**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.

**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.

**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.

**Customers Table**

DROP TABLE IF EXISTS Customers;

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(50),

Age INT,

Balance DECIMAL(10,2),

IsVIP BOOLEAN DEFAULT FALSE

);

Loans Table

DROP TABLE IF EXISTS Loans;

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount DECIMAL(10,2),

InterestRate DECIMAL(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Insert Sample Data**

-- Insert Customers

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000.00, FALSE);

INSERT INTO Customers VALUES (2, 'Bob', 58, 9500.00, FALSE);

INSERT INTO Customers VALUES (3, 'Charlie', 72, 8000.00, FALSE);

INSERT INTO Customers VALUES (4, 'Diana', 45, 15000.00, FALSE);

-- Insert Loans

INSERT INTO Loans VALUES (1001, 1, 5000.00, 5.00, CURDATE() + INTERVAL 20 DAY);

INSERT INTO Loans VALUES (1002, 2, 6000.00, 6.50, CURDATE() + INTERVAL 40 DAY);

INSERT INTO Loans VALUES (1003, 3, 3000.00, 4.75, CURDATE() + INTERVAL 10 DAY);

INSERT INTO Loans VALUES (1004, 4, 7000.00, 7.00, CURDATE() + INTERVAL 5 DAY);

**Scenario 1: Apply 1% Discount to Interest Rates for Customers Over 60**

BEGIN

FOR cust IN (

SELECT c.CustomerID, l.LoanID

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE c.Age > 60

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

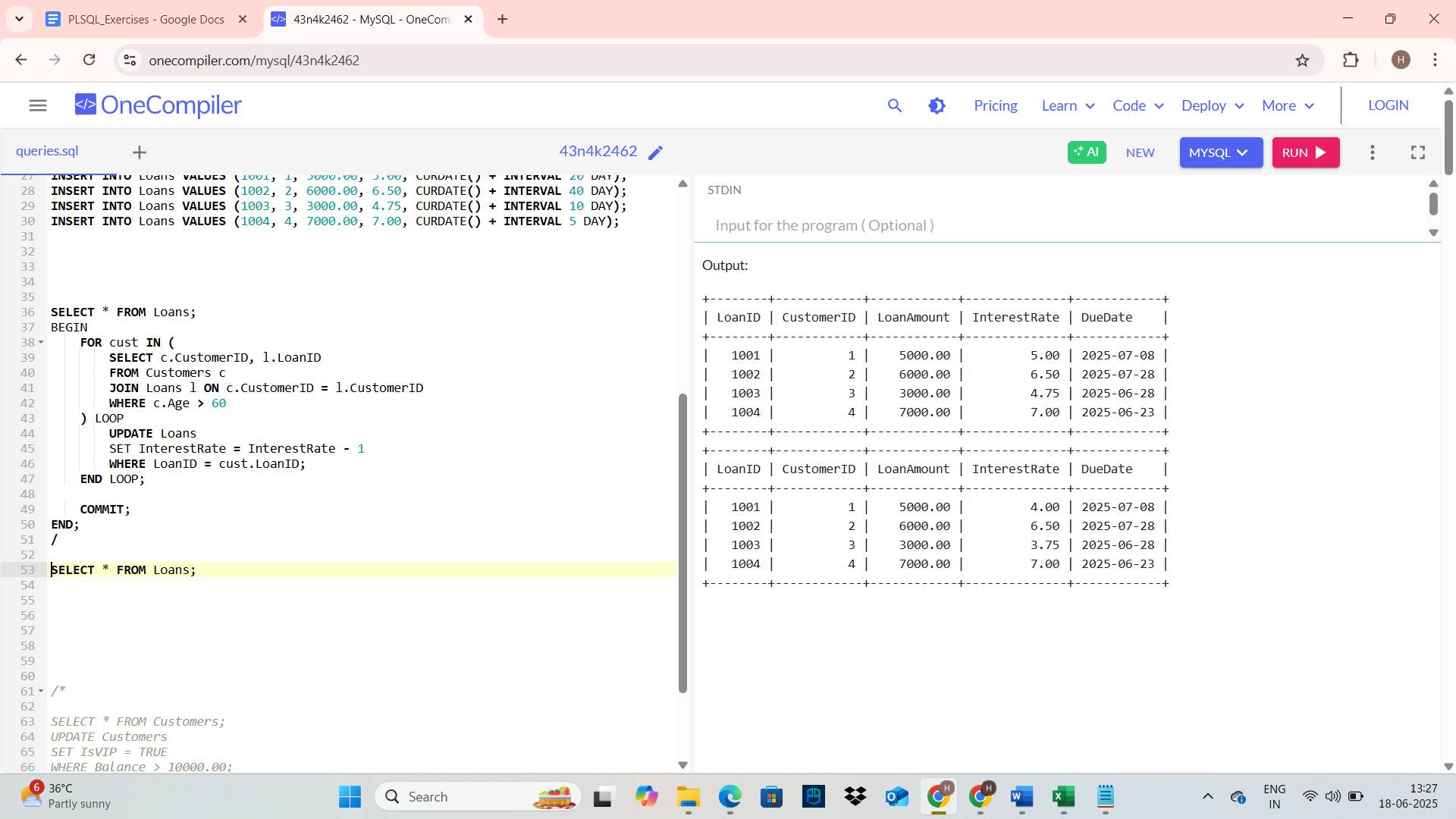
WHERE LoanID = cust.LoanID;

END LOOP;

COMMIT;

END;

/



**Scenario 2: Promote Customers to VIP Based on Balance > $10,000**

BEGIN

FOR cust IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

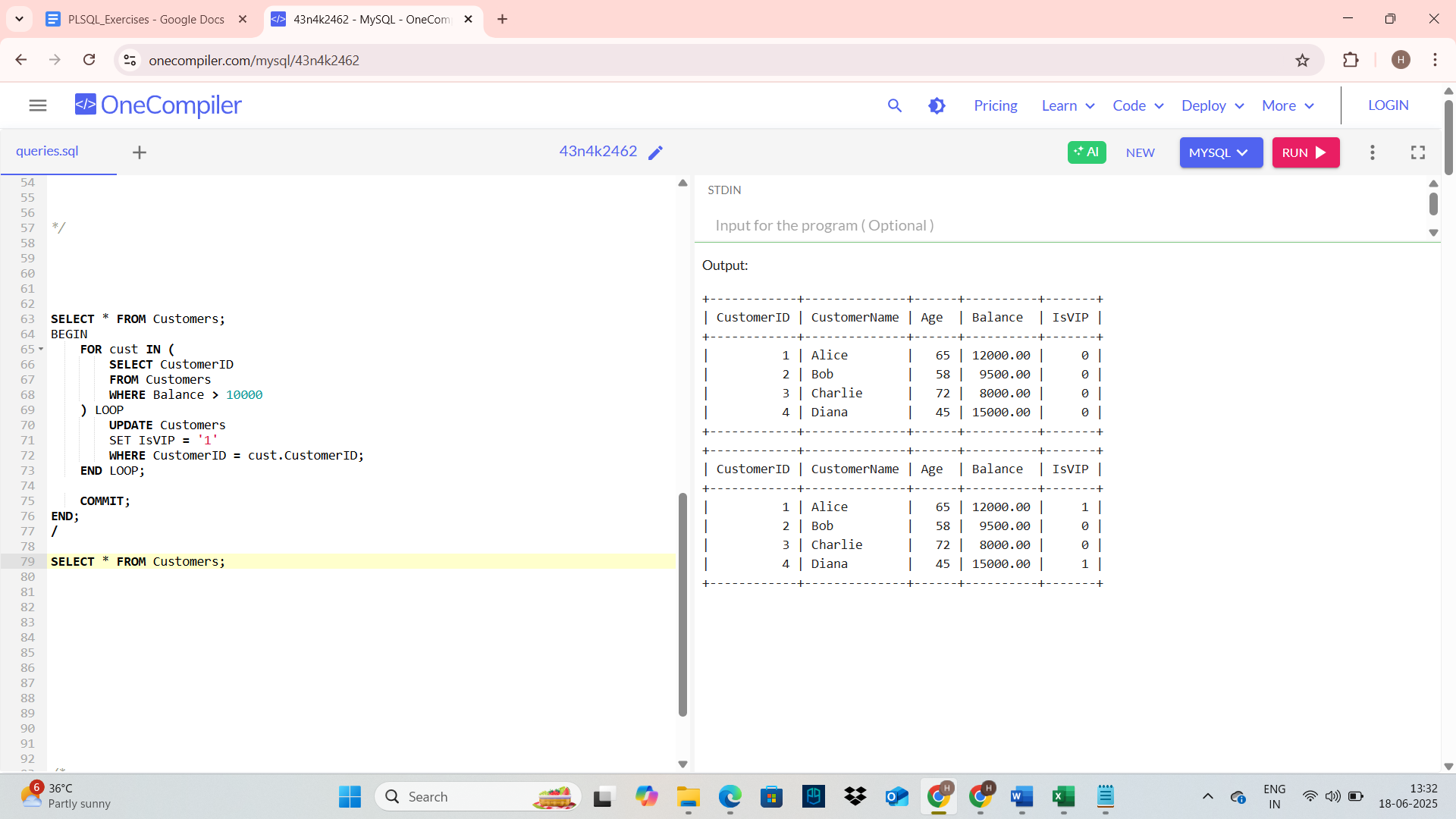
WHERE CustomerID = cust.CustomerID;

END LOOP;

COMMIT;

END;

/



**Scenario 3: Send Reminders for Loans Due in Next 30 Days**

BEGIN

FOR rec IN (

SELECT c.CustomerName, l.LoanID, 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(

'Reminder: Loan ' || rec.LoanID || ' for ' || rec.CustomerName ||

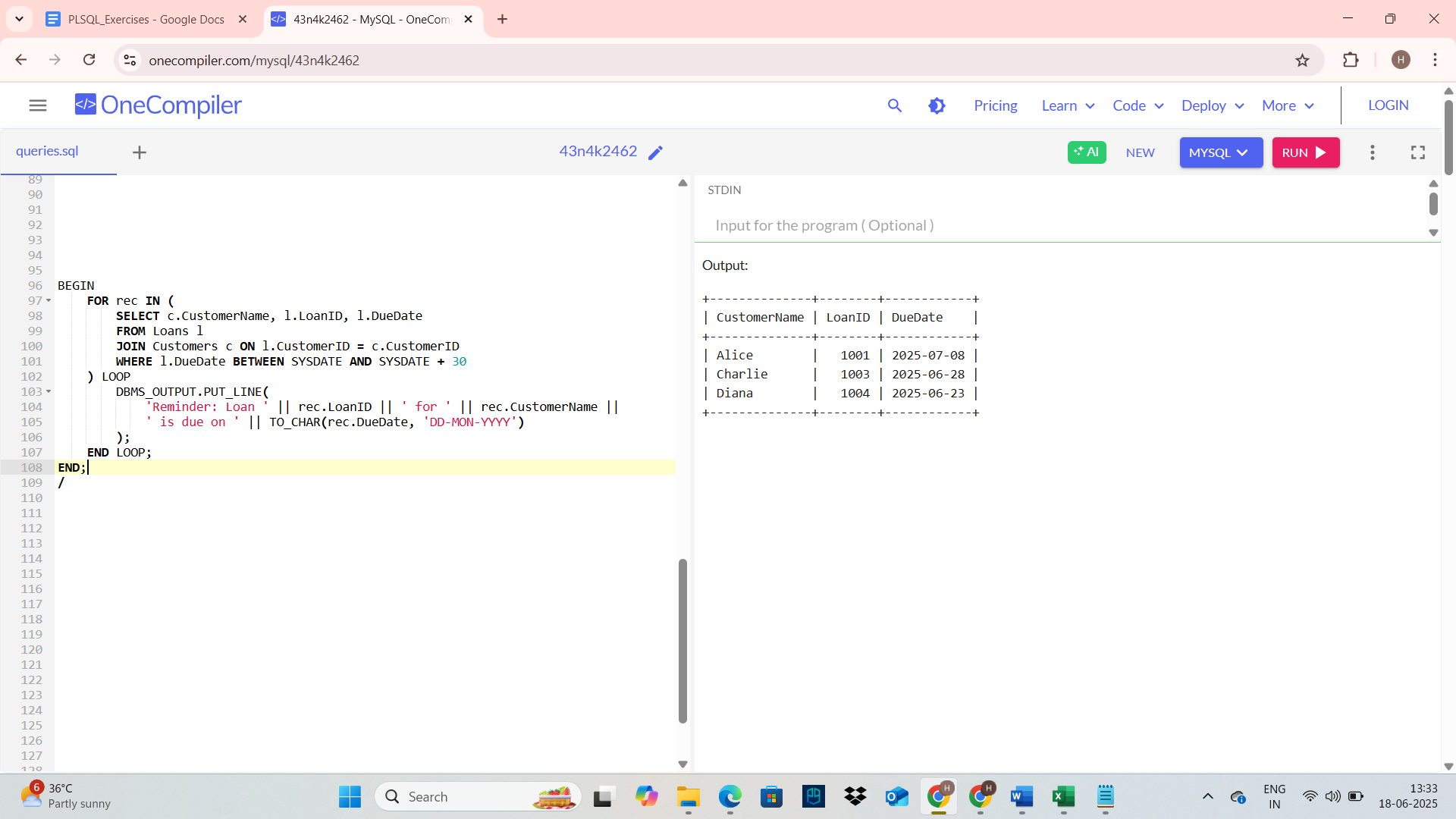
' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY')

);

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.

**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.

**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.

Accounts Table (For Scenarios 1 & 3)

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20), -- 'SAVINGS' or 'CURRENT'

Balance NUMBER

);

Employees Table (For Scenario 2)

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

DepartmentID NUMBER,

Salary NUMBER

);

**Insert Sample Data**

Insert into Accounts

INSERT INTO Accounts VALUES (101, 1, 'SAVINGS', 1000);

INSERT INTO Accounts VALUES (102, 2, 'SAVINGS', 2000);

INSERT INTO Accounts VALUES (103, 3, 'CURRENT', 3000);

INSERT INTO Accounts VALUES (104, 4, 'SAVINGS', 4000);

Insert into Employees

INSERT INTO Employees VALUES (1, 'Alice', 10, 5000);

INSERT INTO Employees VALUES (2, 'Bob', 10, 6000);

INSERT INTO Employees VALUES (3, 'Charlie', 20, 5500);

**Scenario 1**: ProcessMonthlyInterest for Savings Accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

-- Apply 1% interest to all savings accounts

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'SAVINGS';

COMMIT;

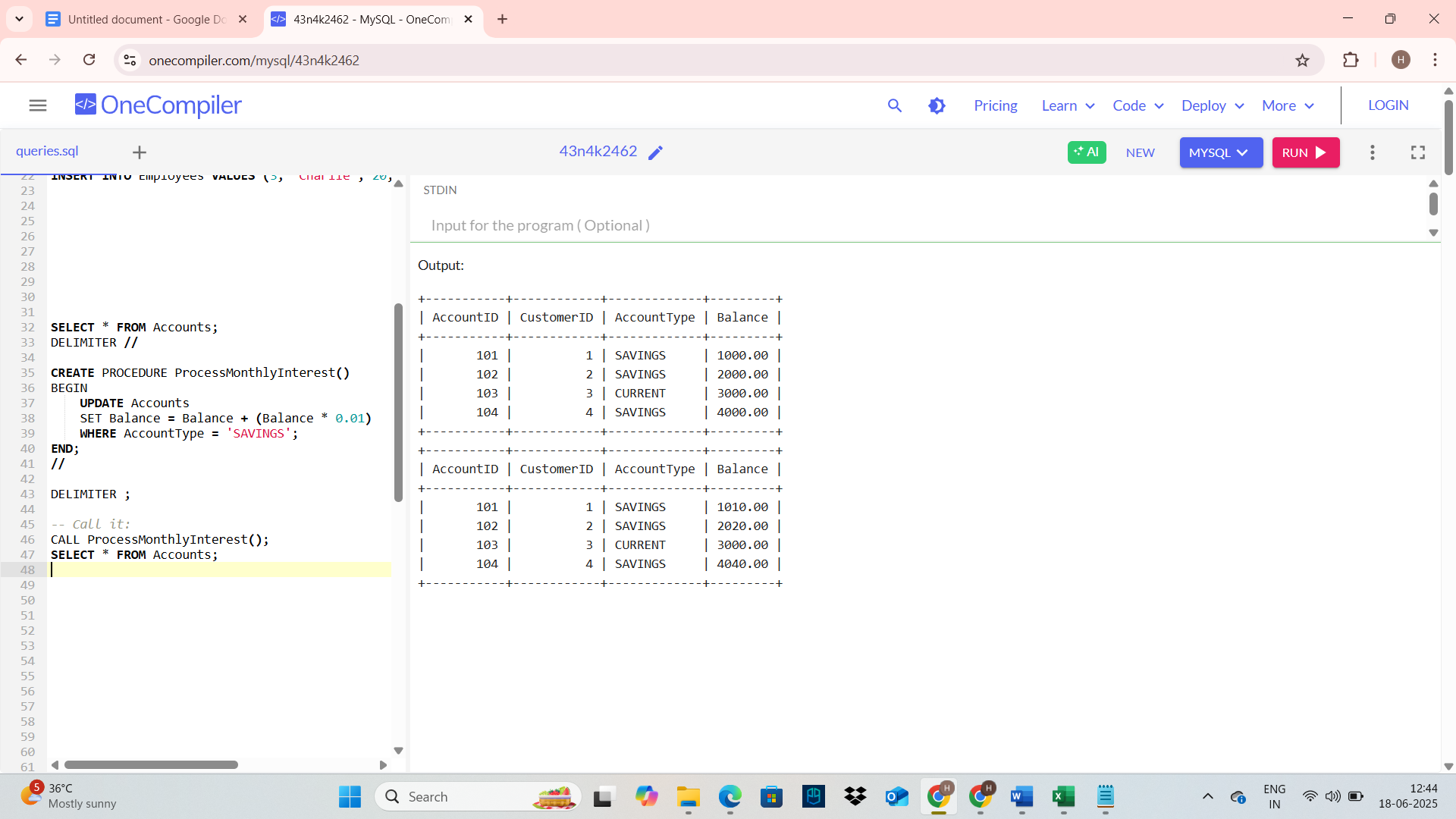
END;

/

-- Call it:

EXEC ProcessMonthlyInterest();

SELECT \* FROM Accounts;



**Scenario 2**: UpdateEmployeeBonus Based on Department and Bonus %

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) IS

BEGIN

-- Add bonus to employee salaries

UPDATE Employees

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

WHERE DepartmentID = p\_DepartmentID;

COMMIT;

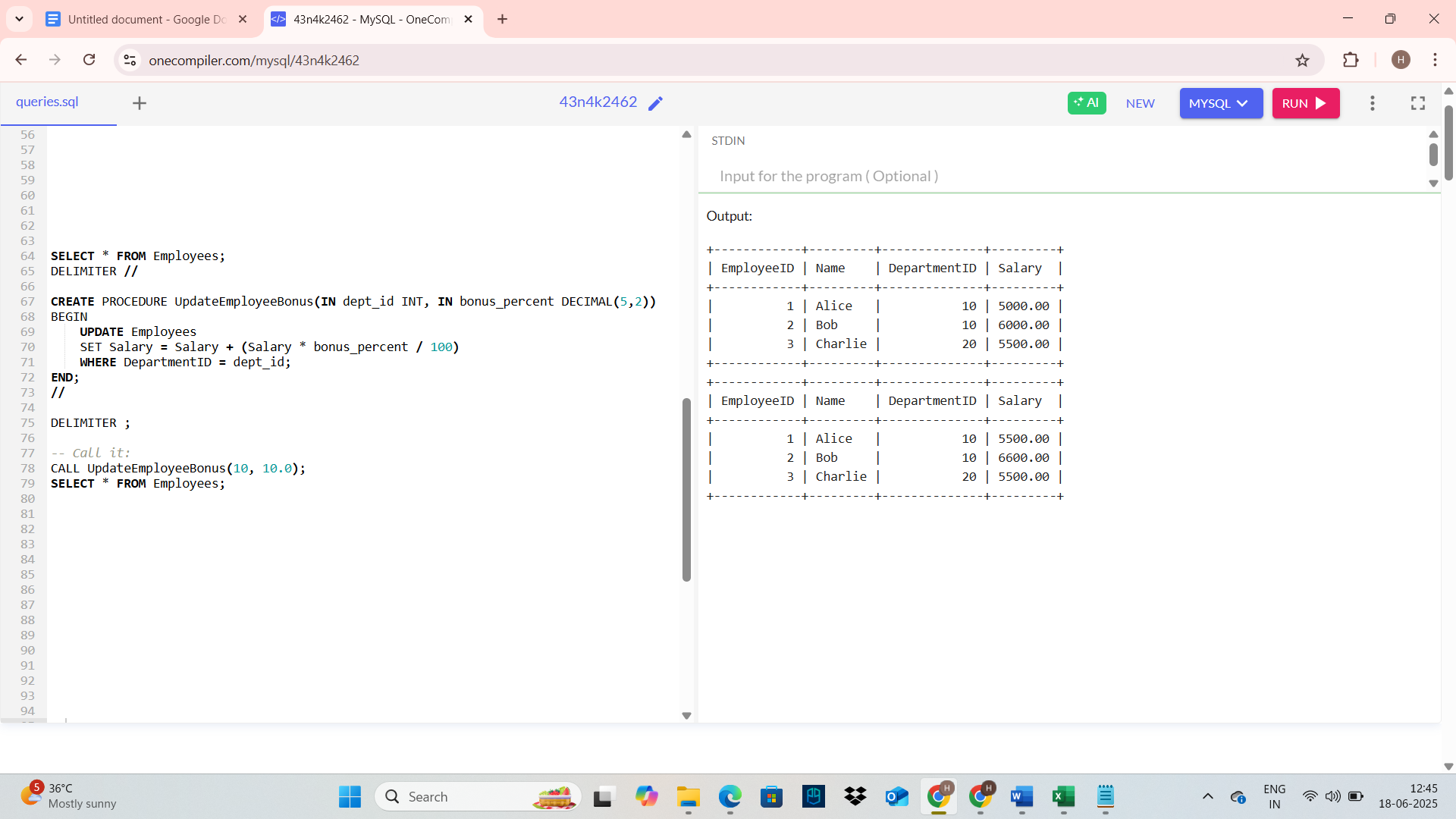
END;

/

-- Call it:

EXEC UpdateEmployeeBonus(10, 10.0);

SELECT \* FROM Employees;



**Scenario 3**: TransferFunds Between Accounts with Balance Check

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

-- Get balance of source account

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

-- Check if source has sufficient balance

IF v\_FromBalance < p\_Amount THEN

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

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

END;

/

-- Call it:

EXEC TransferFunds(104, 103, 500.00);

SELECT \* FROM Accounts;