**PL/SQL EXERCISES**

**Exercise 1: Control Structures**

**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.

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (SELECT CustomerID, DOB FROM Customers) LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to customer ID: ' || rec.CustomerID);

END IF;

END LOOP;

COMMIT;

END;

/

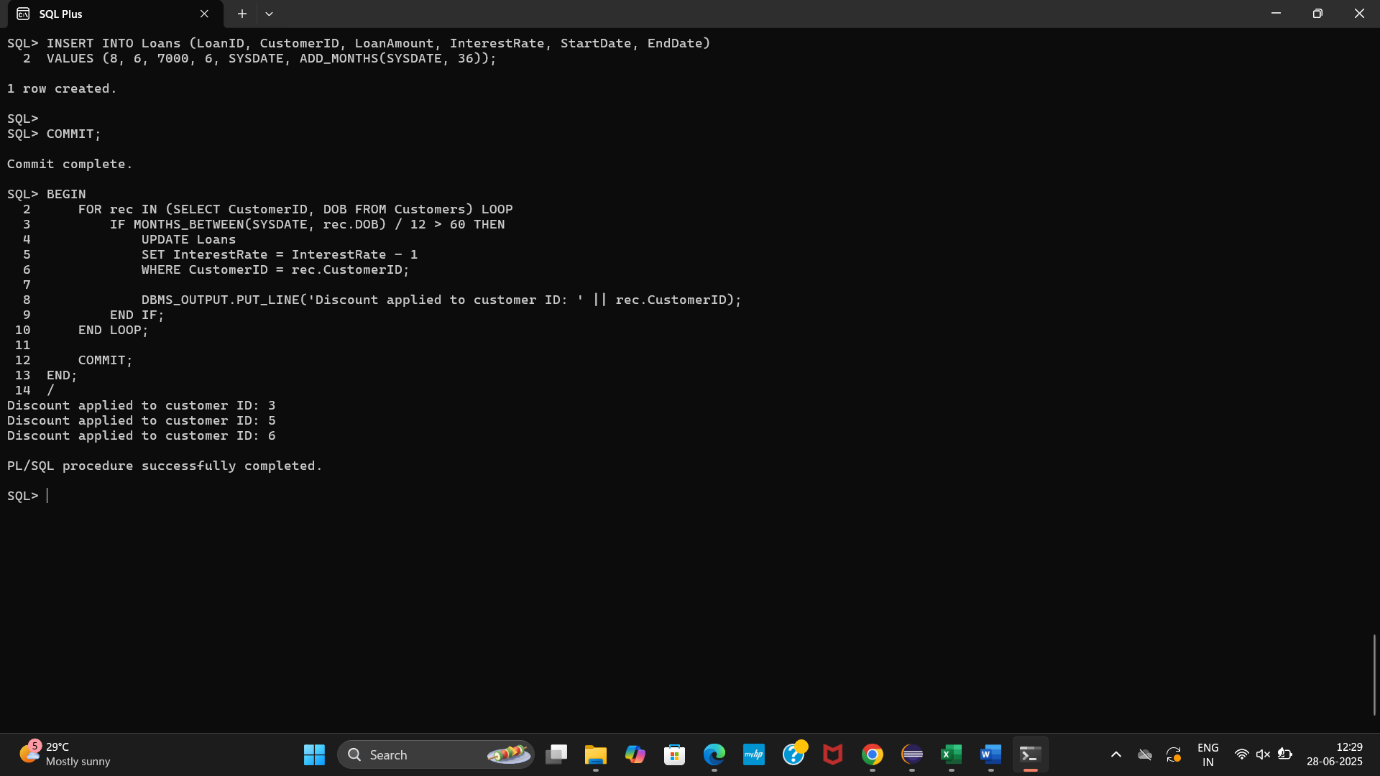
**OUTPUT:**

**Discount applied to customer ID: 3**

**Discount applied to customer ID: 5**

**Discount applied to customer ID: 6**

**PL/SQL procedure successfully completed.**

****

**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.**

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.CustomerID || ' marked as VIP');

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

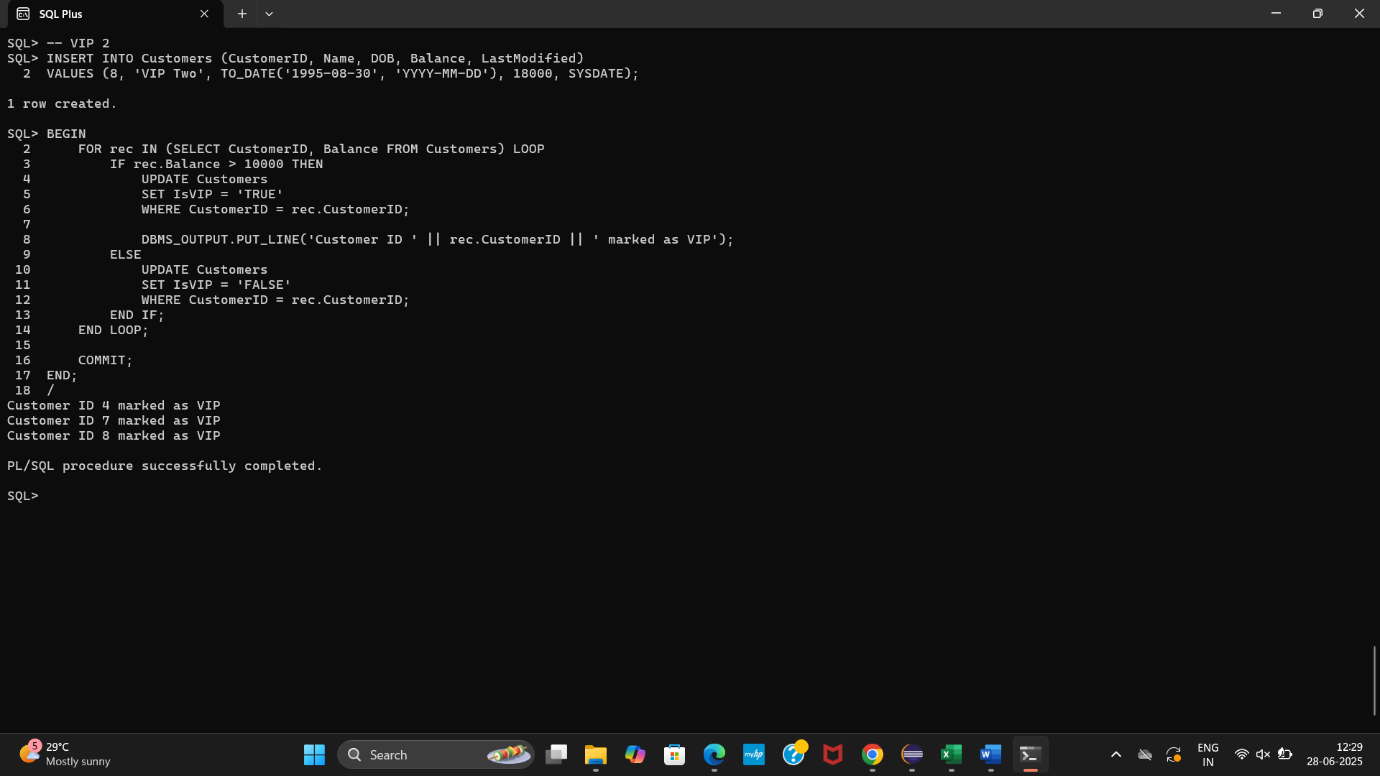
**OUTPUT:**

**Customer ID 4 marked as VIP**

**Customer ID 7 marked as VIP**

**Customer ID 8 marked as VIP**

**PL/SQL procedure successfully completed.**



**Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.**

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

' for Customer ID ' || rec.CustomerID ||

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

END LOOP;

END;

/

**OUTPUT:**

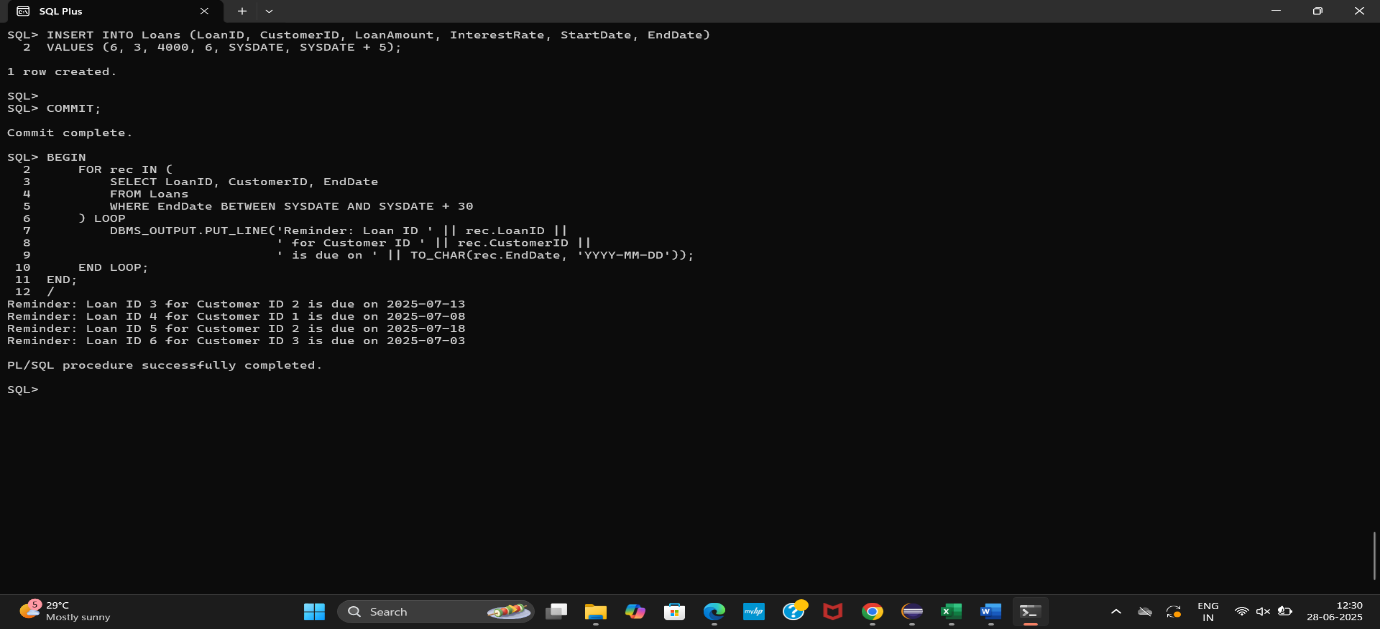
**Reminder: Loan ID 3 for Customer ID 2 is due on 2025-07-13**

**Reminder: Loan ID 4 for Customer ID 1 is due on 2025-07-08**

**Reminder: Loan ID 5 for Customer ID 2 is due on 2025-07-18**

**Reminder: Loan ID 6 for Customer ID 3 is due on 2025-07-03**

**PL/SQL procedure successfully completed.**

****

**Exercise 2: Error Handling**

**Question: Write a stored procedure SafeTransferFunds that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

-- Get source account balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

-- Check if sufficient balance

IF v\_balance < p\_amount THEN

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

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful!');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

**(procedure created successfully.)**

SET SERVEROUTPUT ON;

EXEC SafeTransferFunds(1, 2, 200);

**(Transfer successful!)**

EXEC SafeTransferFunds(1, 2, 5000);

**(Error: ORA-20001: Insufficient funds)**

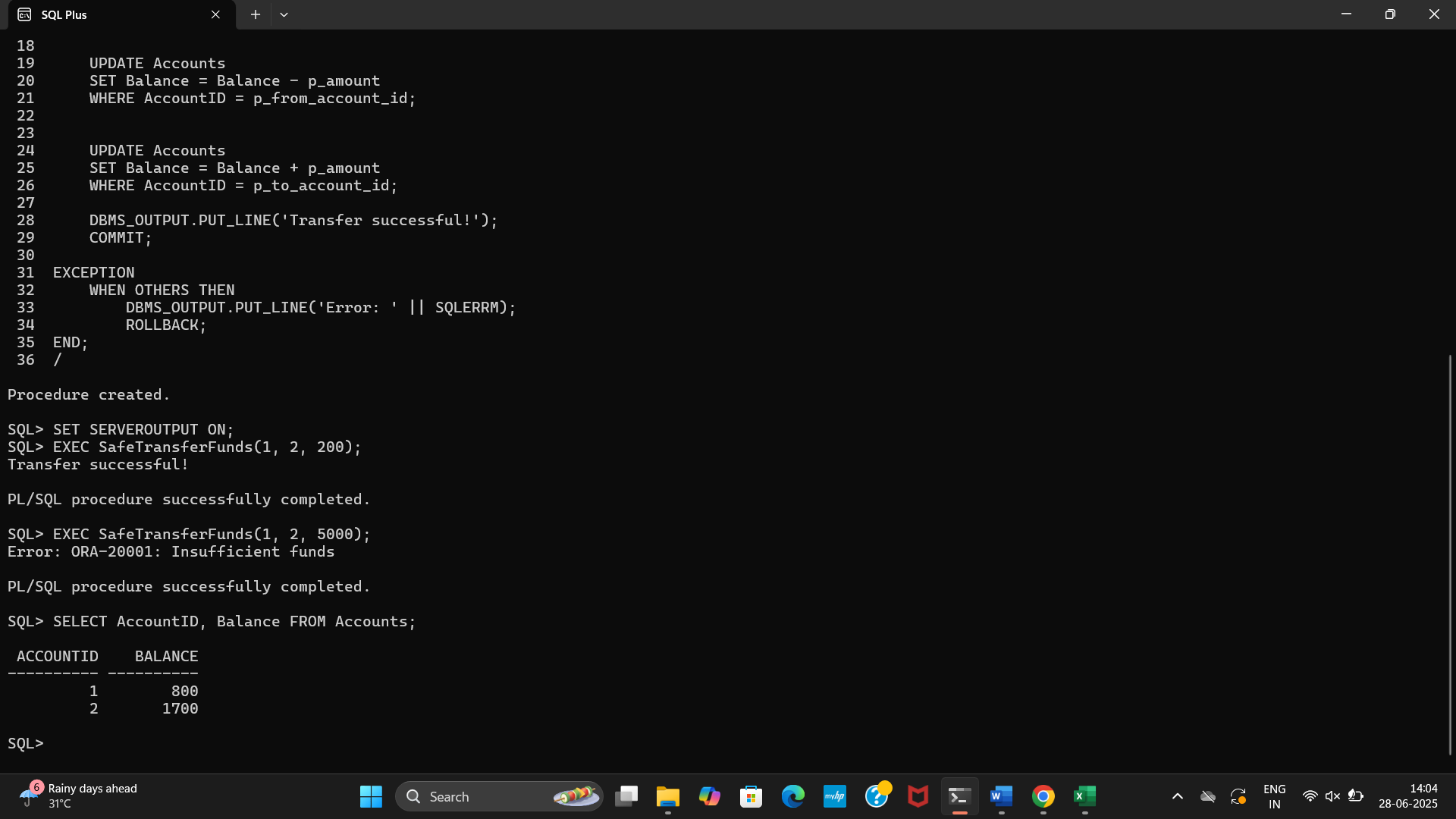
SELECT AccountID, Balance FROM Accounts;

**ACCOUNTID BALANCE**

**---------- ----------**

**1 800**

**2 1700**

****

**Question: Write a stored procedure UpdateSalary that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found');

END IF;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully!');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

**(Procedure created.)**

SET SERVEROUTPUT ON;

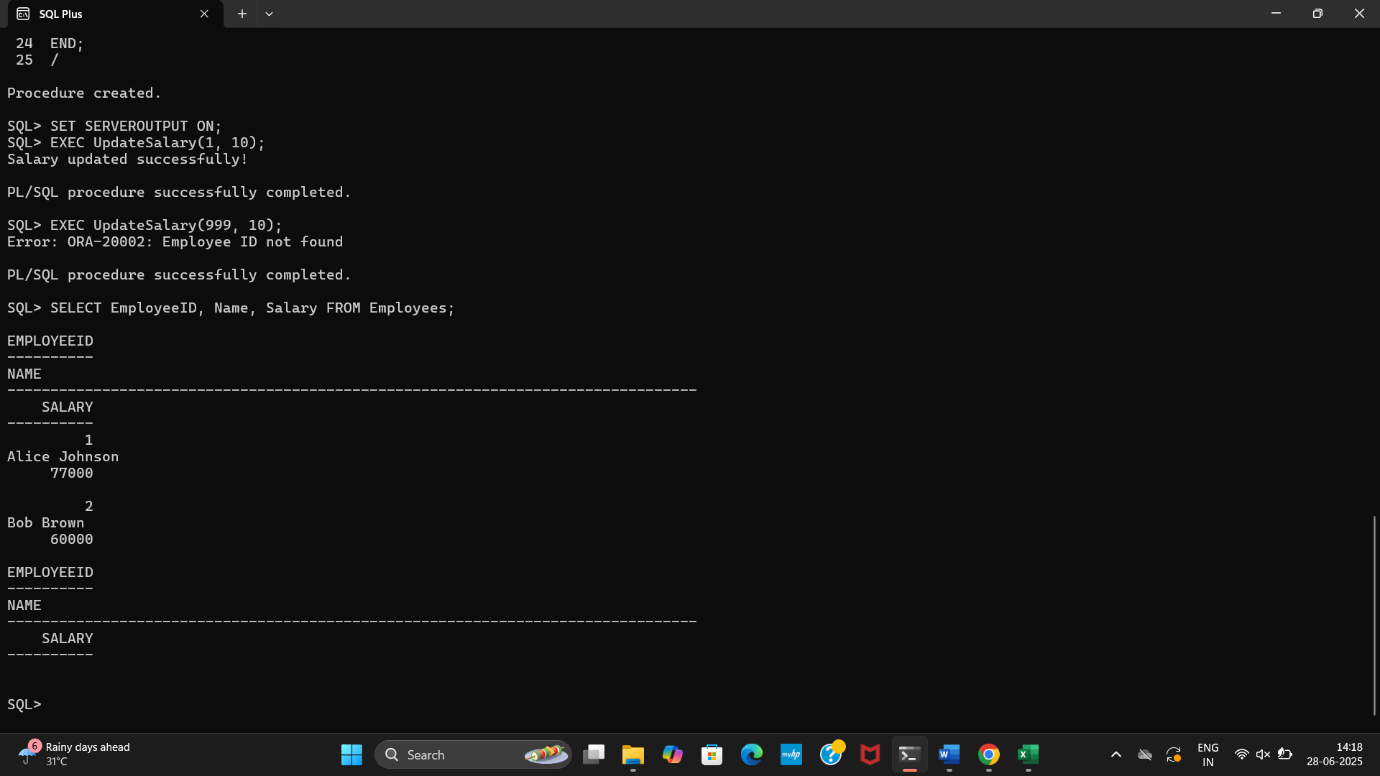
EXEC UpdateSalary(1, 10);

**(Salary updated successfully!)**

EXEC UpdateSalary(999, 10);

**(Error: ORA-20002: Employee ID not found)**

SELECT EmployeeID, Name, Salary FROM Employees;



**Question: Write a stored procedure AddNewCustomer that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

)

IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully!');

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists!');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

ROLLBACK;

END;

/

**(Procedure created)**

SET SERVEROUTPUT ON;

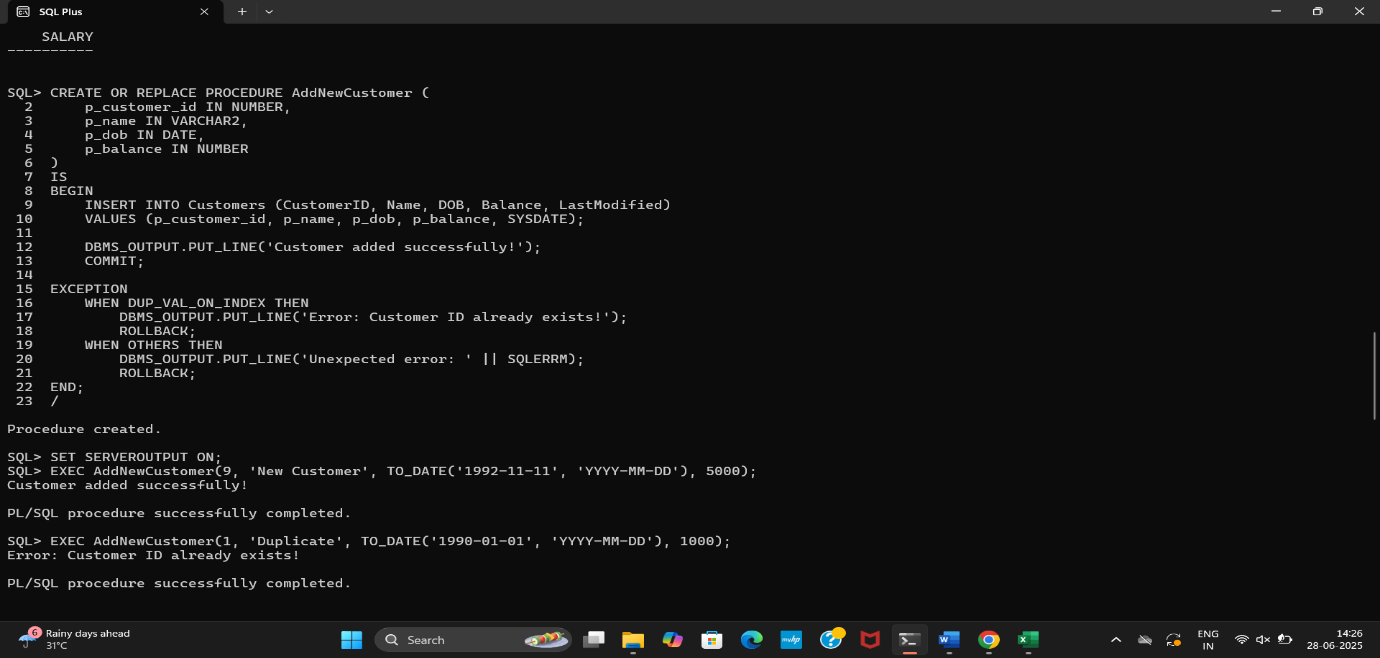
EXEC AddNewCustomer(9, 'New Customer', TO\_DATE('1992-11-11', 'YYYY-MM-DD'), 5000);

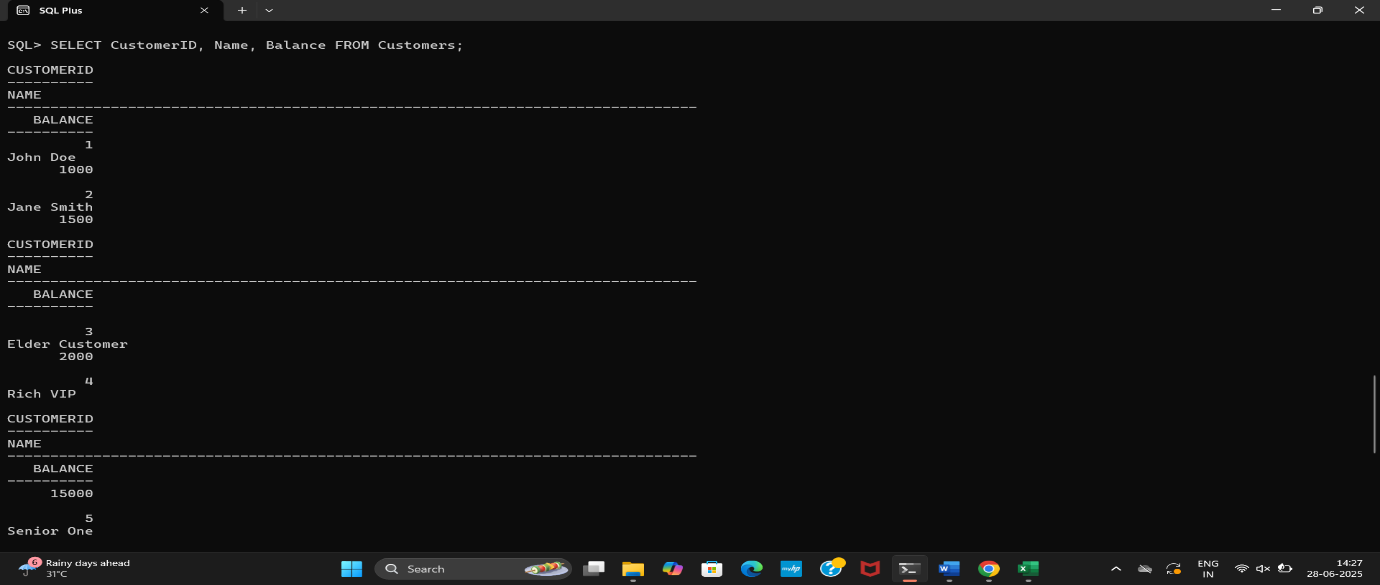
**(Customer added successfully!)**

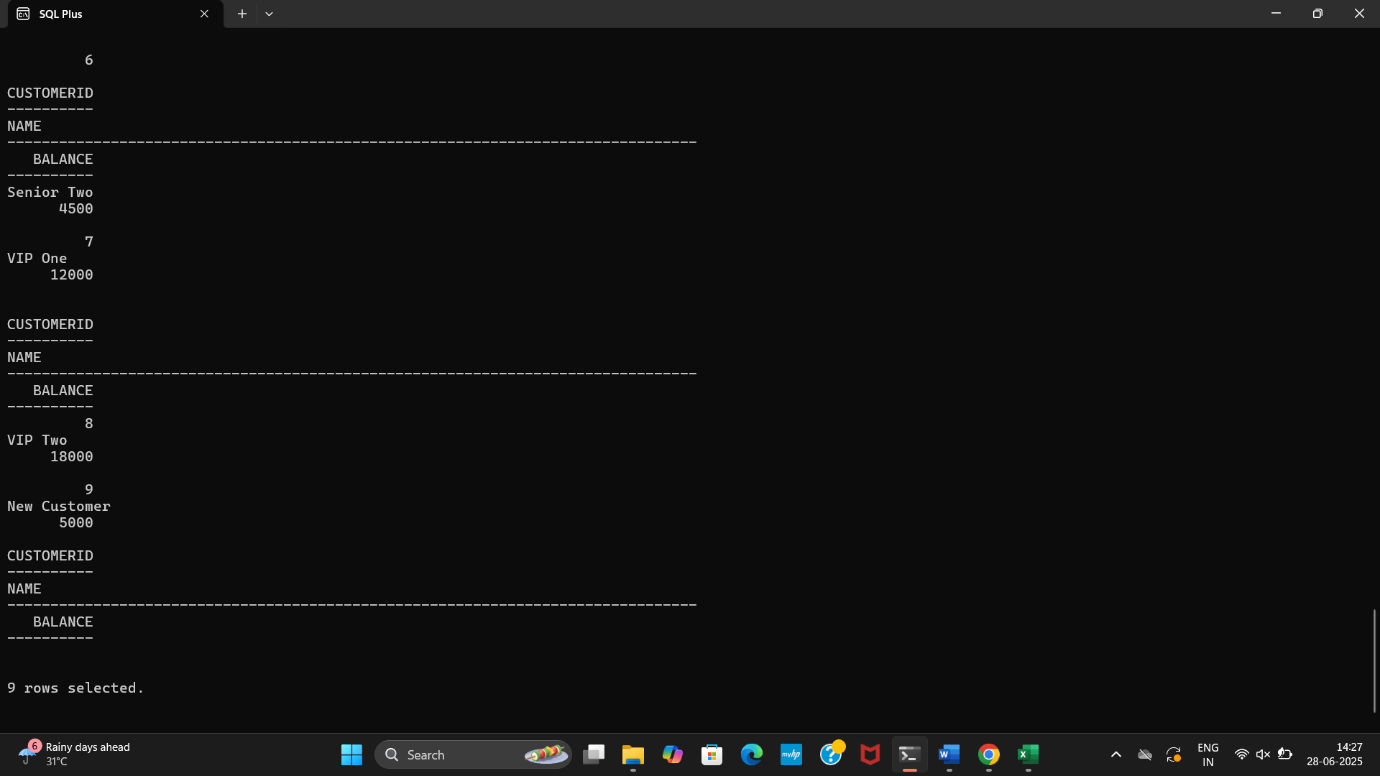
EXEC AddNewCustomer(1, 'Duplicate', TO\_DATE('1990-01-01', 'YYYY-MM-DD'), 1000);

**(Error: Customer ID already exists!)**

SELECT CustomerID, Name, Balance FROM Customers;







**Exercise 3: Stored Procedures**

**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 OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

FOR rec IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

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

WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || rec.AccountID);

END LOOP;

COMMIT;

END;

/

**(Procedure created.)**

SET SERVEROUTPUT ON;

EXEC ProcessMonthlyInterest;

**Interest applied to Account ID: 1**

**PL/SQL procedure successfully completed.**

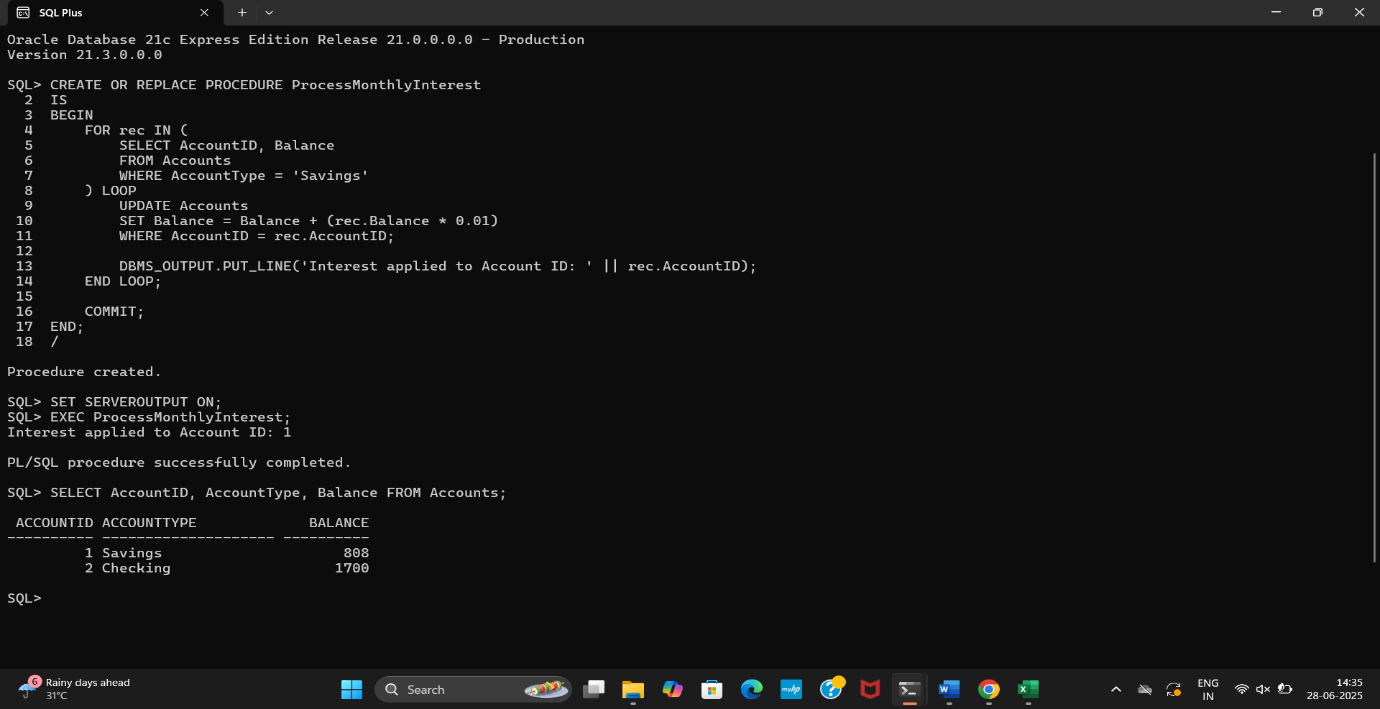
SELECT AccountID, AccountType, Balance FROM Accounts;

**ACCOUNTID ACCOUNTTYPE BALANCE**

**---------- -------------------- ----------**

**1 Savings 808**

**2 Checking 1700**



**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 OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

)

IS

BEGIN

FOR rec IN (

SELECT EmployeeID, Name, Salary

FROM Employees

WHERE Department = p\_department

) LOOP

UPDATE Employees

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

WHERE EmployeeID = rec.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || rec.Name ||

' (Employee ID: ' || rec.EmployeeID || ')');

END LOOP;

COMMIT;

END;

/

**(Procedure created)**

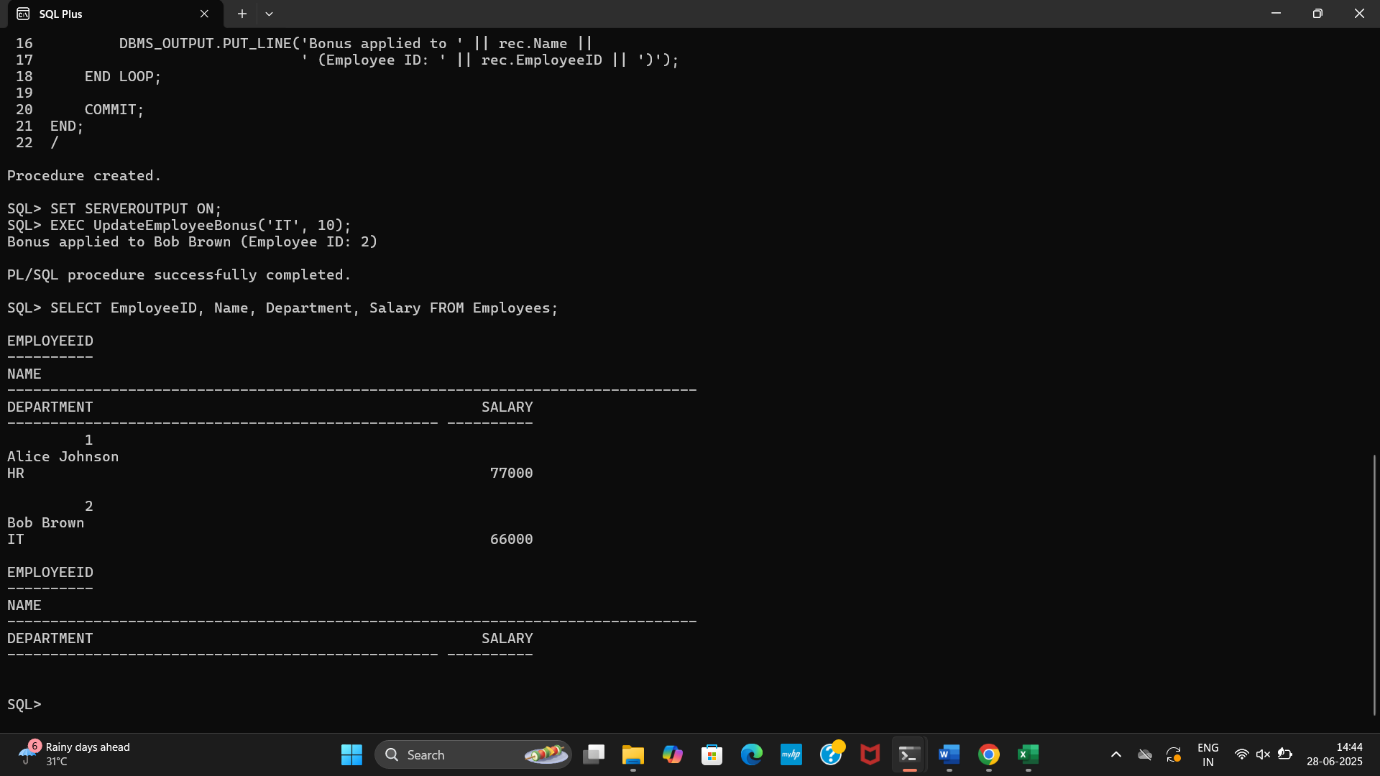
SET SERVEROUTPUT ON;

EXEC UpdateEmployeeBonus('IT', 10);

**Bonus applied to Bob Brown (Employee ID: 2)**

**PL/SQL procedure successfully completed.**

SELECT EmployeeID, Name, Department, Salary FROM Employees;



**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 OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance in source account.');

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;

DBMS\_OUTPUT.PUT\_LINE('Transfer completed successfully!');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

**(Procedure created.)**

SET SERVEROUTPUT ON;

EXEC TransferFunds(1, 2, 300);

**(Transfer completed successfully!)**

EXEC TransferFunds(1, 2, 100000);

**(Error: ORA-20003: Insufficient balance in source account.)**

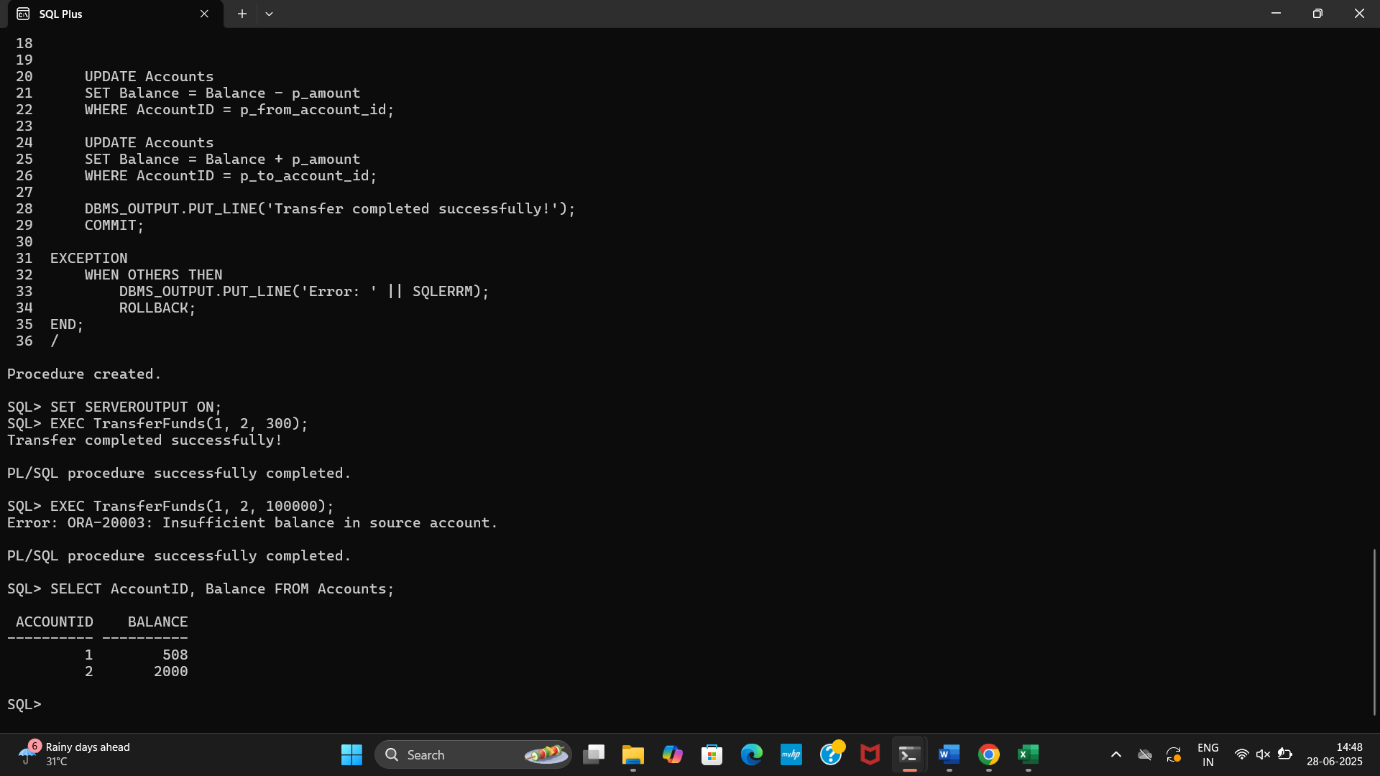
SELECT AccountID, Balance FROM Accounts;

**ACCOUNTID BALANCE**

**---------- ----------**

**1 508**

**2 2000**

****

**Exercise 4: Functions**

**Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

)

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

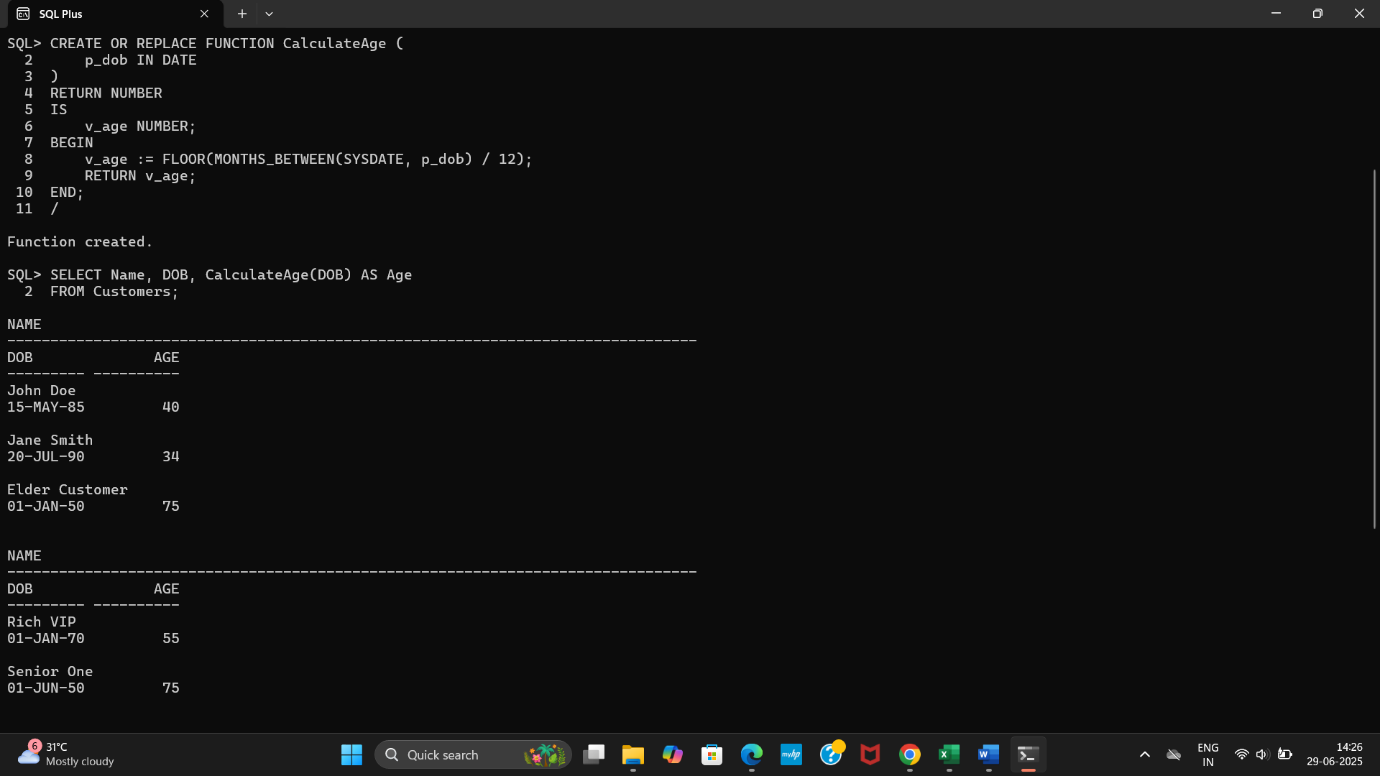
/

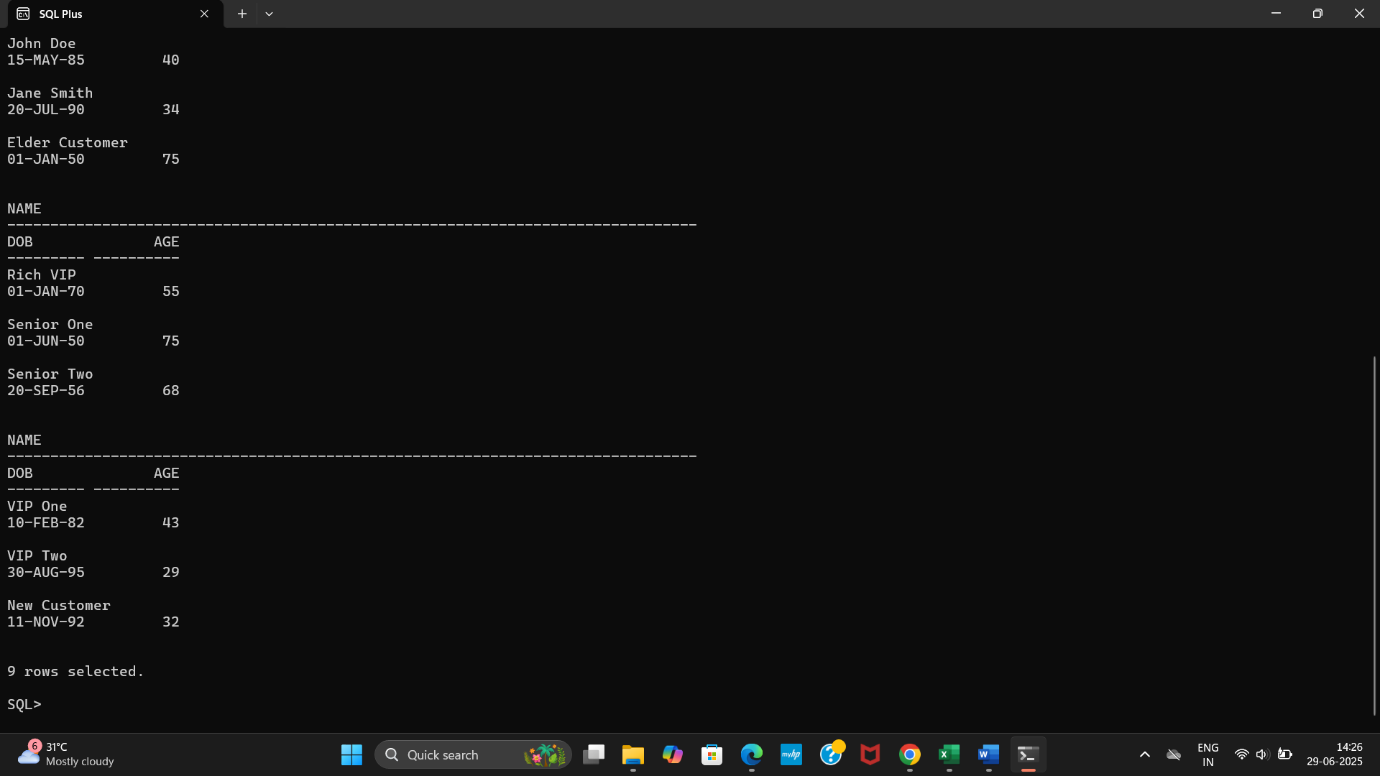
**(Function created.)**

SELECT Name, DOB, CalculateAge(DOB) AS Age

FROM Customers;

**OUTPUT**

****

****

**Question: Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_interest\_rate IN NUMBER,

p\_years IN NUMBER

)

RETURN NUMBER

IS

r NUMBER;

n NUMBER;

emi NUMBER;

BEGIN

r := p\_interest\_rate / 12 / 100;

n := p\_years \* 12;

emi := (p\_loan\_amount \* r \* POWER(1 + r, n)) /

(POWER(1 + r, n) - 1);

RETURN ROUND(emi, 2);

END;

/

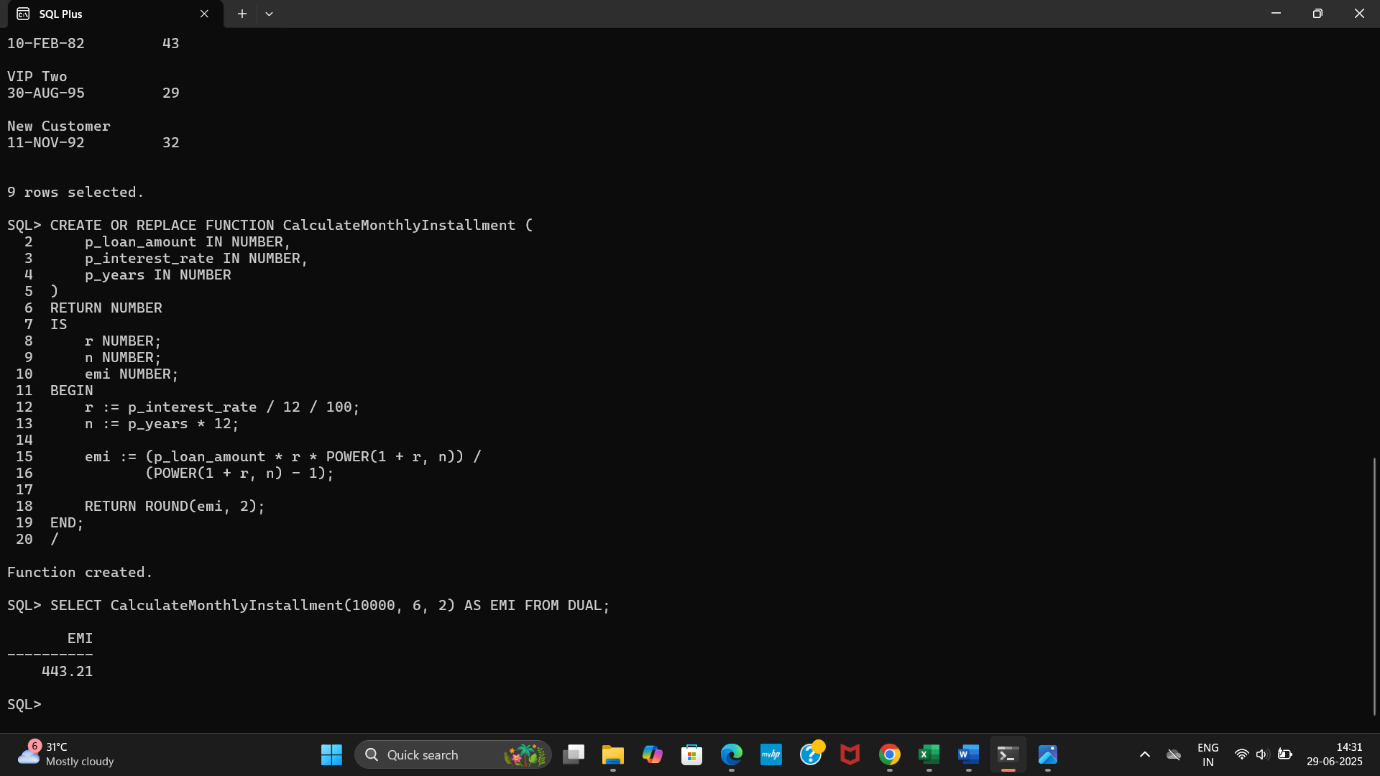
**(Function created.)**

SELECT CalculateMonthlyInstallment(10000, 6, 2) AS EMI FROM DUAL;

**EMI**

**----------**

**443.21**



**Question: Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

RETURN NUMBER

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

RETURN 0;

END;

/

**(Function created.)**

SELECT HasSufficientBalance(1, 200) AS Result FROM DUAL;

**RESULT**

**-------**

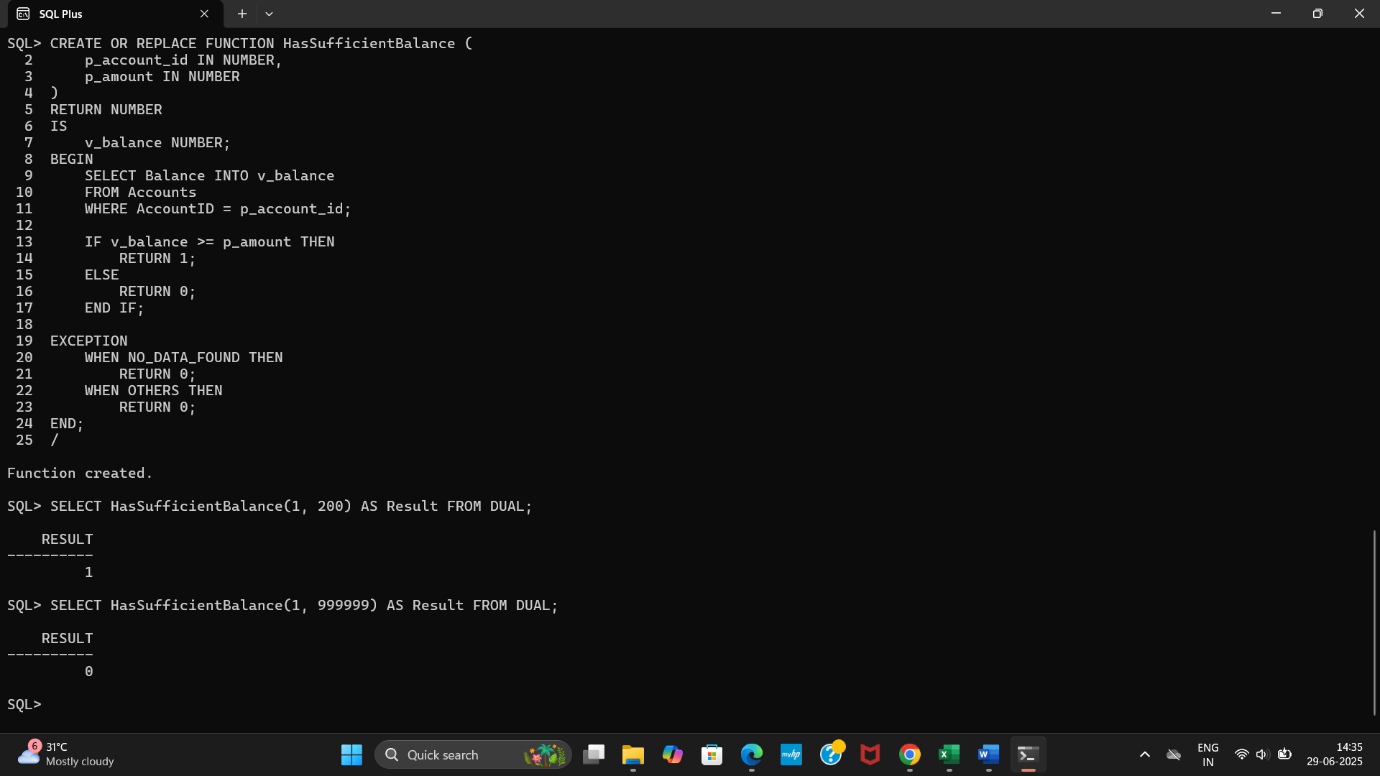
**1**

SELECT HasSufficientBalance(1, 999999) AS Result FROM DUAL;

**RESULT**

**-------**

**0**

****

**Exercise 5: Triggers**

**Question: Write a trigger UpdateCustomerLastModified that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**(Trigger created.)**

UPDATE Customers

SET Name = 'Updated John Doe'

WHERE CustomerID = 1;

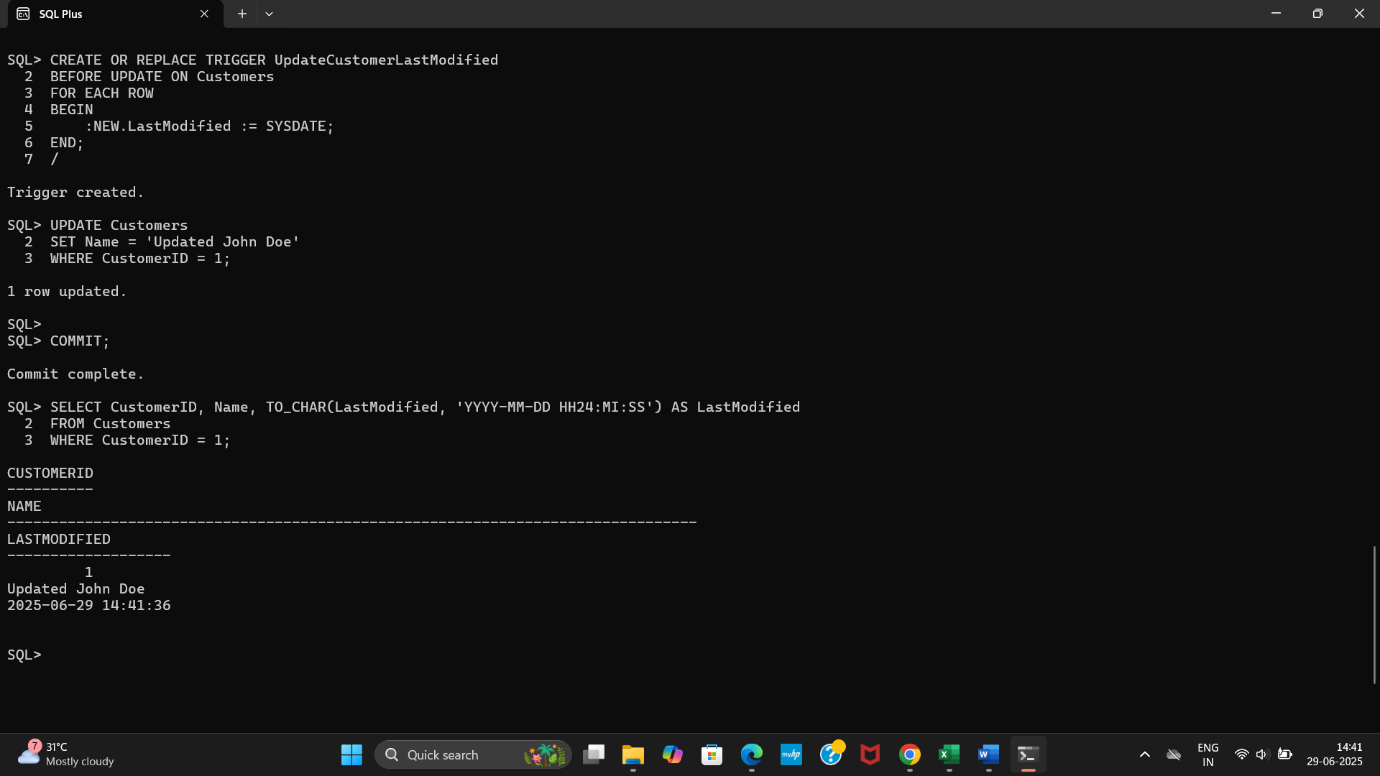
COMMIT;

**(Commit complete.)**

SELECT CustomerID, Name, TO\_CHAR(LastModified, 'YYYY-MM-DD HH24:MI:SS') AS LastModified

FROM Customers

WHERE CustomerID = 1;



**Question: Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

Amount NUMBER,

LoggedAt DATE

);

**(Table created.)**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, Amount, LoggedAt)

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.Amount, SYSDATE);

END;

/

**(Trigger Created.)**

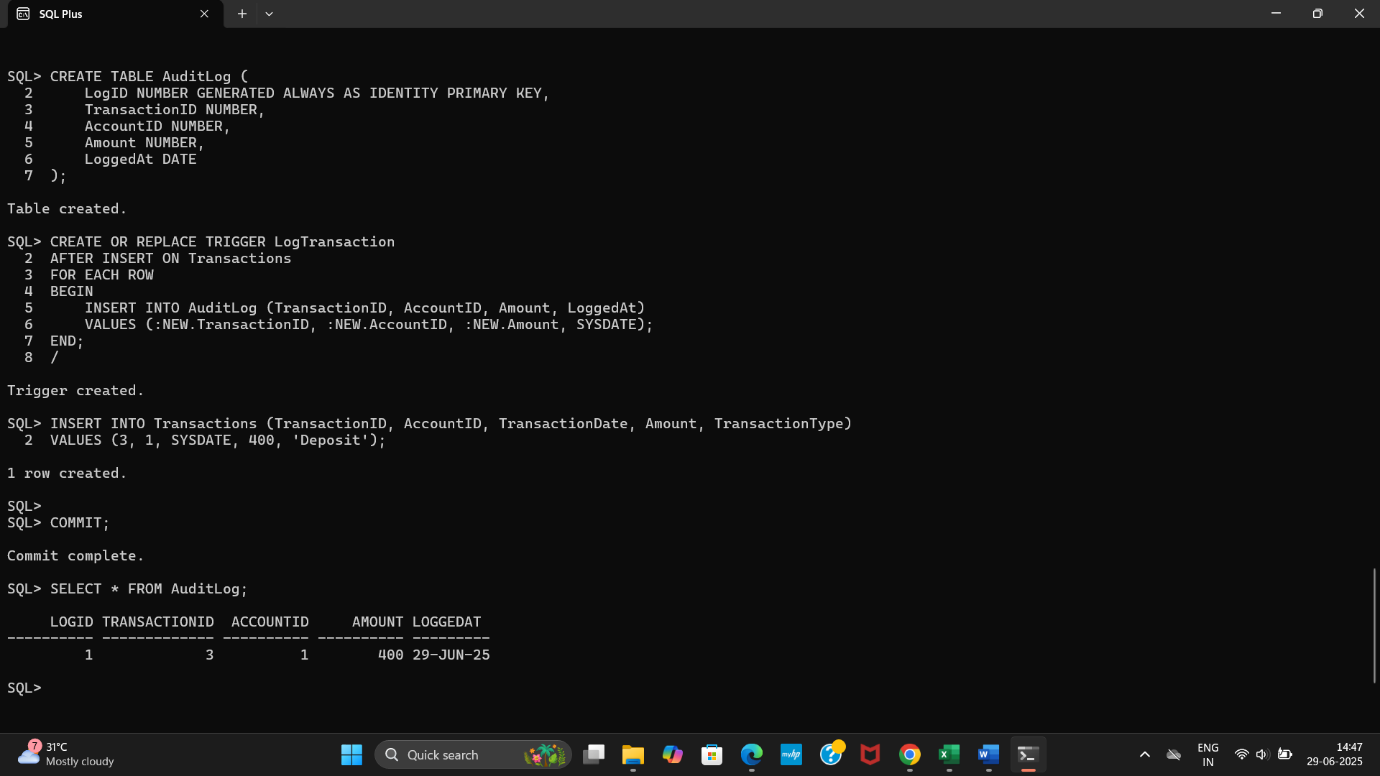
INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (3, 1, SYSDATE, 400, 'Deposit');

COMMIT;

**(Commit complete.)**

SELECT \* FROM AuditLog;



**Question: Write a trigger CheckTransactionRules that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20010, 'Withdrawal amount exceeds account balance.');

END IF;

IF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20011, 'Deposit amount must be positive.');

END IF;

END;

/

**(Trigger Created.)**

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (4, 1, SYSDATE, 100, 'Deposit');

COMMIT;

**(Commit complete.)**

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

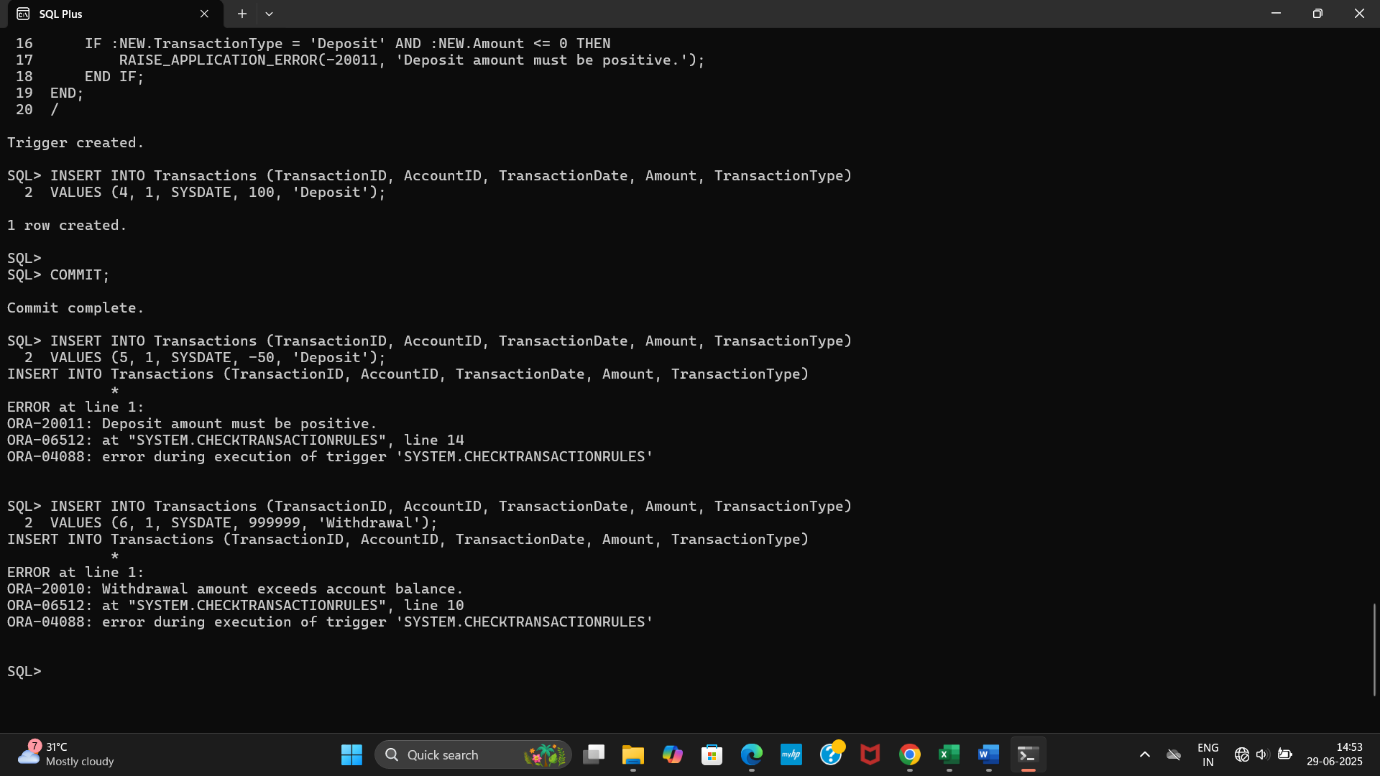
VALUES (5, 1, SYSDATE, -50, 'Deposit');

**(ORA-20011: Deposit amount must be positive.)**

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (6, 1, SYSDATE, 999999, 'Withdrawal');

**(ORA-20010: Withdrawal amount exceeds account balance.)**

****

**Exercise 6: Cursors**

**Question: Write a PL/SQL block using an explicit cursor GenerateMonthlyStatements that retrieves all transactions for the current month and prints a statement for each customer.**

SET SERVEROUTPUT ON;

DECLARE

CURSOR cur\_monthly\_txns IS

SELECT t.TransactionID, t.AccountID, t.TransactionDate,

t.Amount, t.TransactionType, c.Name

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE TO\_CHAR(t.TransactionDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY');

v\_txn cur\_monthly\_txns%ROWTYPE;

BEGIN

OPEN cur\_monthly\_txns;

LOOP

FETCH cur\_monthly\_txns INTO v\_txn;

EXIT WHEN cur\_monthly\_txns%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(

'Customer: ' || v\_txn.Name ||

' | Txn ID: ' || v\_txn.TransactionID ||

' | Type: ' || v\_txn.TransactionType ||

' | Amount: ' || v\_txn.Amount ||

' | Date: ' || TO\_CHAR(v\_txn.TransactionDate, 'DD-Mon-YYYY')

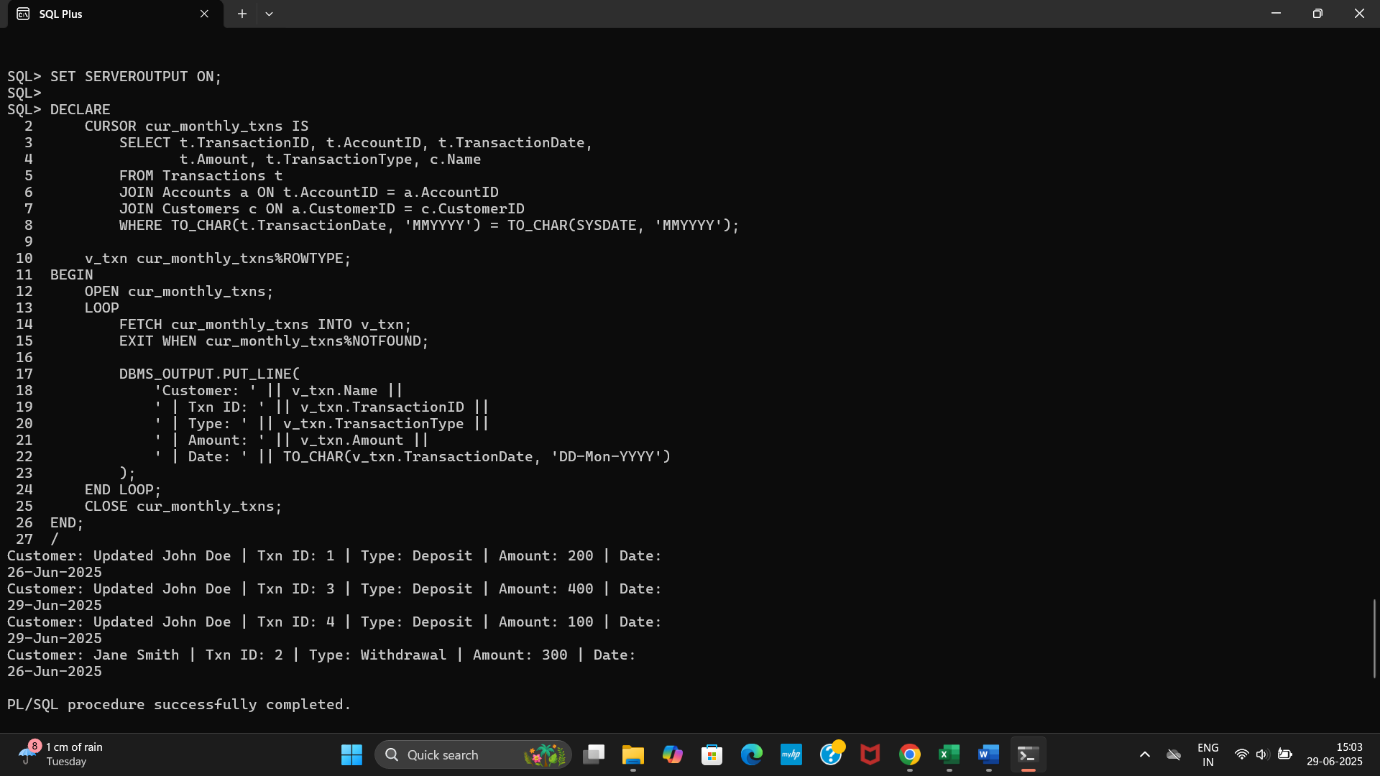
);

END LOOP;

CLOSE cur\_monthly\_txns;

END;

/



**Write a PL/SQL block using an explicit cursor ApplyAnnualFee that deducts an annual maintenance fee from the balance of all accounts.**

SET SERVEROUTPUT ON;

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_account cur\_accounts%ROWTYPE;

v\_fee CONSTANT NUMBER := 100;

BEGIN

OPEN cur\_accounts;

LOOP

FETCH cur\_accounts INTO v\_account;

EXIT WHEN cur\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = v\_account.AccountID;

DBMS\_OUTPUT.PUT\_LINE(

'Annual fee of ₹' || v\_fee || ' deducted from Account ID: ' || v\_account.AccountID

);

END LOOP;

CLOSE cur\_accounts;

COMMIT;

END;

/

**Annual fee of ₹100 deducted from Account ID: 1**

**Annual fee of ₹100 deducted from Account ID: 2**

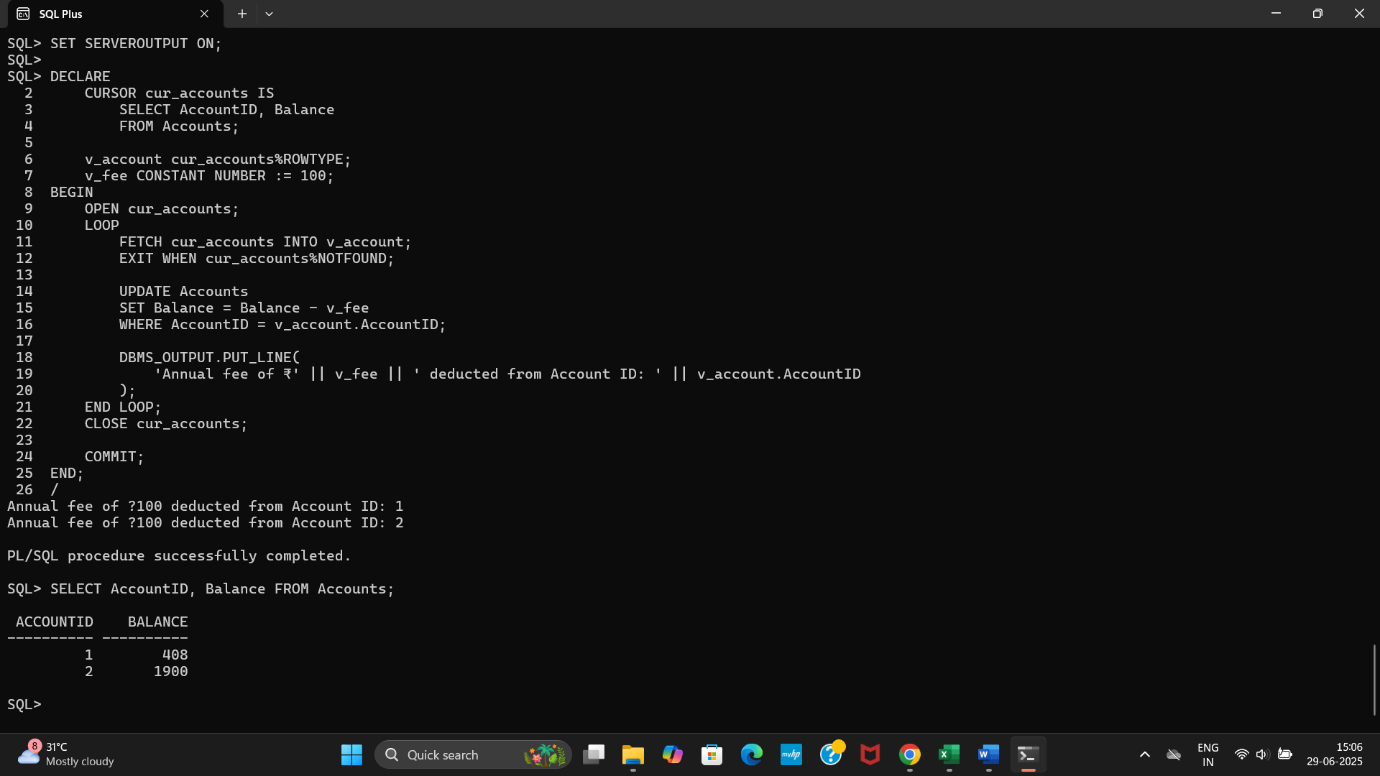
SELECT AccountID, Balance FROM Accounts;

**ACCOUNTID BALANCE**

**---------- ----------**

**1 408**

**2 1900**

****

**Question: Write a PL/SQL block using an explicit cursor UpdateLoanInterestRates that fetches all loans and updates their interest rates based on the new policy.**

SET SERVEROUTPUT ON;

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, LoanAmount, InterestRate

FROM Loans;

v\_loan cur\_loans%ROWTYPE;

v\_new\_rate NUMBER;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_loan;

EXIT WHEN cur\_loans%NOTFOUND;

IF v\_loan.LoanAmount < 5000 THEN

v\_new\_rate := 6;

ELSIF v\_loan.LoanAmount BETWEEN 5000 AND 10000 THEN

v\_new\_rate := 5;

ELSE

v\_new\_rate := 4.5;

END IF;

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = v\_loan.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

'Loan ID: ' || v\_loan.LoanID ||

' | Old Rate: ' || v\_loan.InterestRate ||

' | New Rate: ' || v\_new\_rate

);

END LOOP;

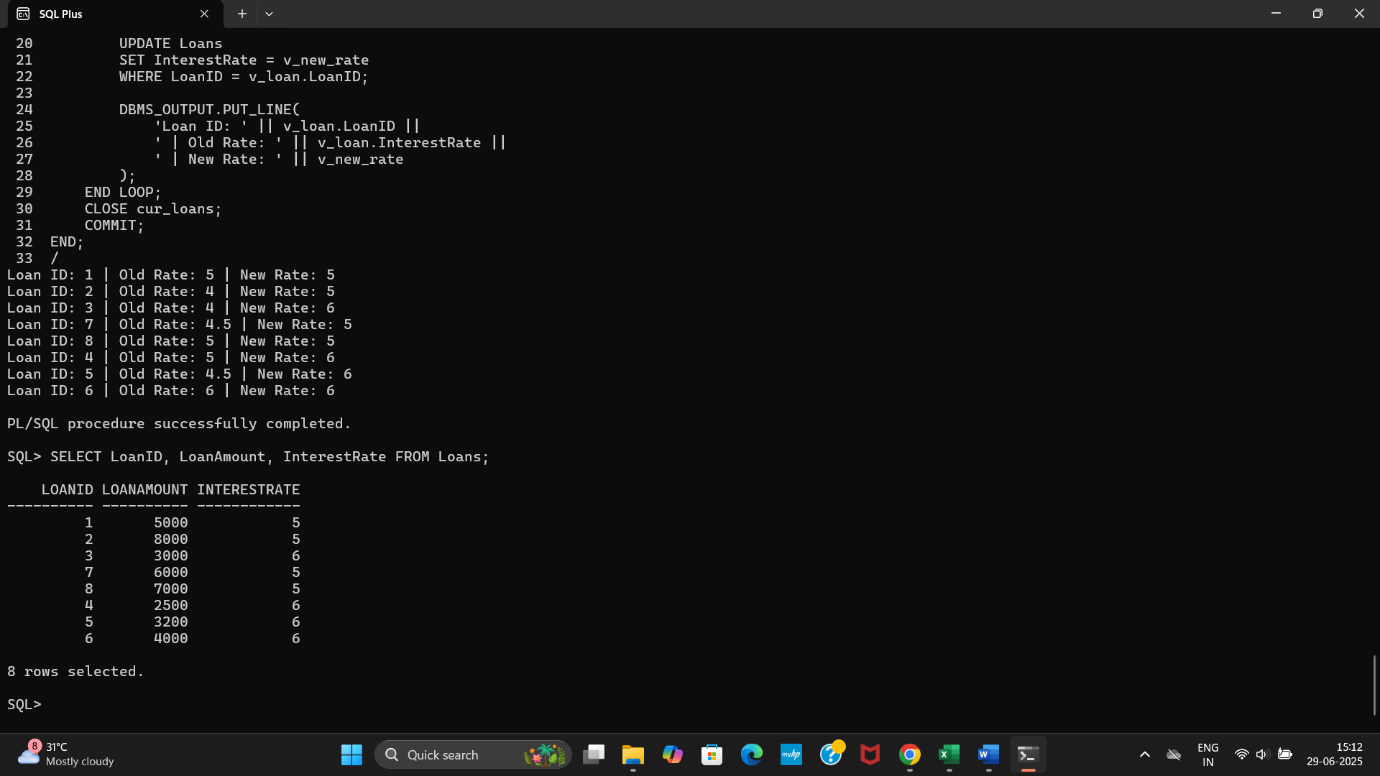
CLOSE cur\_loans;

COMMIT;

END;

/

SELECT LoanID, LoanAmount, InterestRate FROM Loans;



**Exercise 7: Packages**

**Question: Create a package CustomerManagement with procedures for adding a new customer, updating customer details, and a function to get customer balance.**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(

p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER

);

PROCEDURE UpdateCustomer(

p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER

);

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

**(Package created.)**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(

p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER

) IS

BEGIN

INSERT INTO Customers(CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

PROCEDURE UpdateCustomer(

p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER

) IS

BEGIN

UPDATE Customers

SET Name = p\_name,

DOB = p\_dob,

Balance = p\_balance,

LastModified = SYSDATE

WHERE CustomerID = p\_id;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END CustomerManagement;

/

**(Package body created.)**

EXEC CustomerManagement.AddCustomer(3, 'Ravi Kumar', TO\_DATE('1988-03-15','YYYY-MM-DD'), 7500);

**(PL/SQL procedure successfully completed.)**

EXEC CustomerManagement.UpdateCustomer(3, 'Ravi K.', TO\_DATE('1988-03-15','YYYY-MM-DD'), 8000);

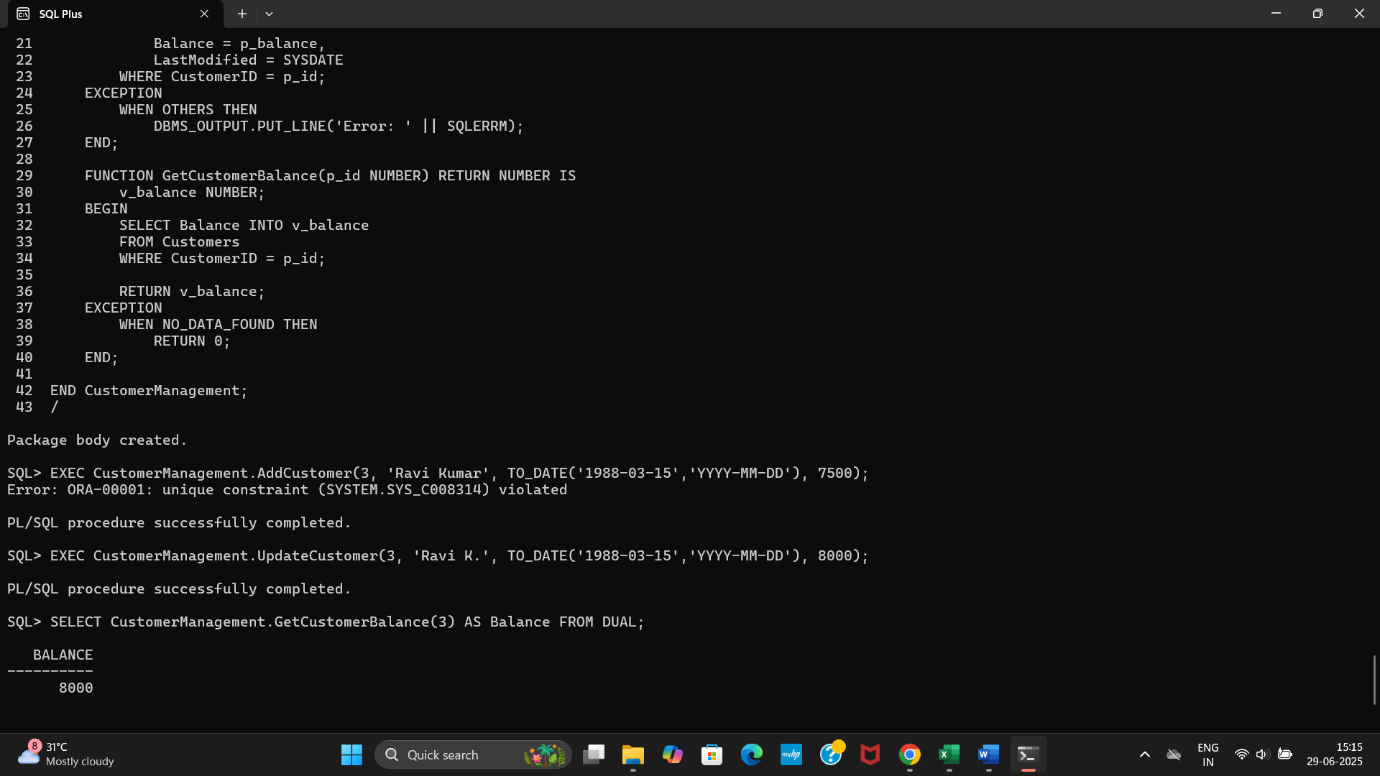
**(PL/SQL procedure successfully completed.)**

SELECT CustomerManagement.GetCustomerBalance(3) AS Balance FROM DUAL;

**BALANCE**

**----------**

**8000**

****

**Question: Write a package EmployeeManagement with procedures to hire new employees, update employee details, and a function to calculate annual salary.**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(

p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2,

p\_salary NUMBER, p\_dept VARCHAR2, p\_hiredate DATE

);

PROCEDURE UpdateEmployee(

p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2,

p\_salary NUMBER, p\_dept VARCHAR2

);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

**(Package created.)**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(

p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2,

p\_salary NUMBER, p\_dept VARCHAR2, p\_hiredate DATE

) IS

BEGIN

INSERT INTO Employees(EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_id, p\_name, p\_position, p\_salary, p\_dept, p\_hiredate);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

PROCEDURE UpdateEmployee(

p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2,

p\_salary NUMBER, p\_dept VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Name = p\_name,

Position = p\_position,

Salary = p\_salary,

Department = p\_dept

WHERE EmployeeID = p\_id;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END EmployeeManagement;

/

**(Package body created.)**

EXEC EmployeeManagement.HireEmployee(3, 'Kiran Reddy', 'Analyst', 50000, 'Finance', SYSDATE);

**(PL/SQL procedure successfully completed.)**

EXEC EmployeeManagement.UpdateEmployee(3, 'Kiran R.', 'Senior Analyst', 55000, 'Finance');

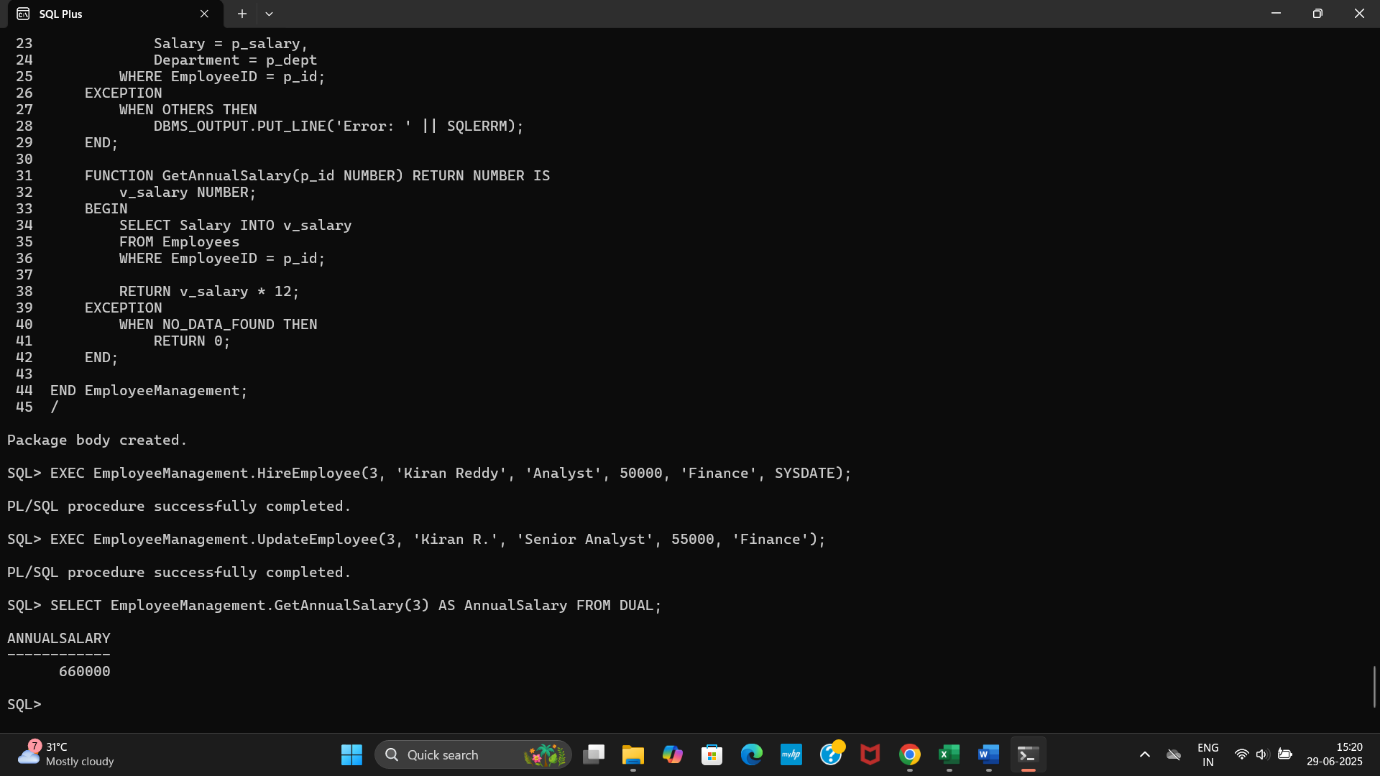
**(PL/SQL procedure successfully completed.)**

SELECT EmployeeManagement.GetAnnualSalary(3) AS AnnualSalary FROM DUAL;

**ANNUALSALARY**

**------------**

**660000**

****

**Question: Create a package AccountOperations with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_type VARCHAR2,

p\_balance NUMBER

);

PROCEDURE CloseAccount(p\_account\_id NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

**(Package created.)**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_type VARCHAR2,

p\_balance NUMBER

) IS

BEGIN

INSERT INTO Accounts(AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_account\_id, p\_customer\_id, p\_type, p\_balance, SYSDATE);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

PROCEDURE CloseAccount(p\_account\_id NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0)

INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

END;

END AccountOperations;

/

**(Package body created.)**

EXEC AccountOperations.OpenAccount(3, 1, 'Current', 2000);

**(PL/SQL procedure successfully completed.)**

EXEC AccountOperations.CloseAccount(3);

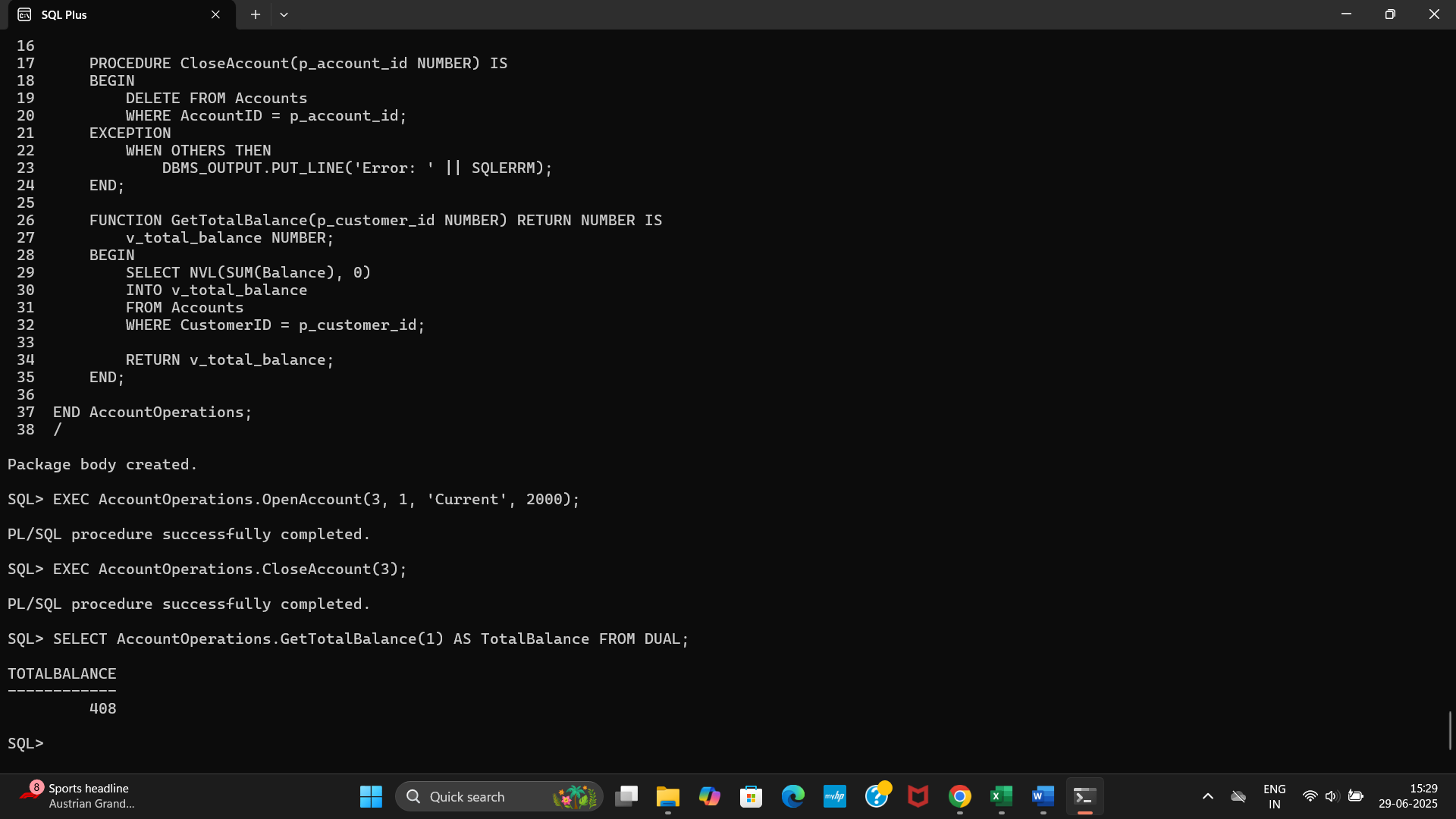
**(PL/SQL procedure successfully completed.)**

SELECT AccountOperations.GetTotalBalance(1) AS TotalBalance FROM DUAL;

**TOTALBALANCE**

**------------**

**408**

****