**Week-2**

**PL/SQL**

**Exercise-1:**

**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.**

**Query:**

SQL> SET SERVEROUTPUT ON;

SQL> BEGIN

FOR loan\_rec IN (

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

FROM Loans l

JOIN Customers c

ON l.CustomerID = c.CustomerID

) LOOP

-- Calculate age in years

IF TRUNC(MONTHS\_BETWEEN(SYSDATE, loan\_rec.DOB)/12) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - (InterestRate \* 0.01)

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

'Applied 1% interest discount for LoanID: ' || loan\_rec.LoanID

);

END IF;

END LOOP;

COMMIT;

END;

/

PL/SQL procedure successfully completed.

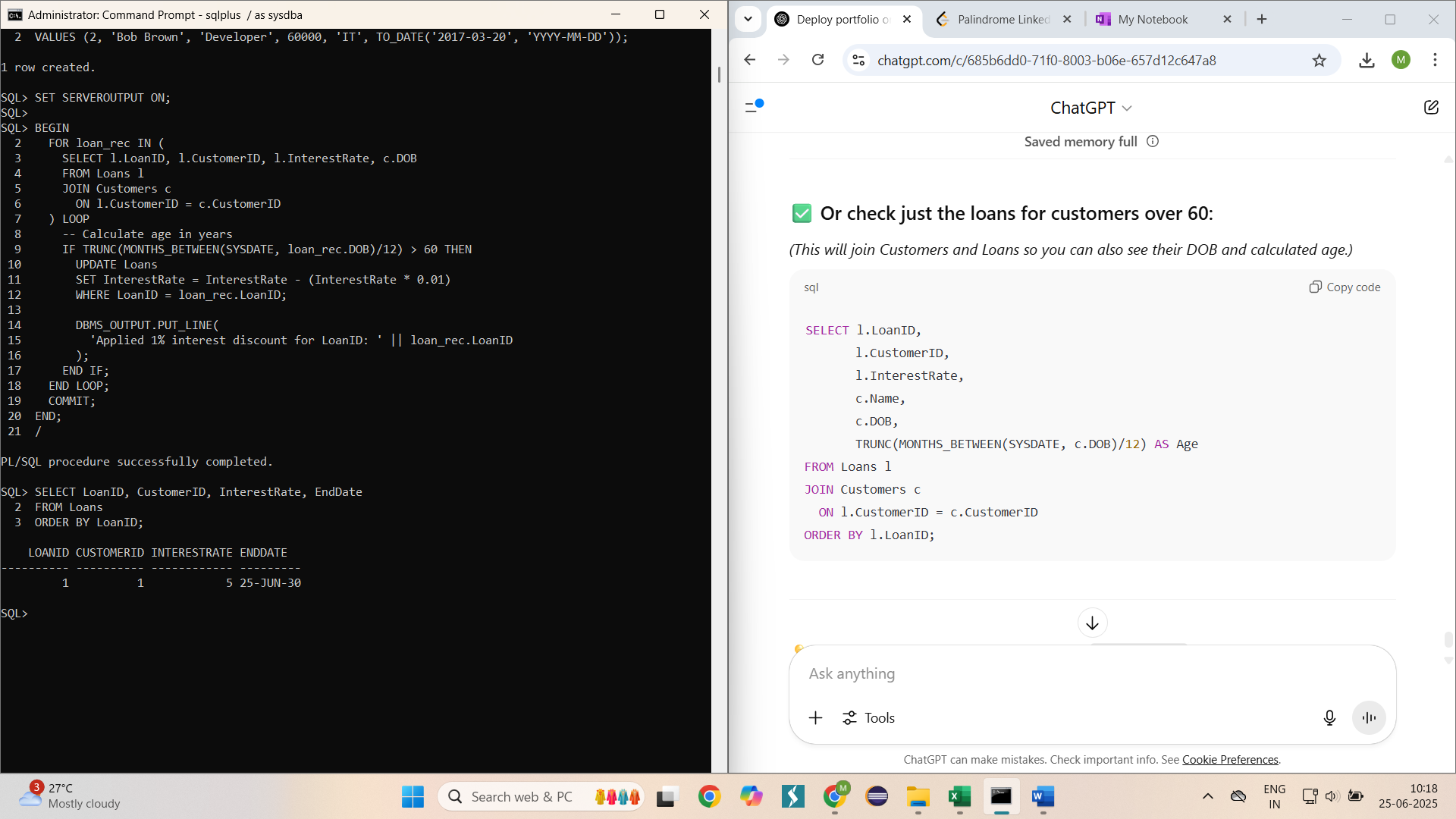
SQL> SELECT LoanID, CustomerID, InterestRate, EndDate

FROM Loans ORDER BY LoanID;

LOANID CUSTOMERID INTERESTRATE ENDDATE

---------- ---------- ------------ ---------

1 1 5 25-JUN-30



**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.**

**Query:**

SQL> ALTER TABLE Customers ADD (IsVIP CHAR(1) DEFAULT 'N');

Table altered.

SQL> SET SERVEROUTPUT ON;

SQL> BEGIN

FOR cust\_rec IN (

SELECT CustomerID, Balance FROM Customers

) LOOP

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

'CustomerID ' || cust\_rec.CustomerID || ' promoted to VIP'

);

END IF;

END LOOP;

COMMIT;

END;

/

PL/SQL procedure successfully completed.

SQL> SELECT CustomerID, Balance, IsVIP

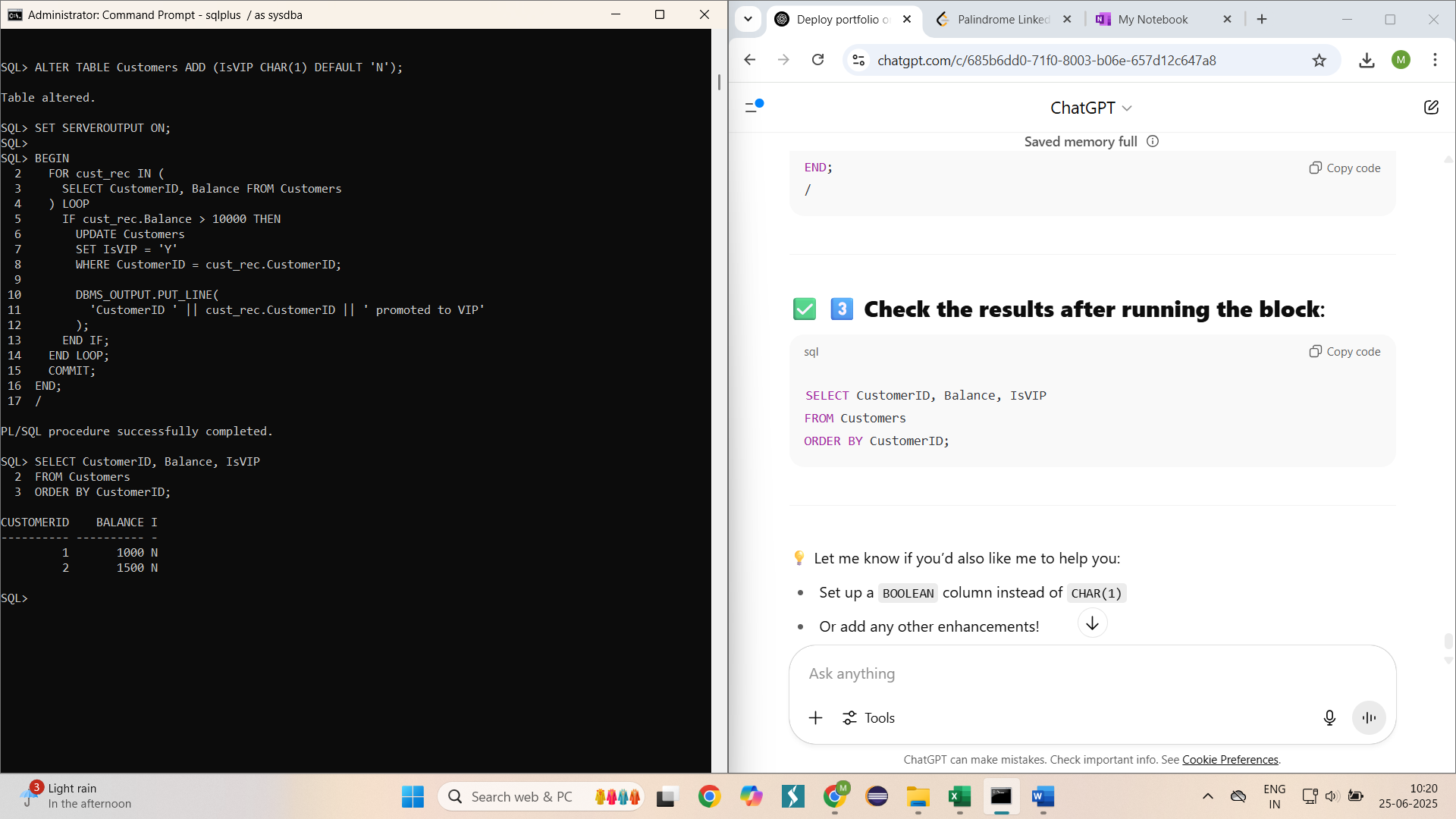
FROM Customers ORDER BY CustomerID;

CUSTOMERID BALANCE I

---------- ---------- -

1 1000 N

2 1500 N



**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.

**Query**:

SQL> INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

2 VALUES (9999, 1, 10000, 5, SYSDATE-100, SYSDATE + 10);

1 row created.

SQL> SET SERVEROUTPUT ON;

SQL> BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.EndDate, c.CustomerID, c.Name

FROM Loans l

JOIN Customers c

ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: LoanID ' || loan\_rec.LoanID ||

' for customer ' || loan\_rec.Name ||

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

);

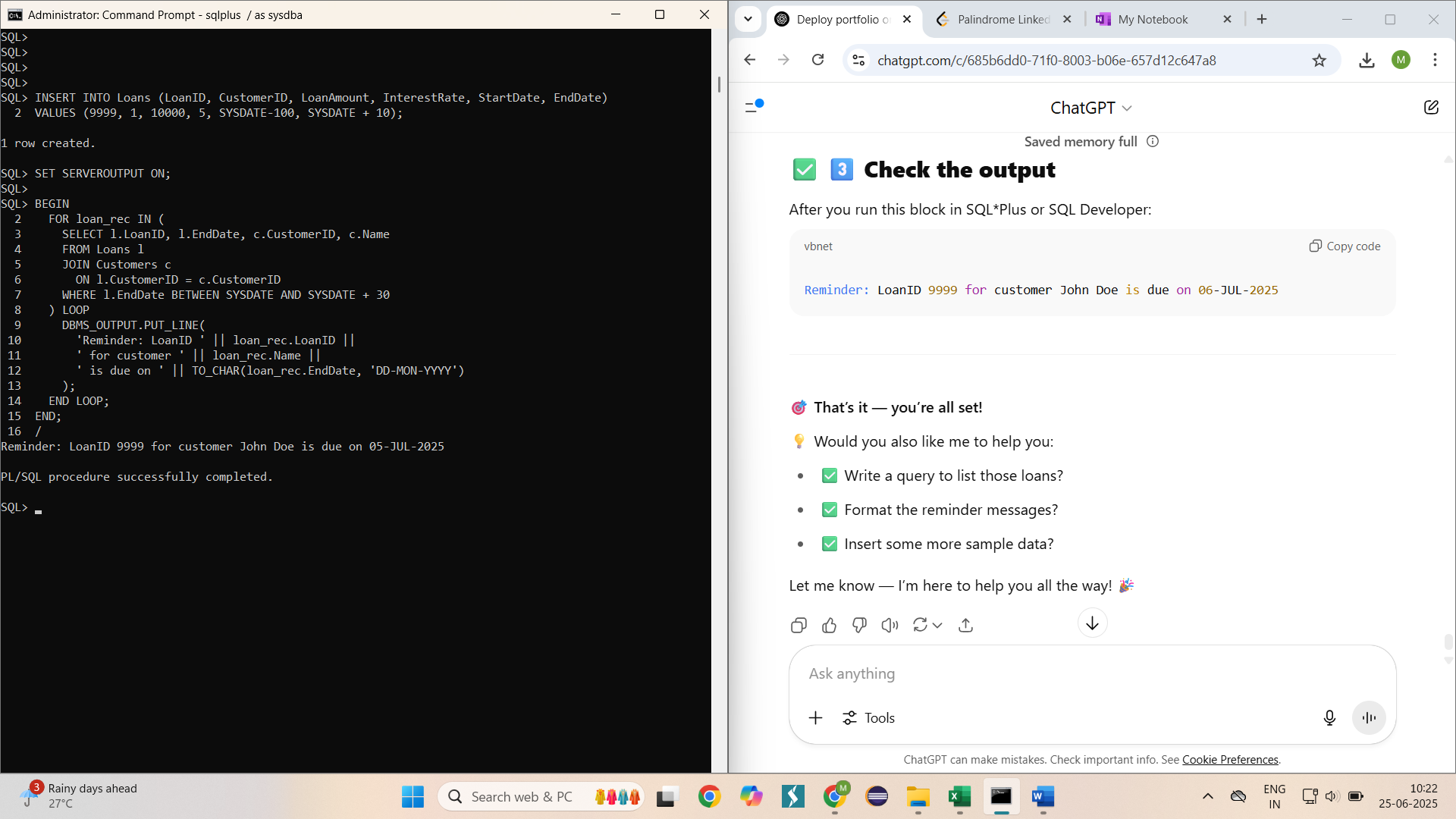
END LOOP;

END;

/

Reminder: LoanID 9999 for customer John Doe is due on 05-JUL-2025

PL/SQL procedure successfully completed.



**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.

**Query :**

SQL> CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

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

END;

/

Procedure created.

SQL> SET SERVEROUTPUT ON;

SQL> EXEC ProcessMonthlyInterest;

Monthly interest processed successfully for all savings accounts.

PL/SQL procedure successfully completed.

SQL> SELECT AccountID, CustomerID, AccountType, Balance

FROM Accounts

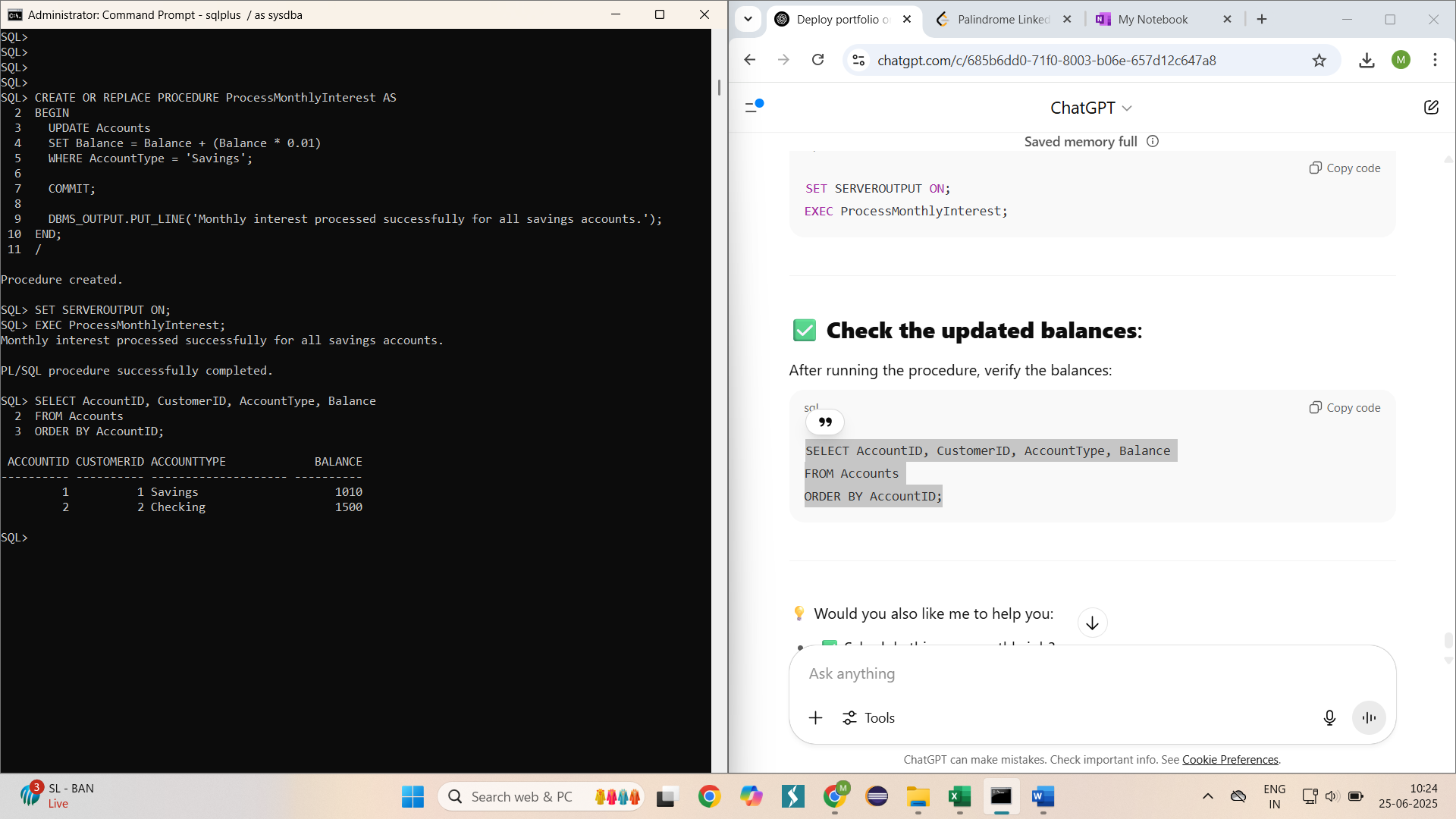
ORDER BY AccountID;

ACCOUNTID CUSTOMERID ACCOUNTTYPE BALANCE

---------- ---------- -------------------- ----------

1 1 Savings 1010

2 2 Checking 1500



**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.

**Query:**

SQL> CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(

'Bonus of ' || p\_bonus\_pct || '% applied to ' || p\_department || ' department.'

);

END;

/

Procedure created.

SQL> SET SERVEROUTPUT ON;

SQL> EXEC UpdateEmployeeBonus('IT', 10);

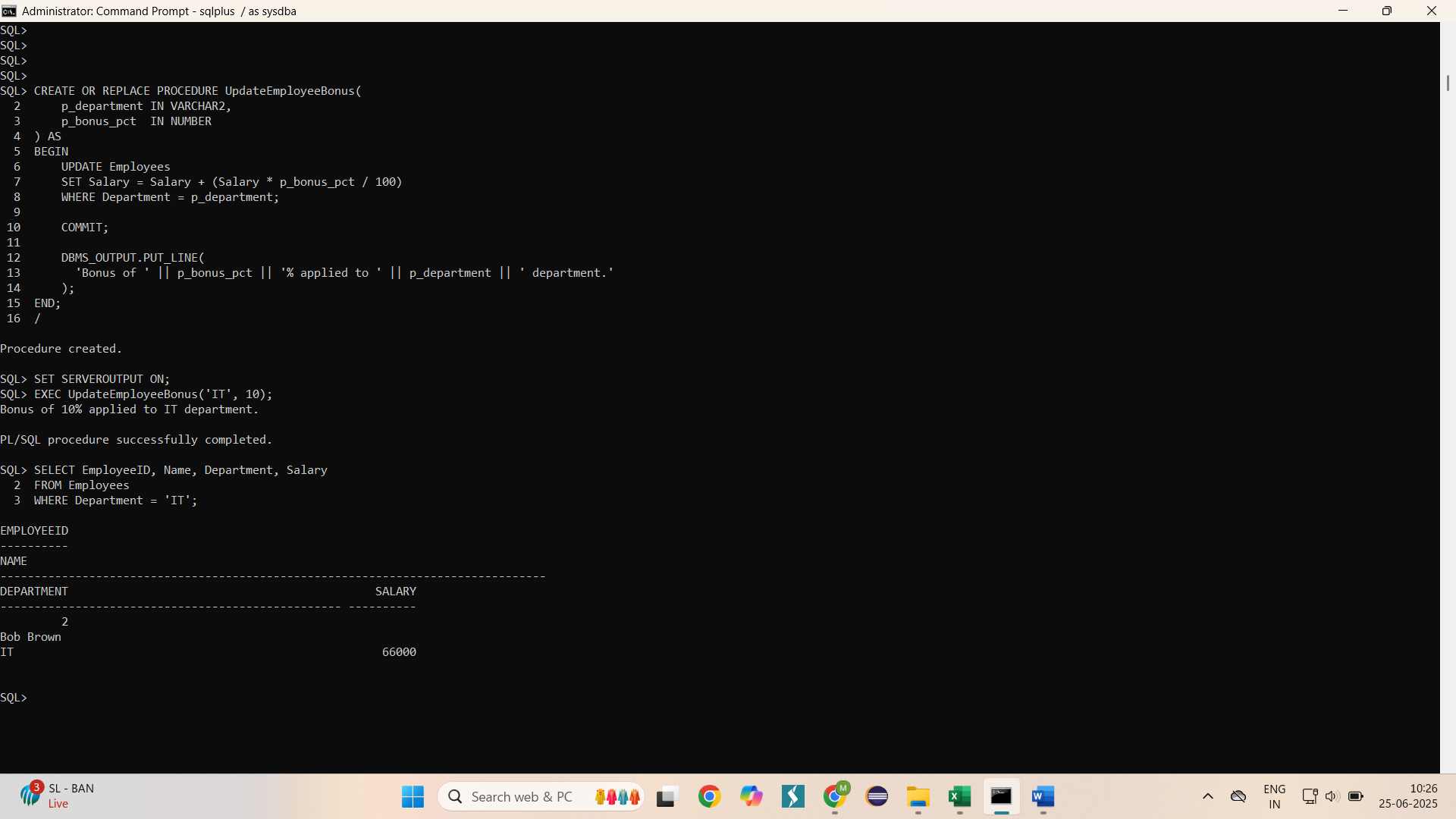
Bonus of 10% applied to IT department.

PL/SQL procedure successfully completed.

SQL> SELECT EmployeeID, Name, Department, Salary

2 FROM Employees

3 WHERE Department = 'IT';



**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.

**Query**

SQL> CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

END;

/

Procedure created.

SQL> EXEC TransferFunds(1,2,500);

PL/SQL procedure successfully completed.

SQL> SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1,2);

ACCOUNTID BALANCE

---------- ----------

1 510

2 2000

