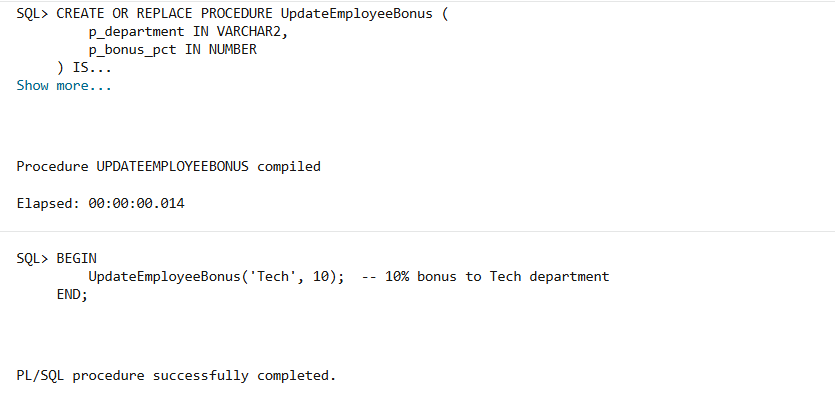
UpdateEmployeeBonus('Tech', 10); -- 10% bonus to Tech department

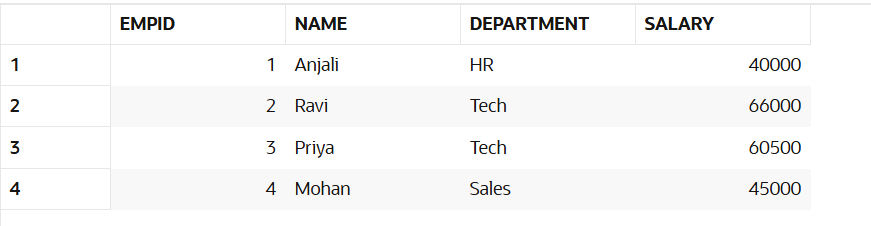
END;

SELECT \* FROM Employees;

**OUTPUT:**

* **Before PL/SQL Block Execution:**
* **After PL/SQL Block Execution:**





## **Scenario3:**

Customers should be able to transfer funds between their accounts. Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

## **CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_from\_account IN INT,

    p\_to\_account IN INT,

    p\_amount IN NUMBER

) IS

    v\_balance NUMBER;

BEGIN

    SELECT Balance INTO v\_balance

    FROM Accounts

    WHERE AccountID = p\_from\_account;

    IF v\_balance < p\_amount THEN

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

    ELSE

        UPDATE Accounts

        SET Balance = Balance - p\_amount

        WHERE AccountID = p\_from\_account;

        UPDATE Accounts

        SET Balance = Balance + p\_amount

        WHERE AccountID = p\_to\_account;

        COMMIT;

    END IF;

END;

BEGIN

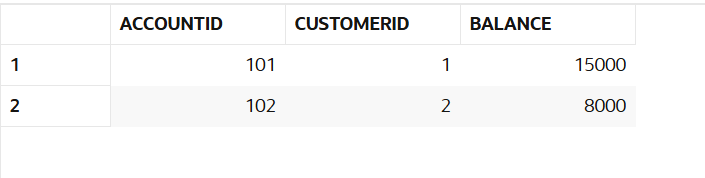
    TransferFunds(101, 102, 2000);  -- Transfer ₹2000 from Account 101 to 102

END;

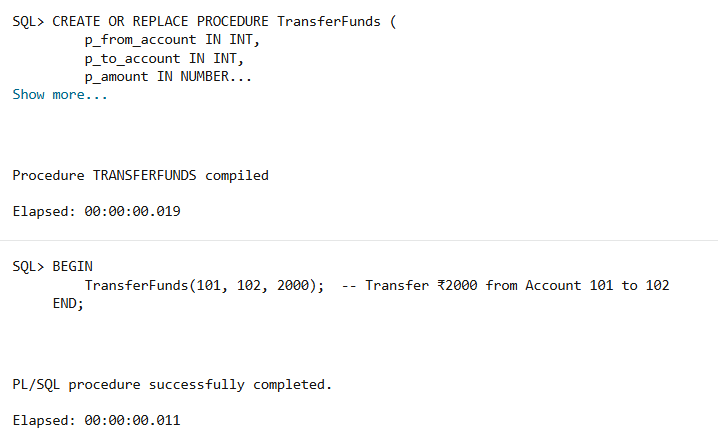
SELECT \* FROM Accounts;

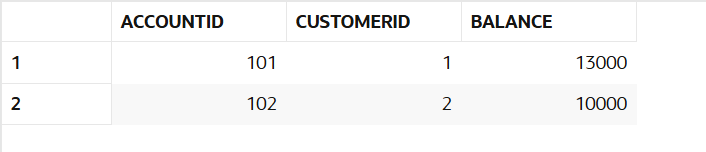
**OUTPUT:**

* **Before PL/SQL Block Execution:**

****

1. **After PL/SQL Block Execution:**

****

****