

Exercise 1: Control Structures

BEGIN

FOR cust IN (SELECT \* FROM Customers) LOOP

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

UPDATE Loans SET InterestRate = InterestRate - 1 WHERE CustomerID = cust.CustomerID;

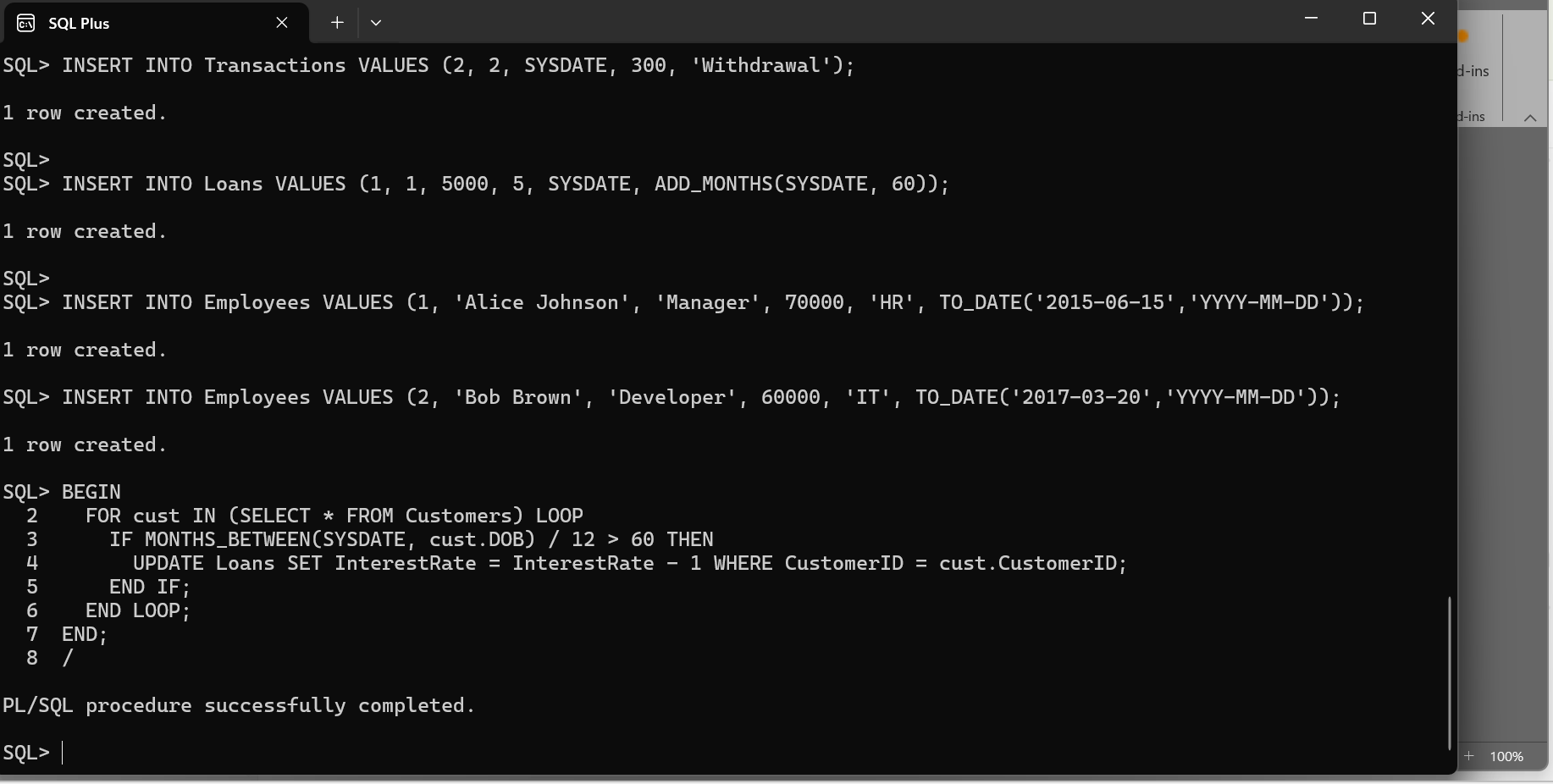
END IF;

END LOOP;

END;

/

**Output**

****

**Exercise 2: Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_balance EXCEPTION;

v\_balance NUMBER;

BEGIN

-- Check balance in source account

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

RAISE insufficient\_balance;

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;

COMMIT;

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

EXCEPTION

WHEN insufficient\_balance THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance for transfer.');

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

v\_salary NUMBER;

BEGIN

-- Fetch current salary to verify employee exists

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

-- Update salary

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee not found.');

WHEN OTHERS THEN

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

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_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\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

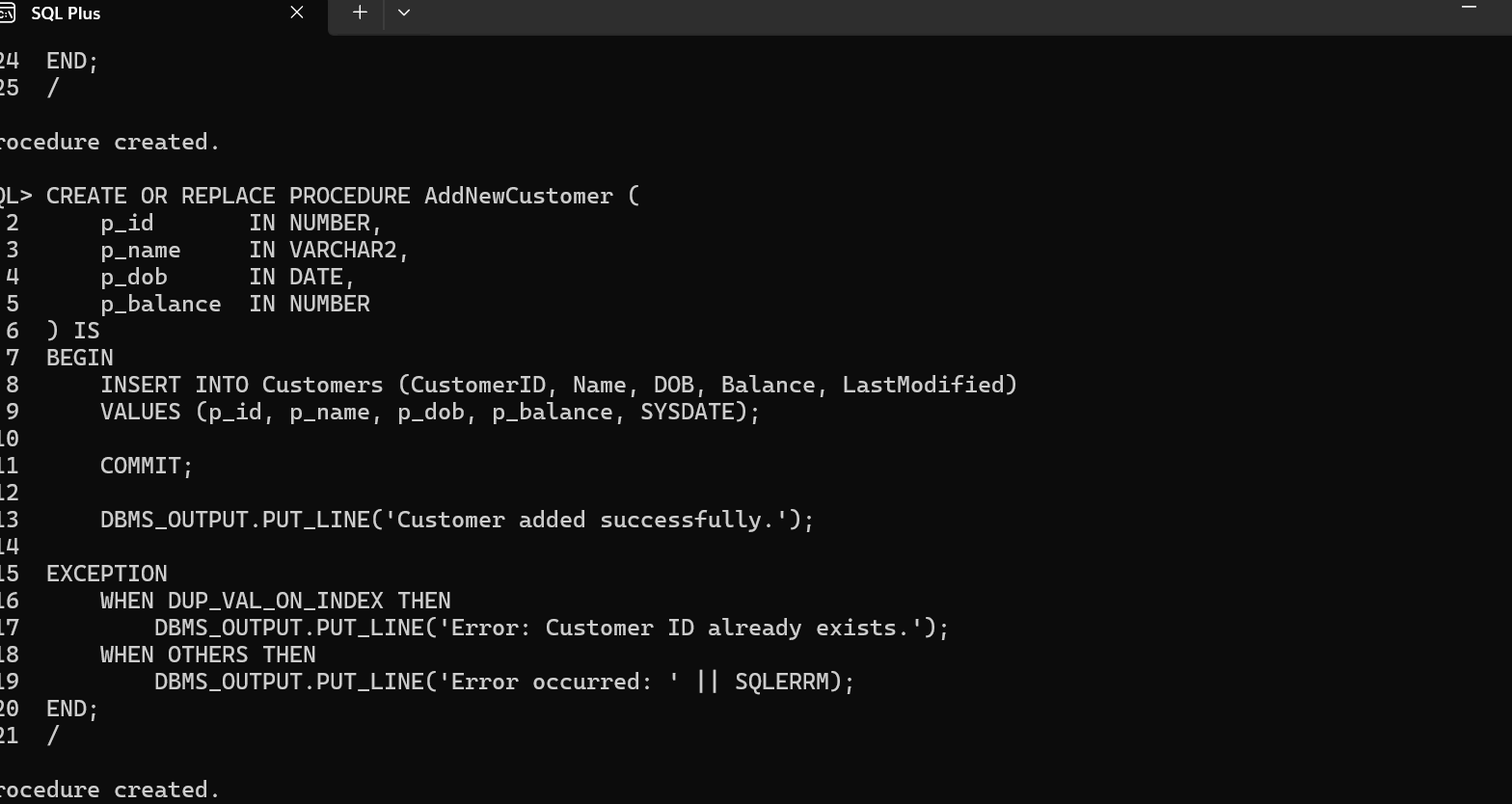
WHEN OTHERS THEN

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

END;

/

**Output**

****

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

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

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus updated for ' || p\_department || ' department.');

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from NUMBER,

p\_to NUMBER,

p\_amt NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from;

IF v\_balance < p\_amt THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_amt WHERE AccountID = p\_from;

UPDATE Accounts SET Balance = Balance + p\_amt WHERE AccountID = p\_to;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN OTHERS THEN

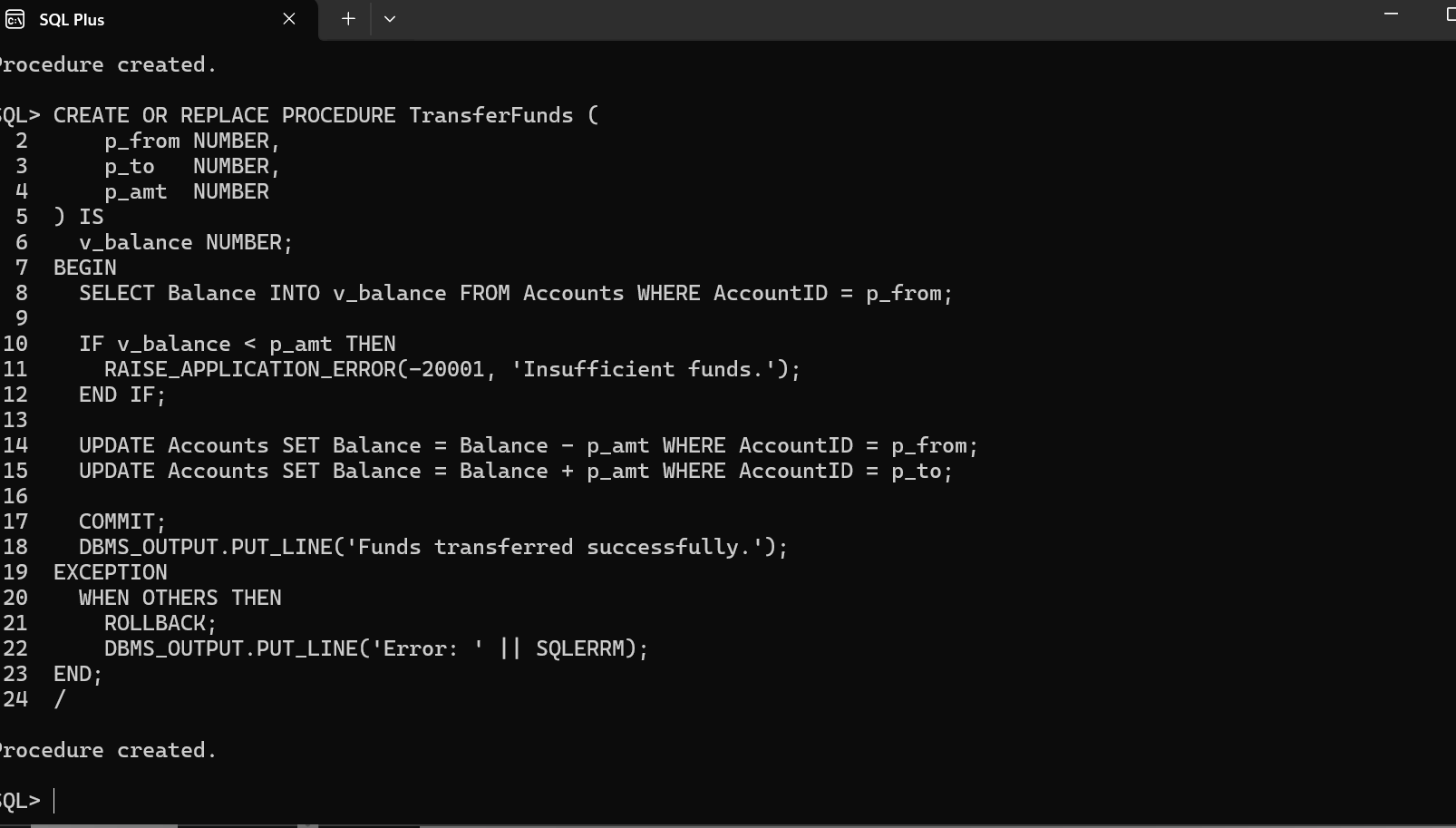
ROLLBACK;

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

END;

/

**Output :**



**Exercise 4: Functions**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob DATE

) RETURN NUMBER IS

BEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);

END;

/

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_amount NUMBER,

p\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

v\_months NUMBER := p\_years \* 12;

v\_r NUMBER := p\_rate / (12 \* 100);

BEGIN

RETURN ROUND(p\_amount \* v\_r / (1 - POWER(1 + v\_r, -v\_months)), 2);

END;

/

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

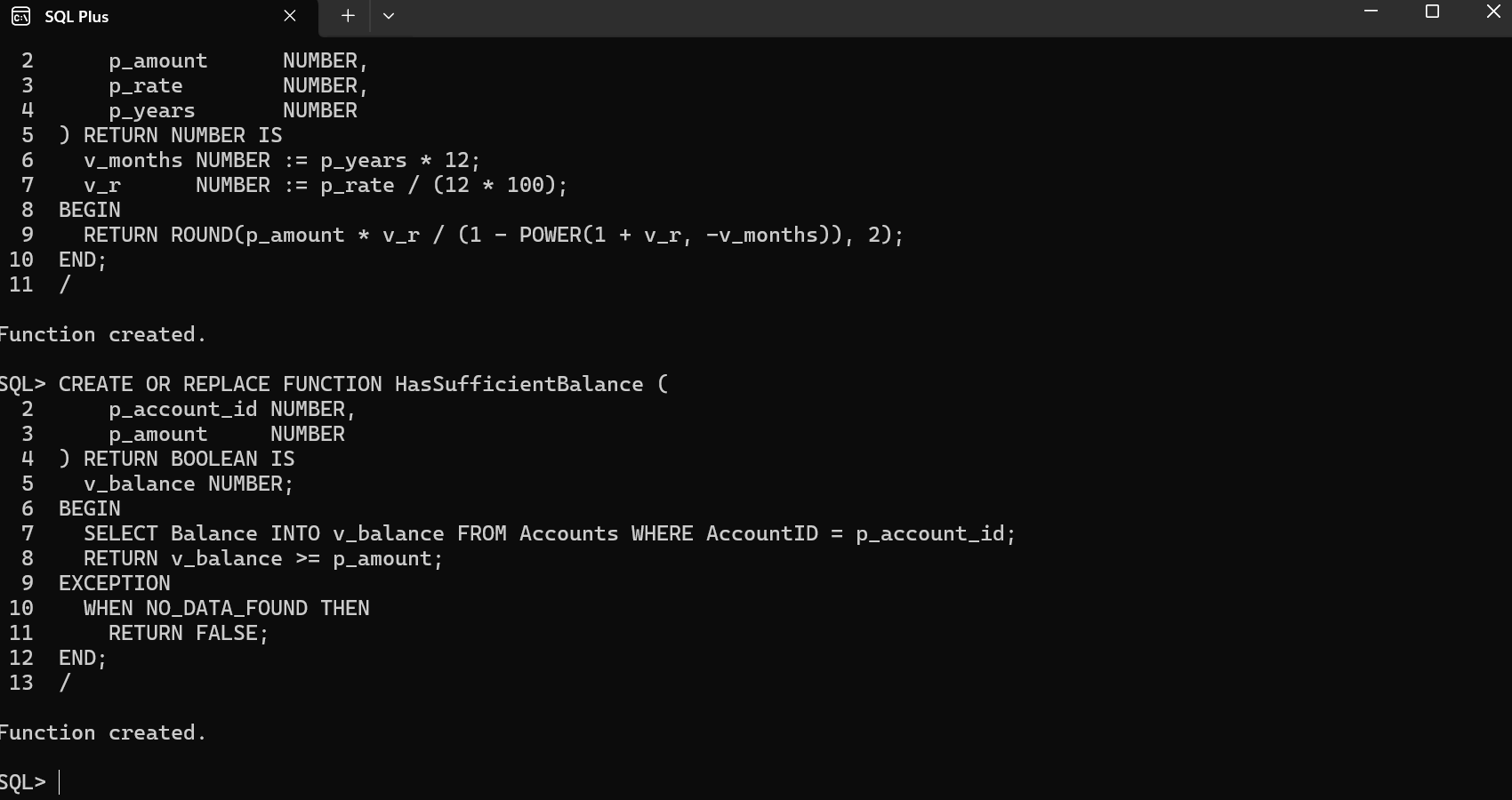
WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

**Output:**

****

**Exercise 5: Triggers**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2:**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY,

AccountID NUMBER,

Action VARCHAR2(50),

LogTime DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AccountID, Action, LogTime)

VALUES (:NEW.AccountID, 'Transaction Inserted', SYSDATE);

END;

/

**Scenario 3:**

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(-20002, 'Withdrawal exceeds balance.');

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

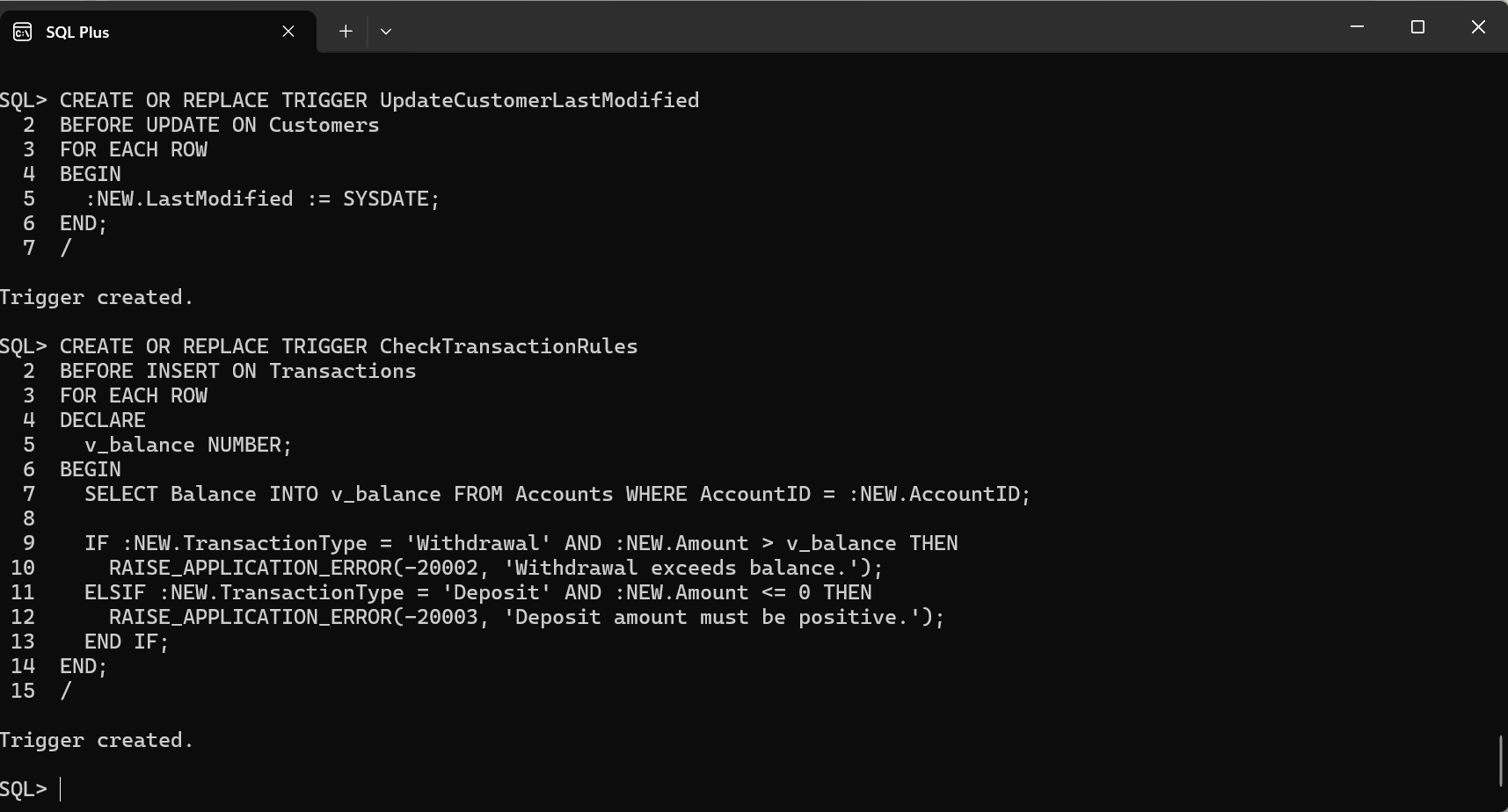
RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be positive.');

END IF;

END;

/

**Output:**

****

**Exercise 6: Cursors**

**Scenario 1:**

BEGIN

FOR rec IN (

SELECT A.CustomerID, T.TransactionDate, T.Amount, T.TransactionType

FROM Transactions T

JOIN Accounts A ON A.AccountID = T.AccountID

WHERE EXTRACT(MONTH FROM T.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ': ' || rec.TransactionType || ' of $' || rec.Amount);

END LOOP;

END;

/

**Scenario 2:**

DECLARE

CURSOR fee\_cur IS SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR rec IN fee\_cur LOOP

UPDATE Accounts

SET Balance = Balance - 100, LastModified = SYSDATE

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Annual fee applied.');

END;

/

**Scenario 3:**

DECLARE

CURSOR loan\_cur IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR rec IN loan\_cur LOOP

UPDATE Loans

SET InterestRate = rec.InterestRate + 0.5

WHERE LoanID = rec.LoanID;

END LOOP;

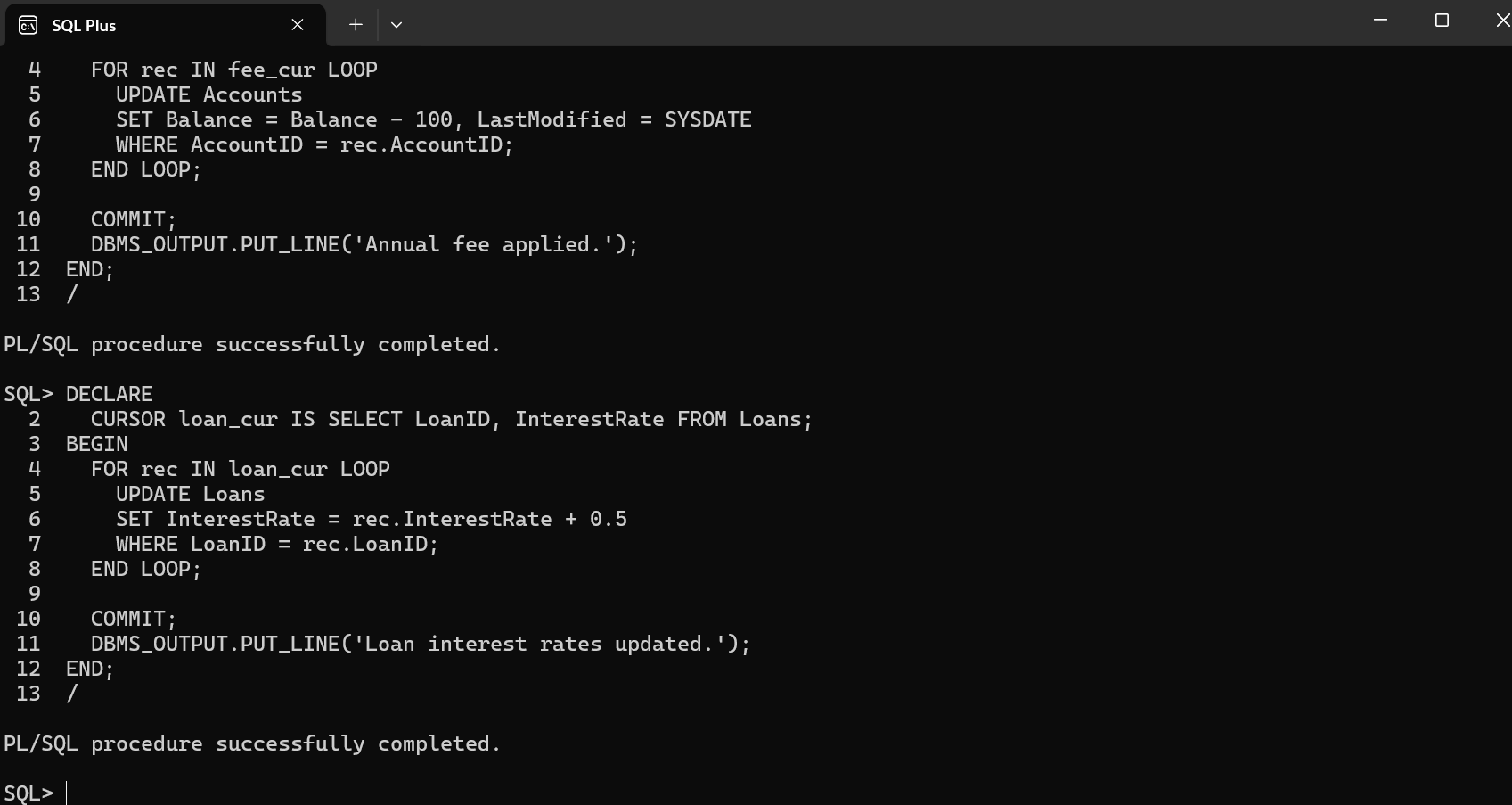
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Loan interest rates updated.');

END;

/

**Output:**

****

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

END;

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_id;

RETURN v\_balance;

END;

END;

/

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_date DATE);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_date DATE) IS

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_pos, p\_salary, p\_dept, p\_date);

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;

END;

END;

/

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_id NUMBER);

FUNCTION TotalBalance(p\_custid NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_id, p\_custid, p\_type, p\_balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id;

END;

FUNCTION TotalBalance(p\_custid NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;

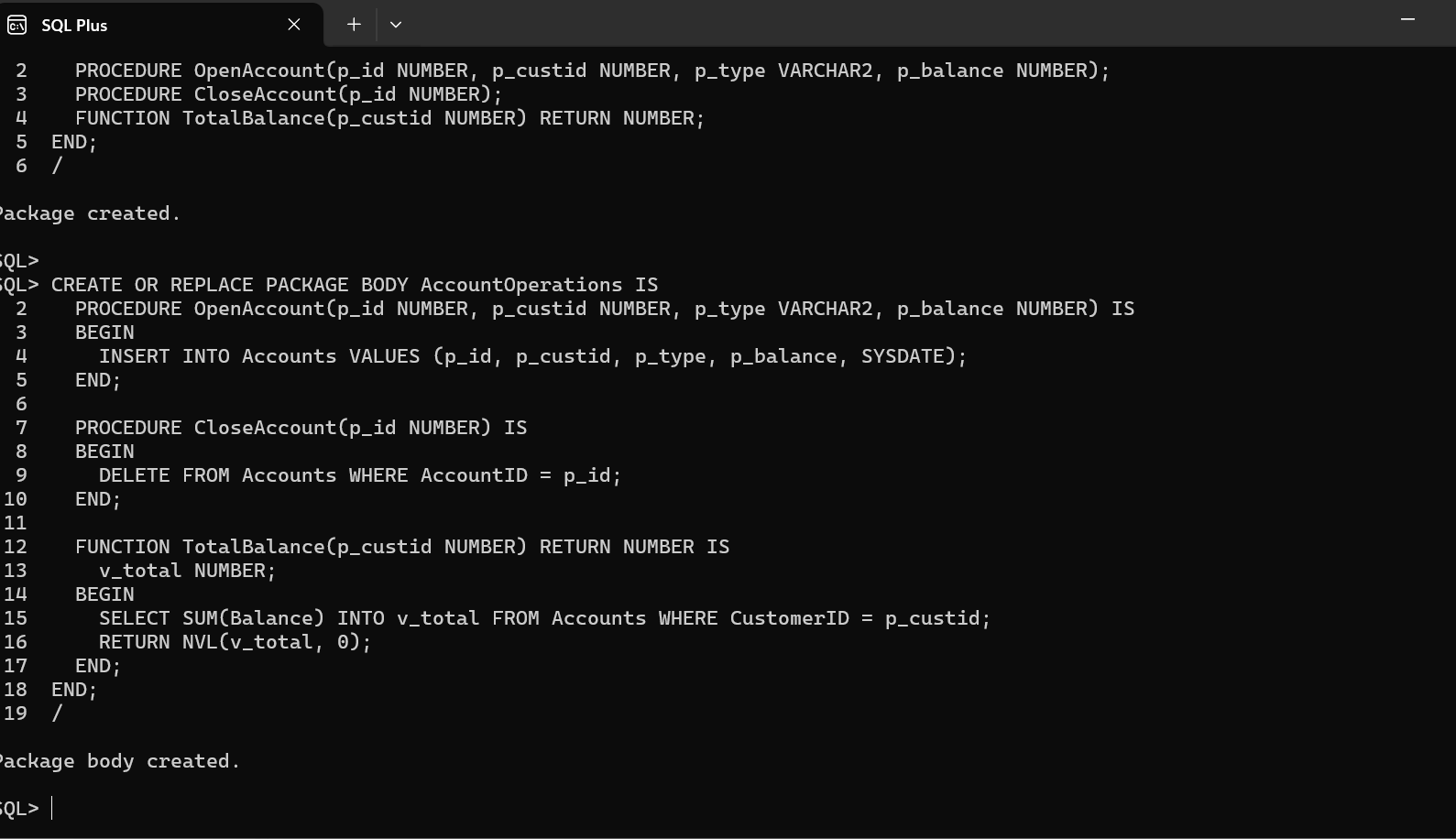
RETURN NVL(v\_total, 0);

END;

END;

/

**Output:**

****