**Week 2 Hands-on exercises:**

PL/SQL Programming :

**Exercise 1:**

Control Structures

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

o 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 :**

CREATE TABLE Customers (

CustomerID NUMBER,

Name VARCHAR2(50),

Age NUMBER,

LoanInterestRate NUMBER

);

INSERT INTO Customers VALUES (101, 'John', 65, 10);

INSERT INTO Customers VALUES (102, 'Alice', 45, 9);

INSERT INTO Customers VALUES (103, 'David', 70, 8);

COMMIT;

BEGIN

FOR rec IN (SELECT CustomerID FROM Customers WHERE Age > 60) LOOP

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

END;

/

SELECT \* from Customers;

**Output :**

A screenshot of a computer

AI-generated content may be incorrect.

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

o 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 :**

CREATE TABLE Customers (

CustomerID NUMBER,

Name VARCHAR2(50),

Age NUMBER,

Balance NUMBER,

LoanInterestRate NUMBER,

IsVIP VARCHAR2(5)

);

INSERT INTO Customers VALUES (1, 'Alice', 35, 15000, 9.0, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 62, 8000, 10.0, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 45, 12000, 8.5, 'FALSE');

COMMIT;

BEGIN

FOR rec IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END LOOP;

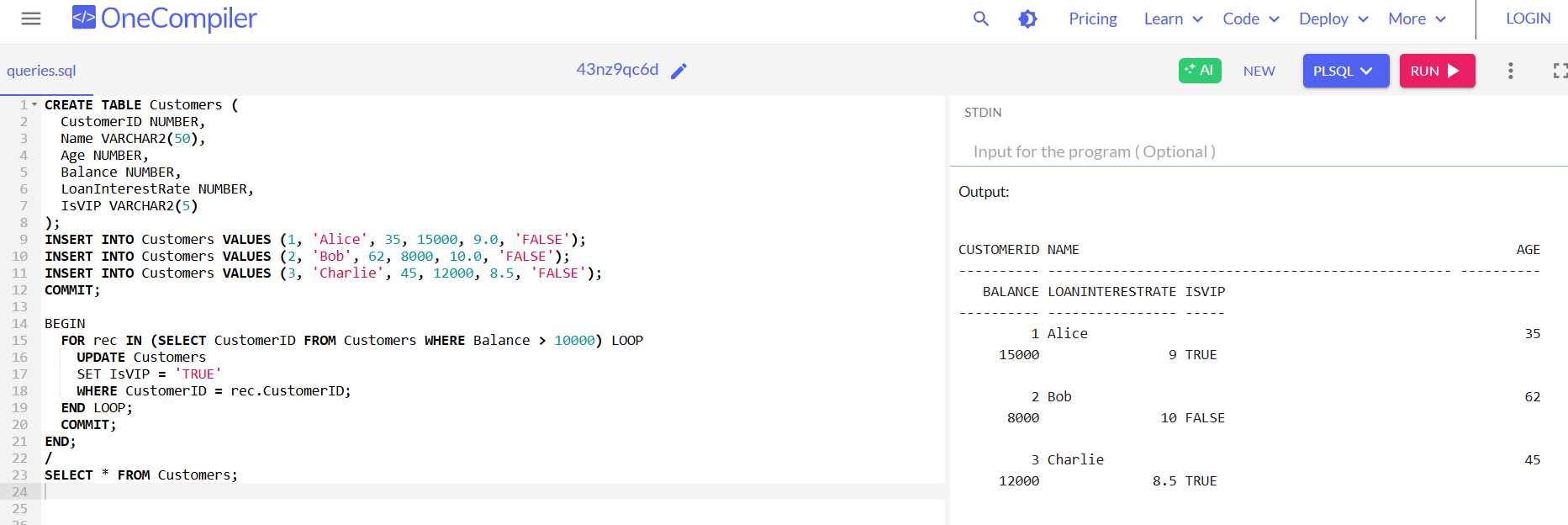
COMMIT;

END;

/

SELECT \* FROM Customers;

**Output :**

****

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

o 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 :**

CREATE TABLE Loans (

LoanID NUMBER,

CustomerID NUMBER,

DueDate DATE

);

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

INSERT INTO Loans VALUES (202, 2, SYSDATE + 25);

INSERT INTO Loans VALUES (203, 3, SYSDATE + 45);

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate <= SYSDATE + 30

) LOOP

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

' for Customer ' || rec.CustomerID ||

' is due on ' || rec.DueDate);

END LOOP;

END;

/

**Output :**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 3 :**

Stored Procedures

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

o 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 :**

CREATE TABLE Accounts (

AccountID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

INSERT INTO Accounts VALUES (1, 'Savings', 1000);

INSERT INTO Accounts VALUES (2, 'Savings', 1500);

INSERT INTO Accounts VALUES (3, 'Current', 2000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

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

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

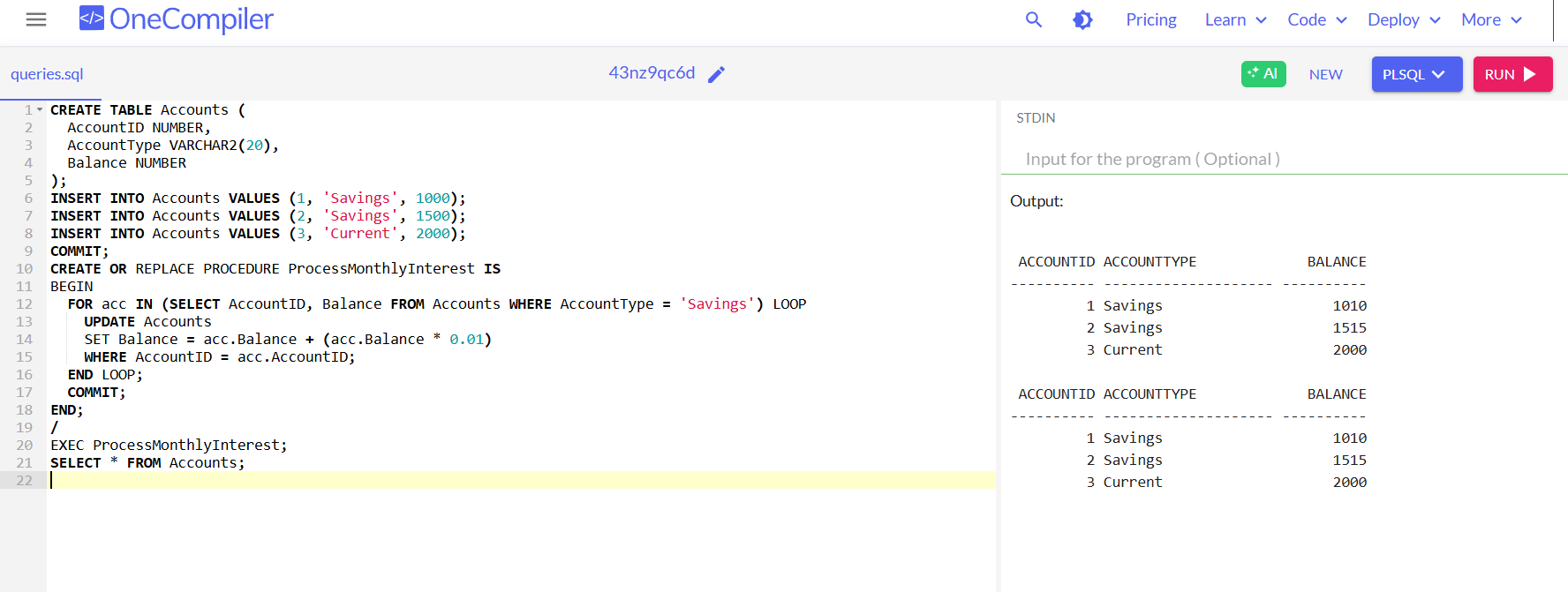
END;

/

EXEC ProcessMonthlyInterest;

SELECT \* FROM Accounts;

**Output :**

****

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

o 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 :**

SET SERVEROUTPUT ON;

CREATE TABLE Employees (

EmployeeID NUMBER,

DepartmentID NUMBER,

Salary NUMBER

);

INSERT INTO Employees VALUES (1, 101, 20000);

INSERT INTO Employees VALUES (2, 101, 25000);

INSERT INTO Employees VALUES (3, 102, 30000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_DeptID IN NUMBER,

p\_BonusPercent IN NUMBER

) IS

v\_Count NUMBER := 0;

BEGIN

UPDATE Employees

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

WHERE DepartmentID = p\_DeptID;

v\_Count := SQL%ROWCOUNT;

COMMIT;

END;

/

EXEC UpdateEmployeeBonus(101, 10);

SELECT \* FROM Employees;

**Output :**



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

o 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 :**

CREATE TABLE Accounts (

AccountID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

INSERT INTO Accounts VALUES (4, 'Savings', 5000);

INSERT INTO Accounts VALUES (5, 'Savings', 3000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccount IN NUMBER,

p\_ToAccount IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAccount;

IF v\_Balance >= p\_Amount THEN

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_FromAccount;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAccount;

COMMIT;

ELSE

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

END IF;

END;

/

EXEC TransferFunds(4, 5, 1000);

SELECT \* FROM Accounts;

**Output :**

****

TDD using JUnit5 and Mockito :

Junit Basic Testing Exercises :

Exercise 1: Setting Up JUnit

Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your

pom.xml:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project.