**WEEK-02**

**MODULE-03:PL/SQL PROGRAMMING**

**MANDATORY HANDSON**

**Exercise 1: Control Structures**

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

**Question: 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.**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'FALSE');

INSERT INTO Customers VALUES (3, 'Carol', 70, 15000, 'FALSE');

INSERT INTO Customers VALUES (4, 'David', 59, 6000, 'FALSE');

INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 15);

INSERT INTO Loans VALUES (102, 2, 9.0, SYSDATE + 45);

INSERT INTO Loans VALUES (103, 3, 7.0, SYSDATE + 5);

INSERT INTO Loans VALUES (104, 4, 8.0, SYSDATE + 29);

BEGIN

FOR rec IN (

SELECT l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE c.Age > 60

) LOOP

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

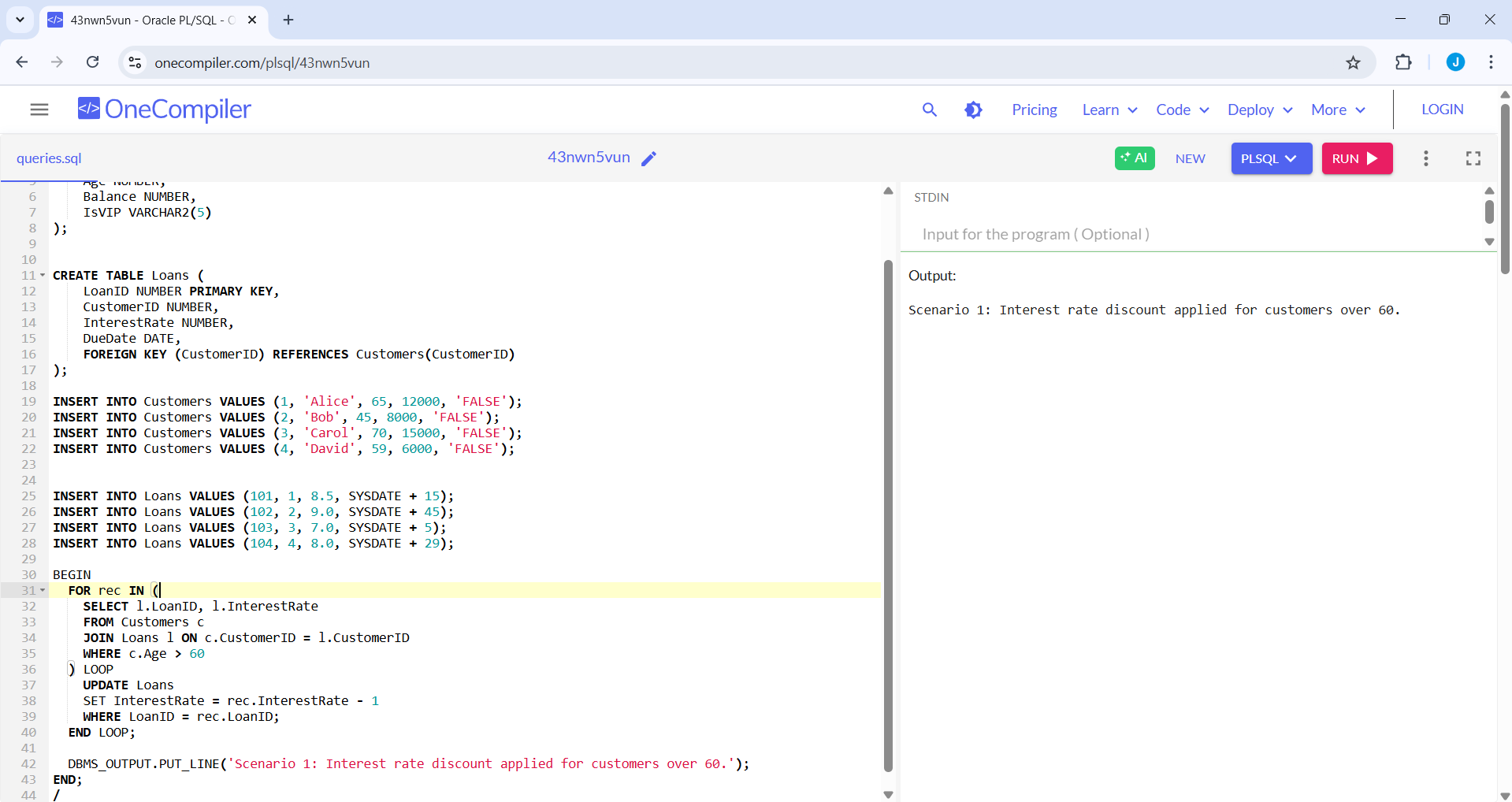
END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Scenario 1: Interest rate discount applied for customers over 60.');

END;

/

**OUTPUT**

****

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

**Question: 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**.

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'FALSE');

INSERT INTO Customers VALUES (3, 'Carol', 70, 15000, 'FALSE');

INSERT INTO Customers VALUES (4, 'David', 59, 6000, 'FALSE');

INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 15);

INSERT INTO Loans VALUES (102, 2, 9.0, SYSDATE + 45);

INSERT INTO Loans VALUES (103, 3, 7.0, SYSDATE + 5);

INSERT INTO Loans VALUES (104, 4, 8.0, SYSDATE + 29);

BEGIN

FOR rec IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

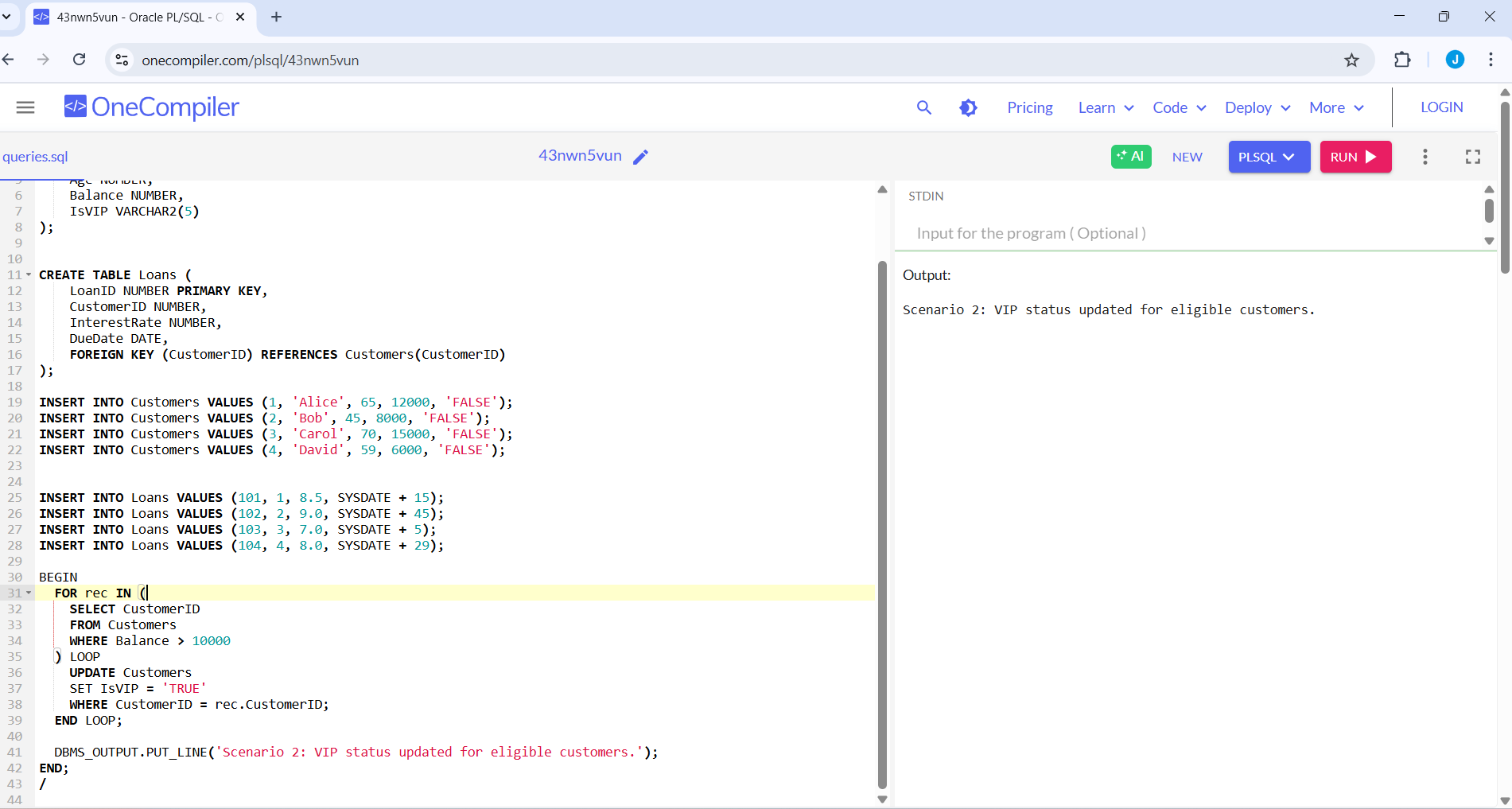
END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Scenario 2: VIP status updated for eligible customers.');

END;

/

**OUTPUT**

****

**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 TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'FALSE');

INSERT INTO Customers VALUES (3, 'Carol', 70, 15000, 'FALSE');

INSERT INTO Customers VALUES (4, 'David', 59, 6000, 'FALSE');

INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 15);

INSERT INTO Loans VALUES (102, 2, 9.0, SYSDATE + 45);

INSERT INTO Loans VALUES (103, 3, 7.0, SYSDATE + 5);

INSERT INTO Loans VALUES (104, 4, 8.0, SYSDATE + 29);

BEGIN

FOR rec IN (

SELECT l.LoanID, c.Name, l.DueDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Scenario 3 Reminder: Loan ID ' || rec.LoanID ||

' for ' || rec.Name ||

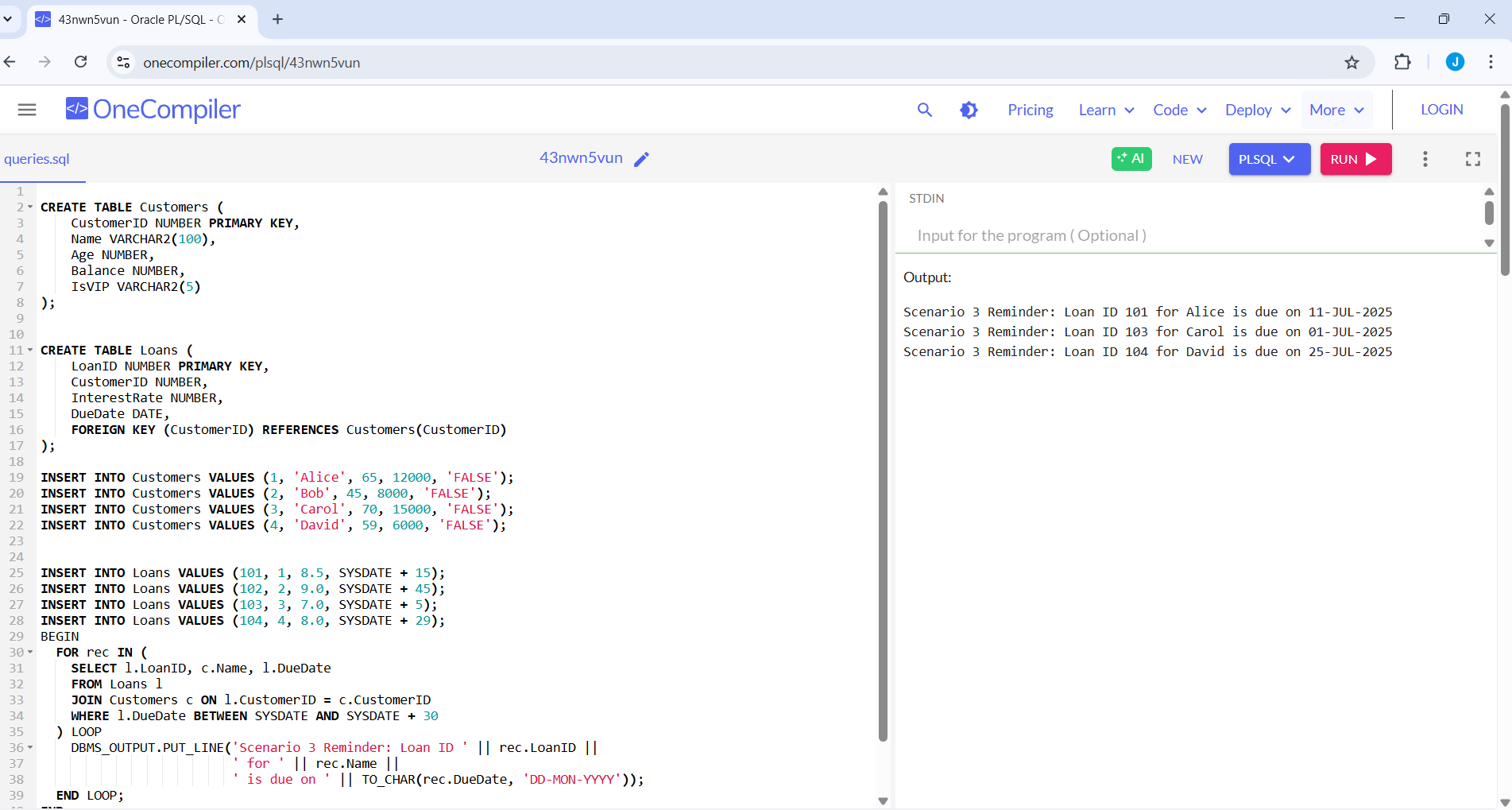
' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**OUTPUT**

****

**Exercise 3: Stored Procedures**

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

**Question: 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.**

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

CREATE TABLE Accounts (

AccountNumber NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER

);

INSERT INTO SavingsAccounts VALUES (101, 'Alice', 10000);

INSERT INTO SavingsAccounts VALUES (102, 'Bob', 15000);

INSERT INTO SavingsAccounts VALUES (103, 'Carol', 8000);

INSERT INTO Employees VALUES (1, 'John', 'HR', 50000);

INSERT INTO Employees VALUES (2, 'Mike', 'IT', 60000);

INSERT INTO Employees VALUES (3, 'Sara', 'HR', 55000);

INSERT INTO Accounts VALUES (201, 'Alice', 20000);

INSERT INTO Accounts VALUES (202, 'Bob', 15000);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM SavingsAccounts) LOOP

UPDATE SavingsAccounts

SET Balance = rec.Balance + (rec.Balance \* 0.01)

WHERE AccountID = rec.AccountID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

END;

/

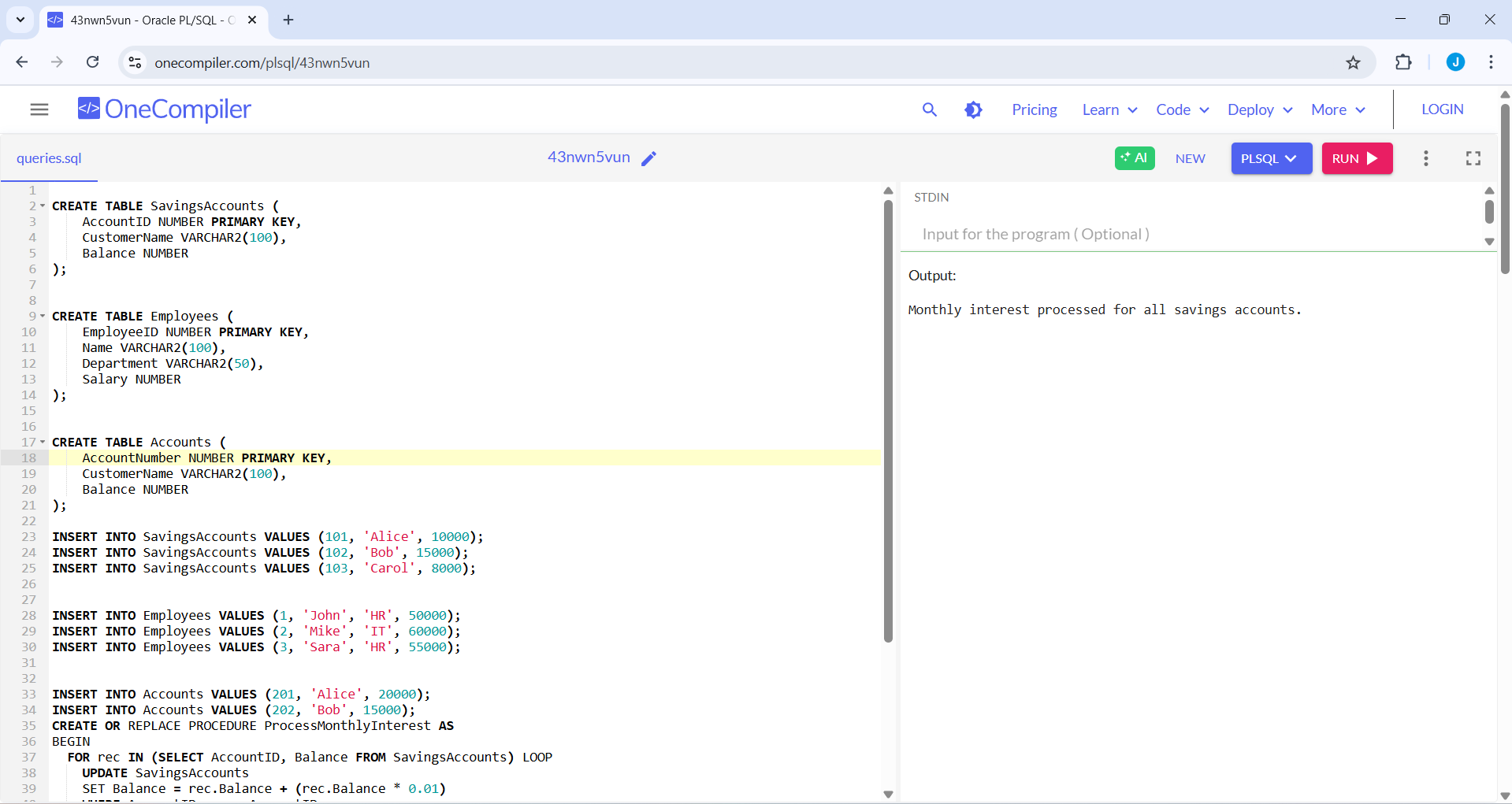
BEGIN

ProcessMonthlyInterest;

END;

/

**OUTPUT**

****

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

**Question: Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.**

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

CREATE TABLE Accounts (

AccountNumber NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER

);

INSERT INTO SavingsAccounts VALUES (101, 'Alice', 10000);

INSERT INTO SavingsAccounts VALUES (102, 'Bob', 15000);

INSERT INTO SavingsAccounts VALUES (103, 'Carol', 8000);

INSERT INTO Employees VALUES (1, 'John', 'HR', 50000);

INSERT INTO Employees VALUES (2, 'Mike', 'IT', 60000);

INSERT INTO Employees VALUES (3, 'Sara', 'HR', 55000);

INSERT INTO Accounts VALUES (201, 'Alice', 20000);

INSERT INTO Accounts VALUES (202, 'Bob', 15000);

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) AS

BEGIN

FOR rec IN (

SELECT EmployeeID, Salary FROM Employees

WHERE Department = p\_Department

) LOOP

UPDATE Employees

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

WHERE EmployeeID = rec.EmployeeID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || p\_Department);

END;

/

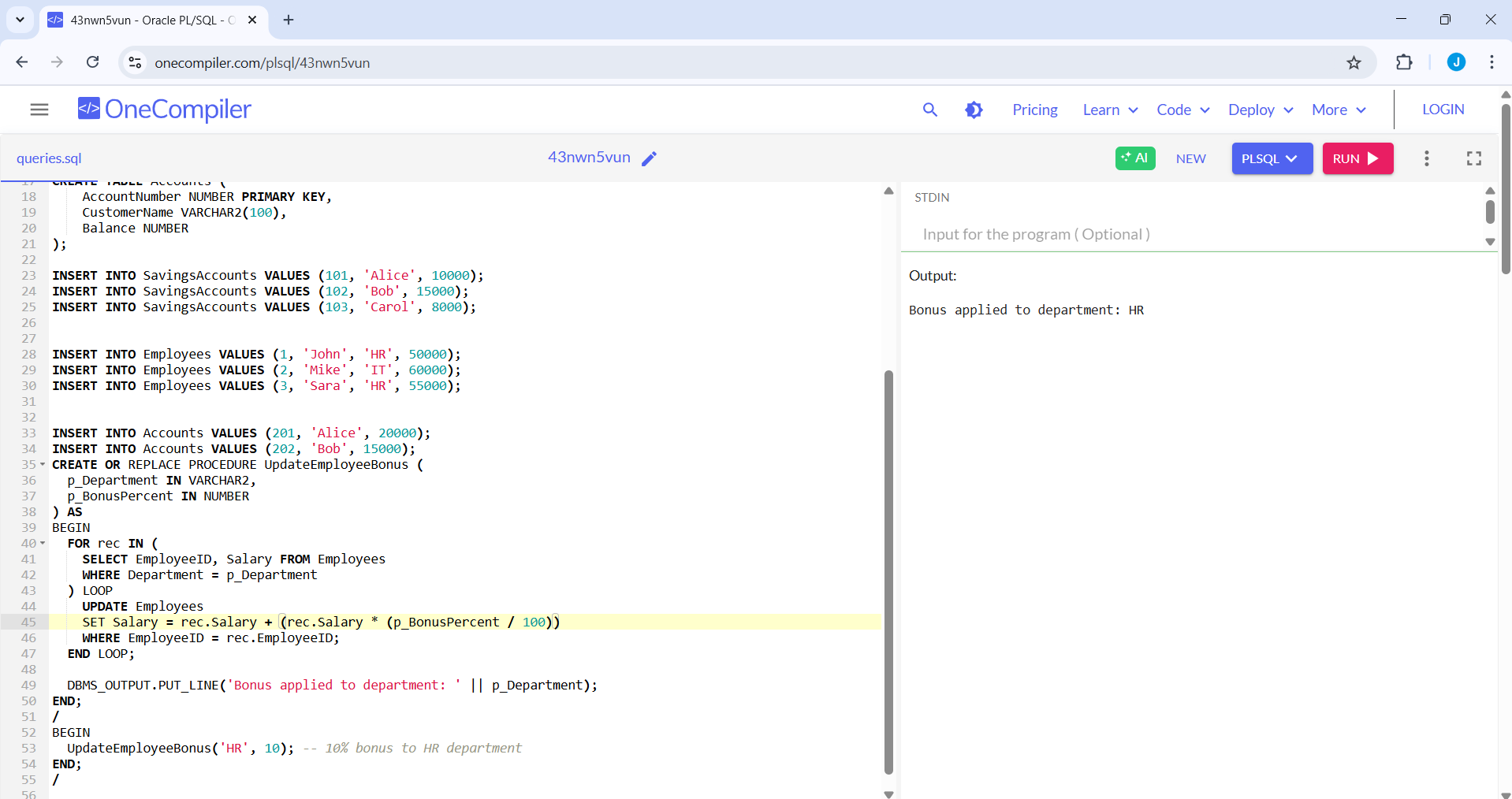
BEGIN

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

END;

/

**OUTPUT**

****

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

**Question: 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**

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

CREATE TABLE Accounts (

AccountNumber NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER

);

INSERT INTO SavingsAccounts VALUES (101, 'Alice', 10000);

INSERT INTO SavingsAccounts VALUES (102, 'Bob', 15000);

INSERT INTO SavingsAccounts VALUES (103, 'Carol', 8000);

INSERT INTO Employees VALUES (1, 'John', 'HR', 50000);

INSERT INTO Employees VALUES (2, 'Mike', 'IT', 60000);

INSERT INTO Employees VALUES (3, 'Sara', 'HR', 55000);

INSERT INTO Accounts VALUES (201, 'Alice', 20000);

INSERT INTO Accounts VALUES (202, 'Bob', 15000);

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAcc IN NUMBER,

p\_ToAcc IN NUMBER,

p\_Amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountNumber = p\_FromAcc;

IF v\_balance < p\_Amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in source account.');

ELSE

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountNumber = p\_FromAcc;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountNumber = p\_ToAcc;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Invalid account number.');

END;

/

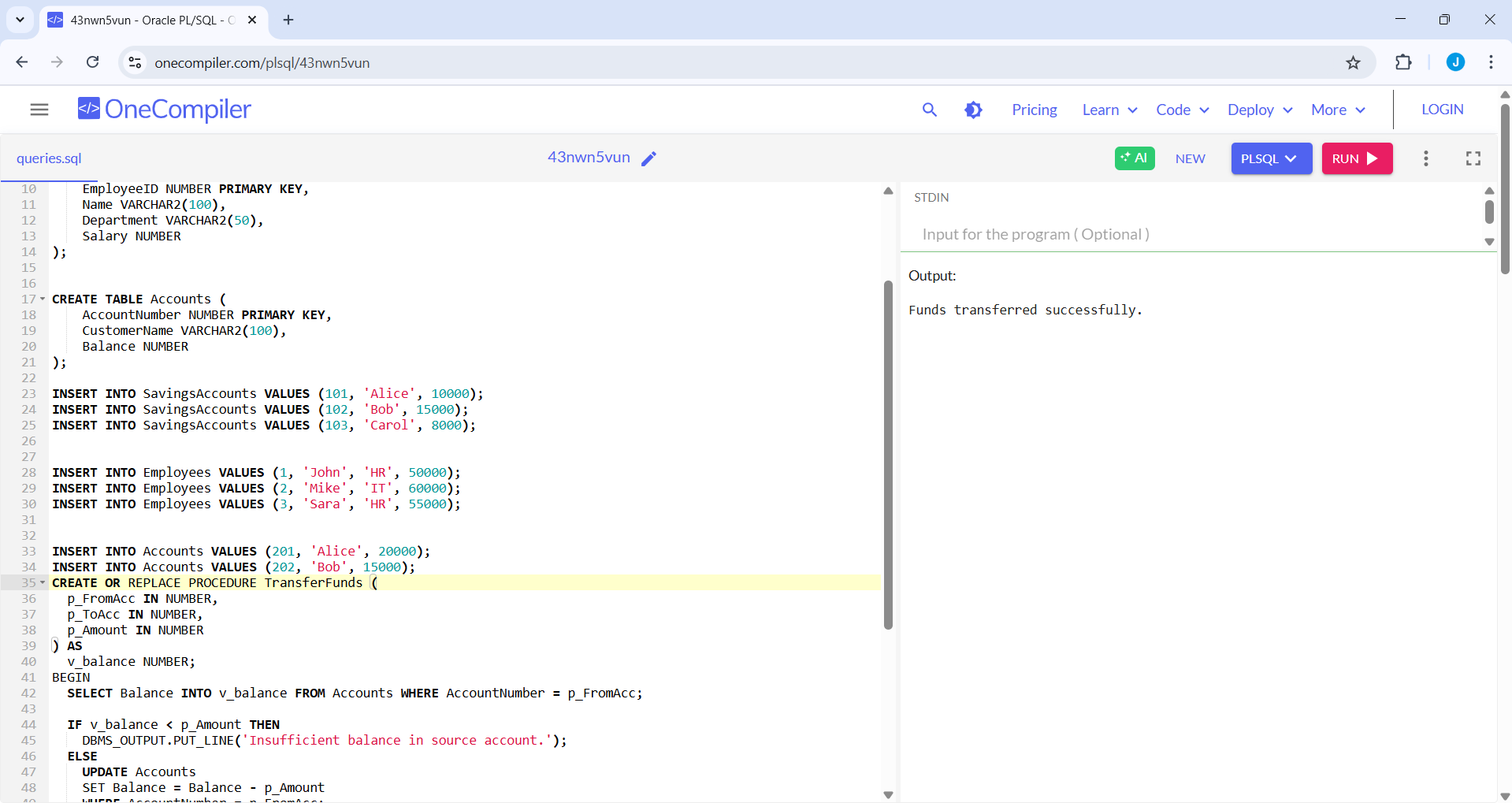
BEGIN

TransferFunds(201, 202, 5000);

END;

/

**OUTPUT**

****