**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.

**Code & Output:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Age NUMBER

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER(5,2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

--customer values

INSERT INTO Customers VALUES (1, 'Anusri', 65);

INSERT INTO Customers VALUES (2, 'Anusha', 45);

INSERT INTO Customers VALUES (3, 'Ranjan', 60);

-- loan values

INSERT INTO Loans VALUES (101, 1, 8.5);

INSERT INTO Loans VALUES (102, 2, 9.0);

INSERT INTO Loans VALUES (103, 3, 10.0);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- BEFORE DISCOUNT ---');

FOR loan IN (

SELECT l.LoanID, c.Name, c.Age, l.InterestRate

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('LoanID: ' || loan.LoanID ||

', Customer: ' || loan.Name ||

', Age: ' || loan.Age ||

', Interest: ' || loan.InterestRate);

END LOOP;

FOR cust IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE c.Age > 60

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- AFTER DISCOUNT ---');

FOR loan IN (

SELECT l.LoanID, c.Name, c.Age, l.InterestRate

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID

) LOOP

DBMS\_OUTPUT.PUT\_LINE('LoanID: ' || loan.LoanID ||

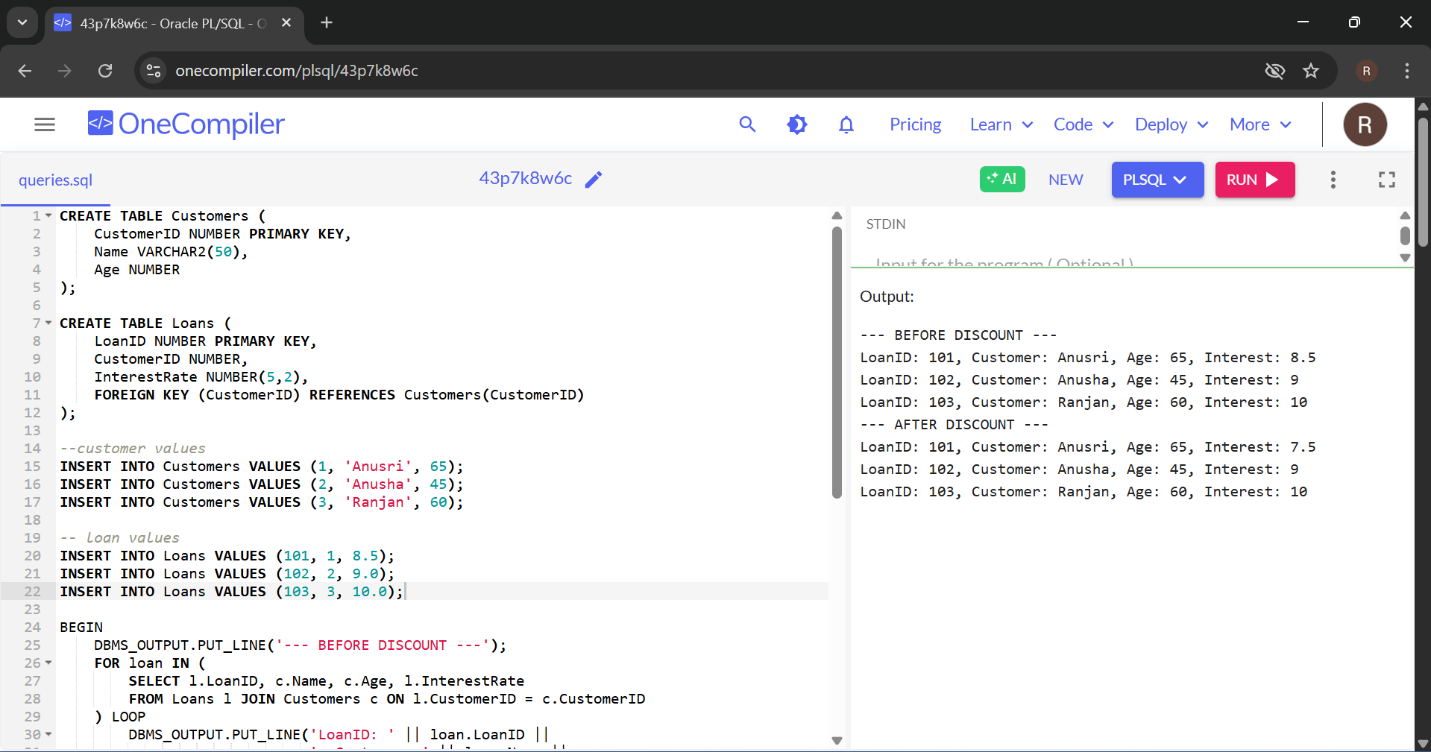
', Customer: ' || loan.Name ||

', Age: ' || loan.Age ||

', Interest: ' || loan.InterestRate);

END LOOP;

END;



**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.

**Code & Output:**

CREATE TABLE Accounts (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Balance NUMBER(10,2),

IsVIP CHAR(1) DEFAULT 'N' -- 'Y' for VIP, 'N' for non-VIP

);

INSERT INTO Accounts VALUES (1, 'Dhanush', 19500, 'N');

INSERT INTO Accounts VALUES (2, 'Varshu', 14000, 'N');

INSERT INTO Accounts VALUES (3, 'Sharan', 17880, 'N');

INSERT INTO Accounts VALUES (4, 'Rohith', 9000, 'N');

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- BEFORE VIP UPDATE ---');

FOR acc IN (SELECT \* FROM Accounts) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || acc.Name ||

', Balance: ' || acc.Balance ||

', IsVIP: ' || acc.IsVIP);

END LOOP;

FOR acc IN (SELECT CustomerID FROM Accounts WHERE Balance > 10000) LOOP

UPDATE Accounts

SET IsVIP = 'Y'

WHERE CustomerID = acc.CustomerID;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- AFTER VIP UPDATE ---');

FOR acc IN (SELECT \* FROM Accounts) LOOP

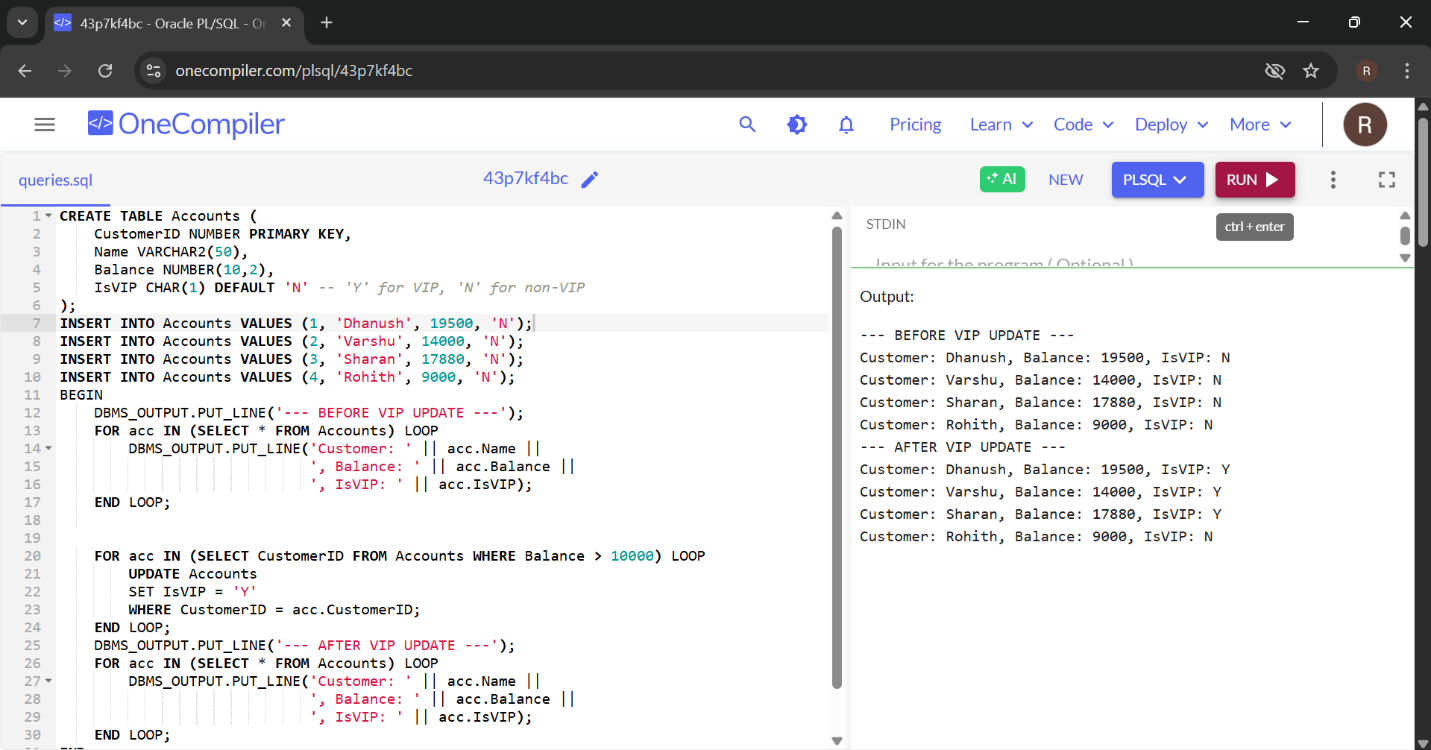
DBMS\_OUTPUT.PUT\_LINE('Customer: ' || acc.Name ||

', Balance: ' || acc.Balance ||

', IsVIP: ' || acc.IsVIP);

END LOOP;

END;



**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.

**Code & Output:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50)

);

CREATE TABLE LoanDetails (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'Dhanush');

INSERT INTO Customers VALUES (2, 'Varshu');

INSERT INTO Customers VALUES (3, 'Sharan');

INSERT INTO Customers VALUES (4, 'Rohith');

INSERT INTO LoanDetails VALUES (201, 1, SYSDATE + 10);

INSERT INTO LoanDetails VALUES (202, 2, SYSDATE + 45);

INSERT INTO LoanDetails VALUES (203, 3, SYSDATE + 5);

INSERT INTO LoanDetails VALUES (204, 4, SYSDATE + 90);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- REMINDER MESSAGES ---');

FOR loan IN (

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

FROM LoanDetails l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

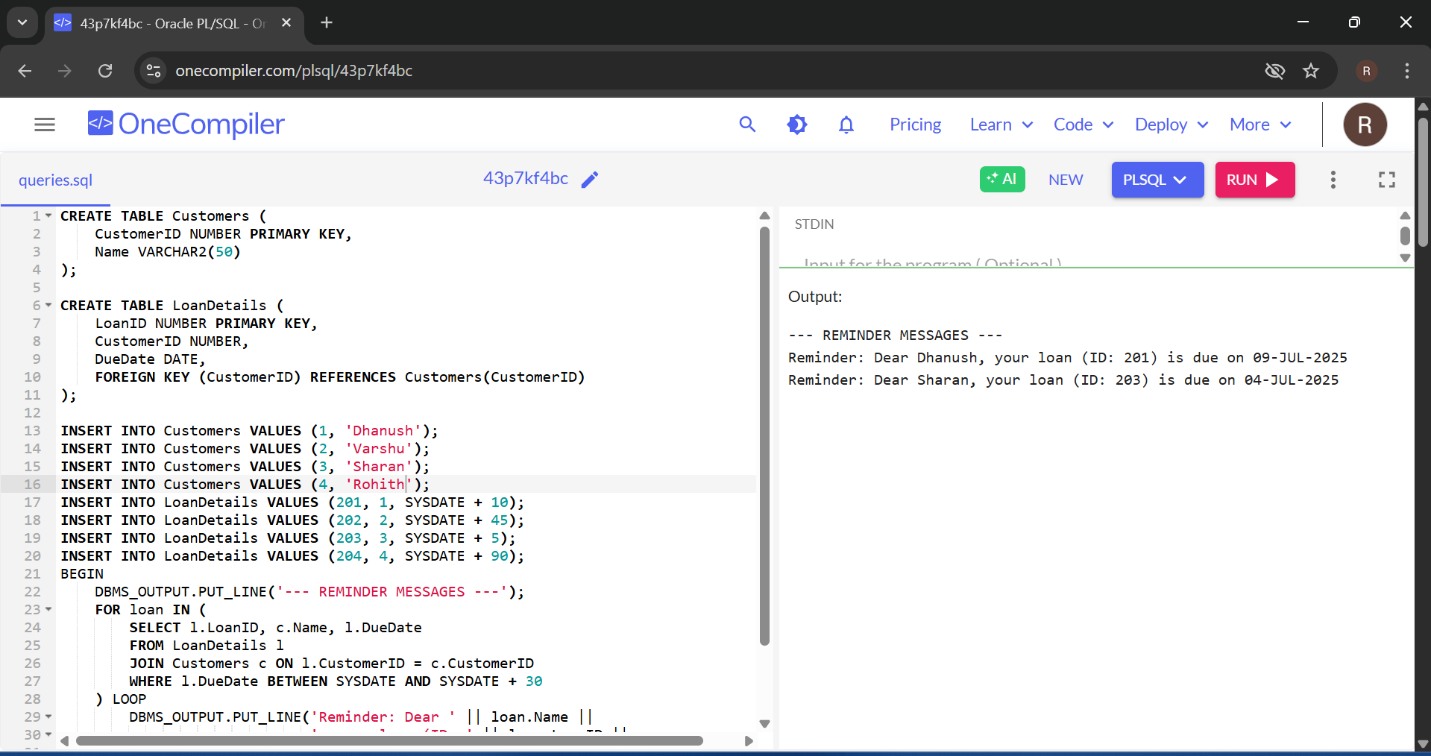
DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan.Name ||

', your loan (ID: ' || loan.LoanID ||

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

END LOOP;

END;



**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.

**Code & Output :**

CREATE TABLE SavingsAccount (

AccountNumber NUMBER PRIMARY KEY,

CustomerName VARCHAR2(50),

Balance NUMBER(10,2)

);

INSERT INTO SavingsAccount VALUES (101, 'Dhanush', 12999);

INSERT INTO SavingsAccount VALUES (102, 'Varshu', 15000);

INSERT INTO SavingsAccount VALUES (103, 'Sharan', 50000);

INSERT INTO SavingsAccount VALUES (104, 'Rohith', 30000);

commit;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

INTEREST\_RATE CONSTANT NUMBER := 0.01;

BEGIN

FOR acc IN (SELECT AccountNumber FROM SavingsAccount) LOOP

UPDATE SavingsAccount

SET Balance = Balance + (Balance \* INTEREST\_RATE)

WHERE AccountNumber = acc.AccountNumber;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- BEFORE INTEREST ---');

FOR acc IN (SELECT \* FROM SavingsAccount) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account: ' || acc.AccountNumber ||

', Name: ' || acc.CustomerName ||

', Balance: ' || TO\_CHAR(acc.Balance, '9999999.99'));

END LOOP;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- AFTER INTEREST ---');

FOR acc IN (SELECT \* FROM SavingsAccount) LOOP

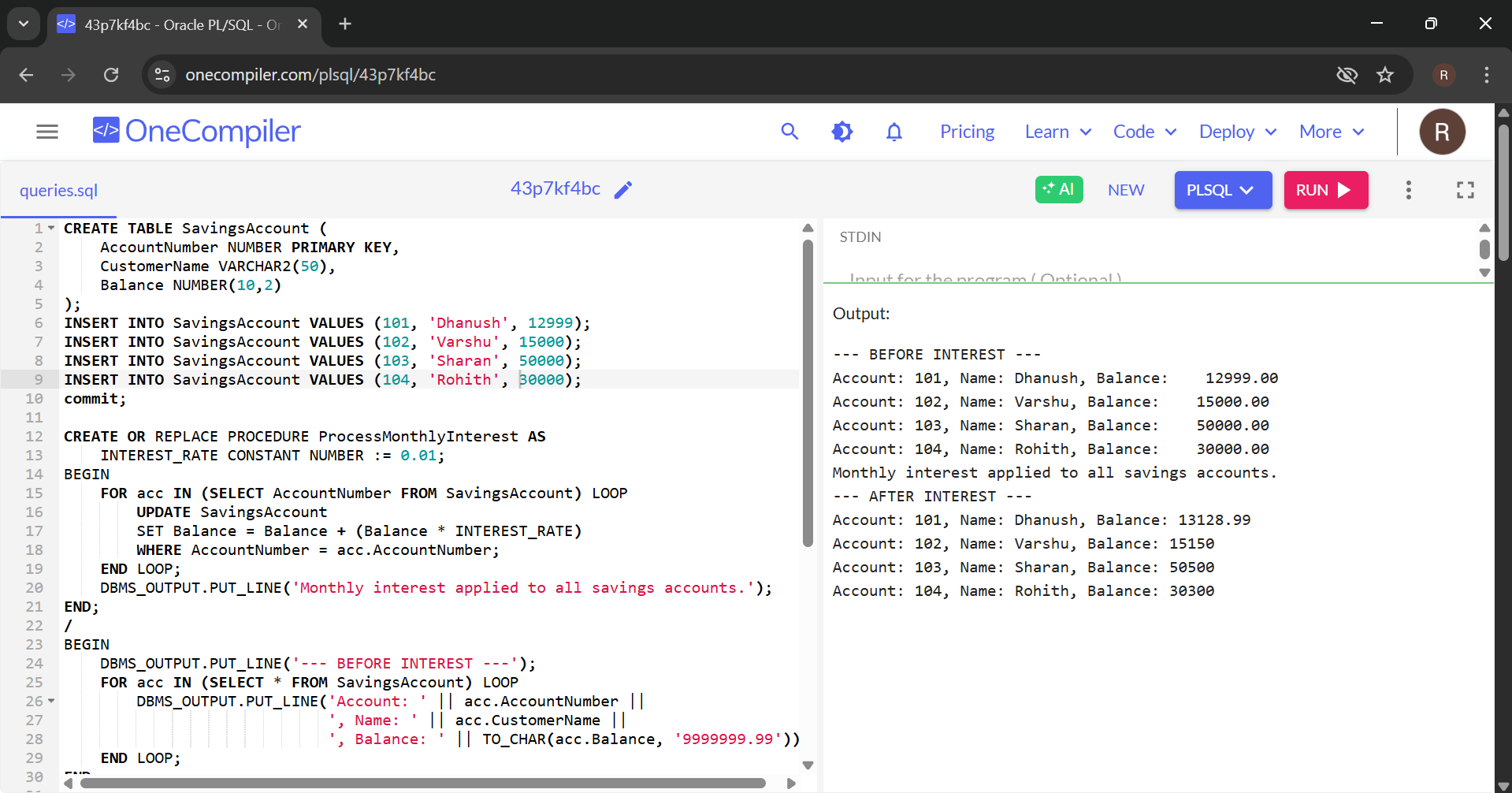
DBMS\_OUTPUT.PUT\_LINE('Account: ' || acc.AccountNumber ||

', Name: ' || acc.CustomerName ||

', Balance: ' || acc.Balance);

END LOOP;

END;



**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.

**Code & Output :**

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Department VARCHAR2(30),

Salary NUMBER(10,2)

);

INSERT INTO Employees VALUES (1, 'Dhanush', 'IT', 40000);

INSERT INTO Employees VALUES (2, 'Varshu', 'Finance', 50000);

INSERT INTO Employees VALUES (3, 'Rohith', 'HR', 42000);

INSERT INTO Employees VALUES (4, 'Sharan', 'IT', 60000);

INSERT INTO Employees VALUES (5, 'Naren', 'Finance', 48000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_pct / 100)

WHERE Department = dept\_name;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || bonus\_pct || '% applied to ' || dept\_name || ' department.');

END;

/

-- Print salaries before bonus

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- BEFORE BONUS ---');

FOR emp IN (SELECT \* FROM Employees) LOOP

DBMS\_OUTPUT.PUT\_LINE('Name: ' || emp.Name || ', Dept: ' || emp.Department || ', Salary: ' || TO\_CHAR(emp.Salary, '9999999.99'));

END LOOP;

END;

/

-- Call procedure

BEGIN

UpdateEmployeeBonus('Finance', 10);

END;

/

-- Print salaries after bonus

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- AFTER BONUS ---');

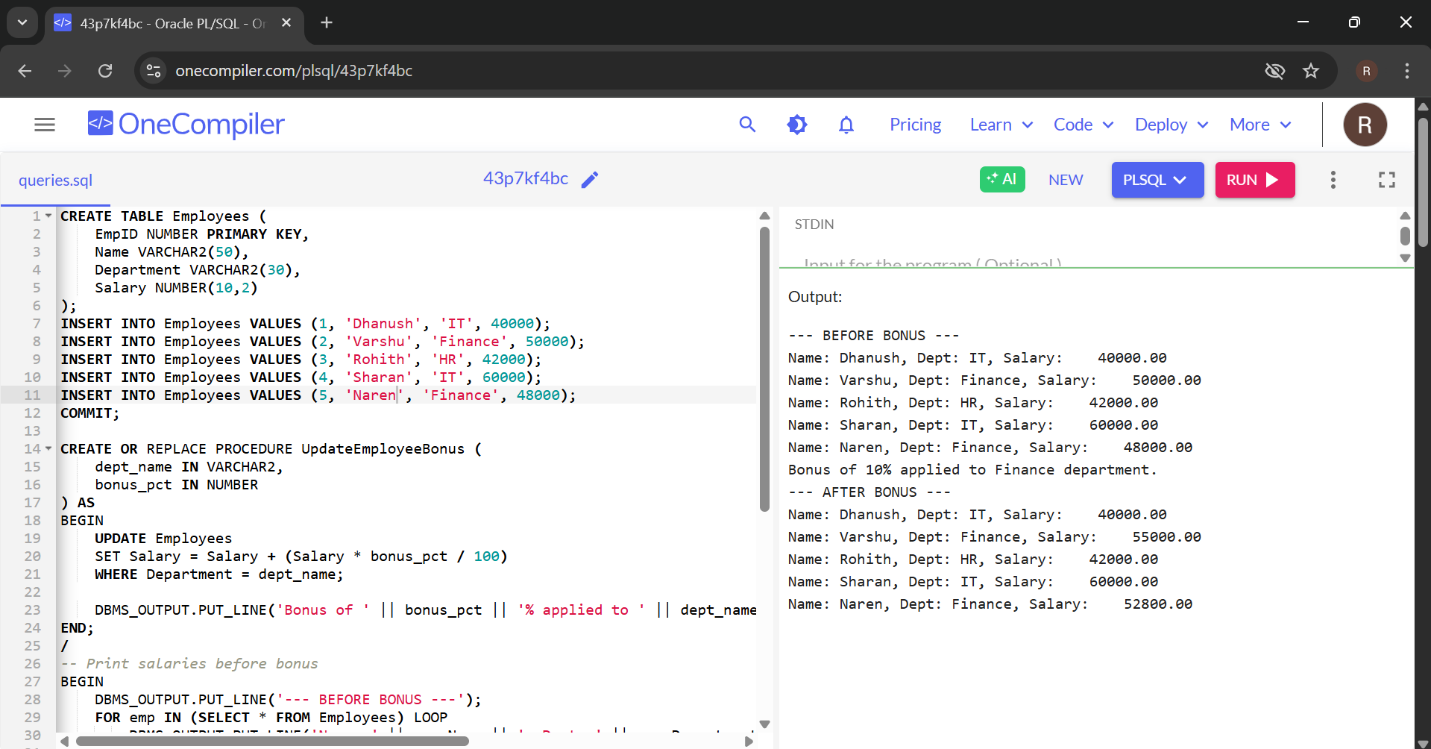
FOR emp IN (SELECT \* FROM Employees) LOOP

DBMS\_OUTPUT.PUT\_LINE('Name: ' || emp.Name || ', Dept: ' || emp.Department || ', Salary: ' || TO\_CHAR(emp.Salary, '9999999.99'));

END LOOP;

END;

/



**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.

**Code & Output:**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountHolder VARCHAR2(50),

Balance NUMBER(12,2)

);

INSERT INTO Accounts VALUES (1, 'Dhanush', 19000);

INSERT INTO Accounts VALUES (2, 'Rohith', 15000);

INSERT INTO Accounts VALUES (3, 'Varshu', 70000);

INSERT INTO Accounts VALUES (4, 'Sharan', 17000);

INSERT INTO Accounts VALUES (5, 'Naren', 47000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account\_id IN NUMBER,

to\_account\_id IN NUMBER,

amount IN NUMBER

)

AS

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account\_id;

IF from\_balance < amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account\_id;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from Account '

|| from\_account\_id || ' to Account ' || to\_account\_id);

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Invalid account ID.');

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- BEFORE TRANSFER ---');

FOR acc IN (SELECT \* FROM Accounts) LOOP

DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || acc.AccountID ||

', Holder: ' || acc.AccountHolder ||

', Balance: ' || acc.Balance);

END LOOP;

END;

/

EXEC TransferFunds(1, 2, 7000);

EXEC TransferFunds(1, 4, 12000);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- AFTER TRANSFER ---');

FOR acc IN (SELECT \* FROM Accounts) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account: ' || acc.AccountID ||

', Holder: ' || acc.AccountHolder ||

', Balance: ' || acc.Balance);

END LOOP;

END;

