**WEEK 2**

**PL/SQL**

**MANDATORY**

1) CONTROL STRUCTURES

i) Loan Interest rates

BEGIN

FOR cust IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

)

LOOP

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

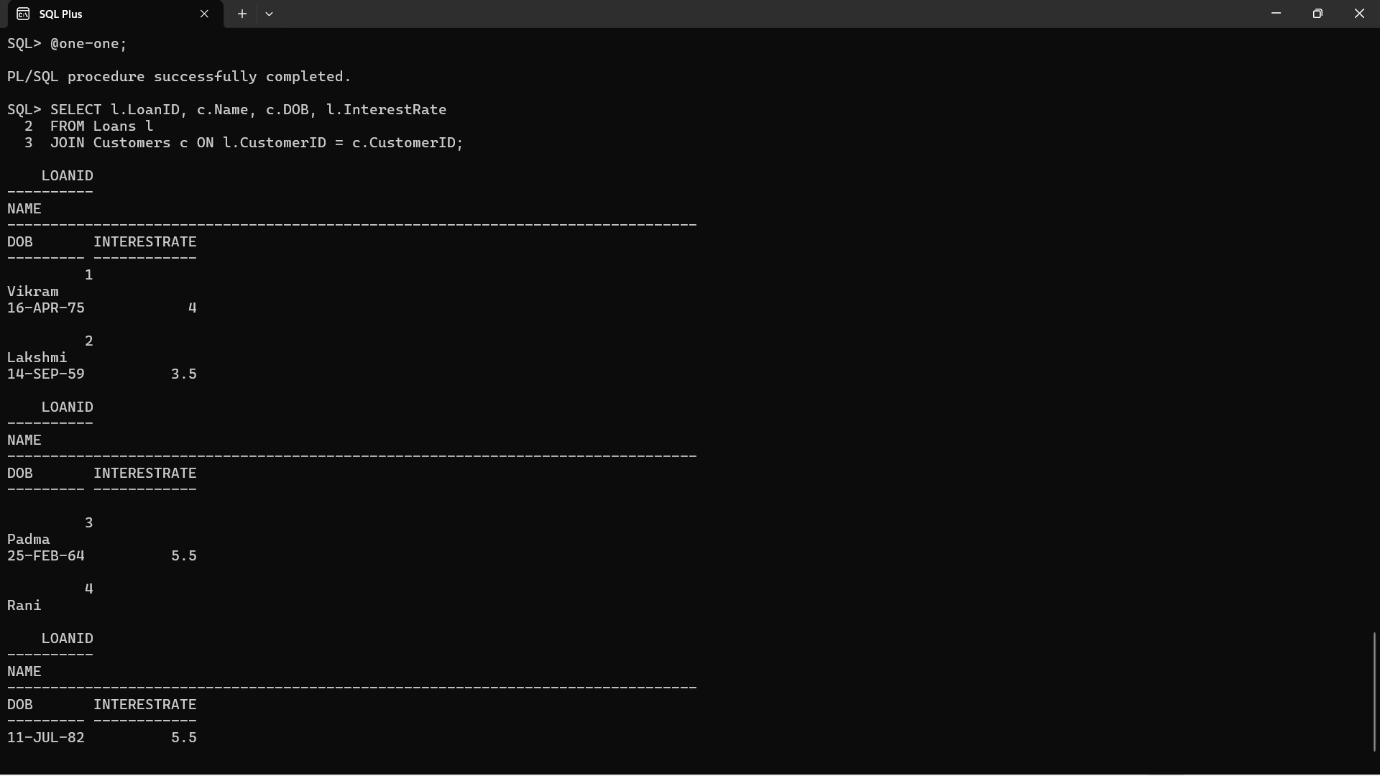
END IF;

END LOOP;

COMMIT;

END;

/



ii) VIP status

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

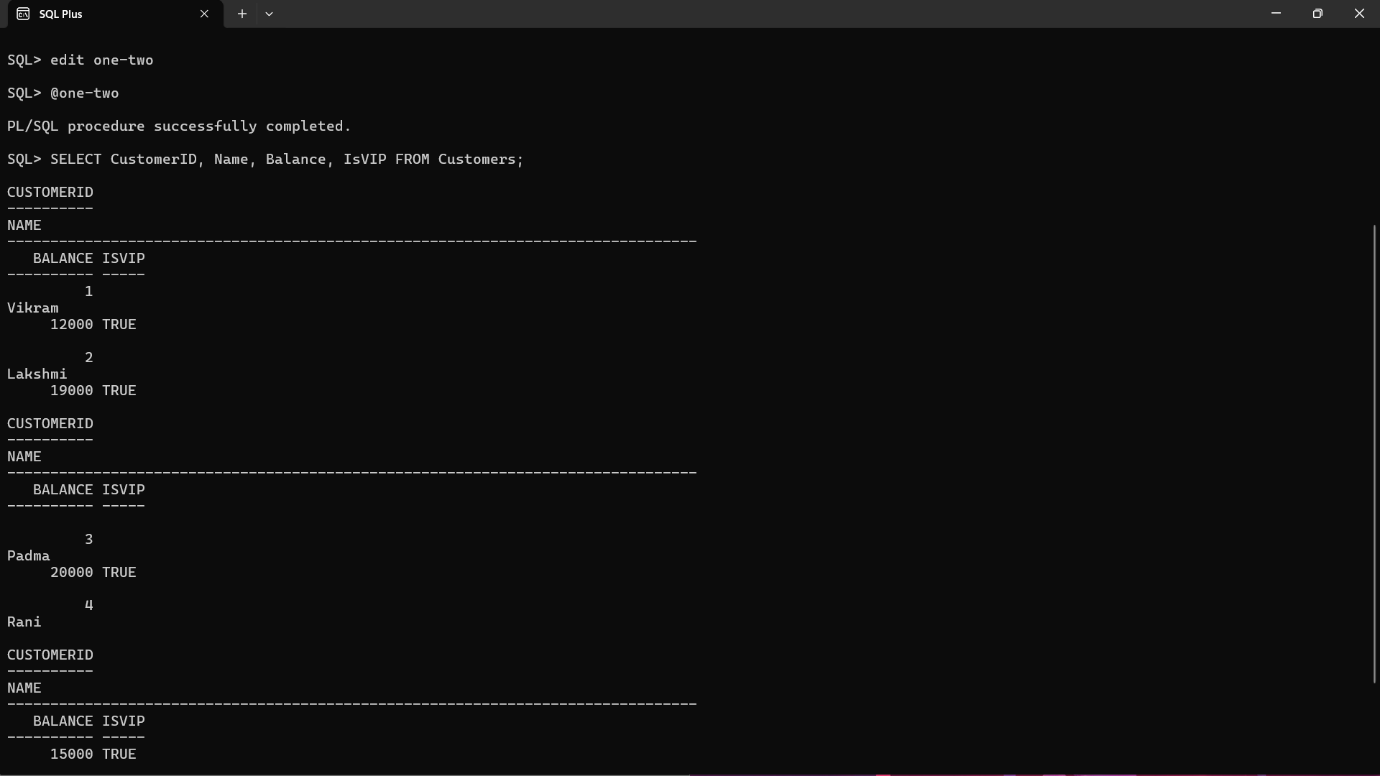
END IF;

END LOOP;

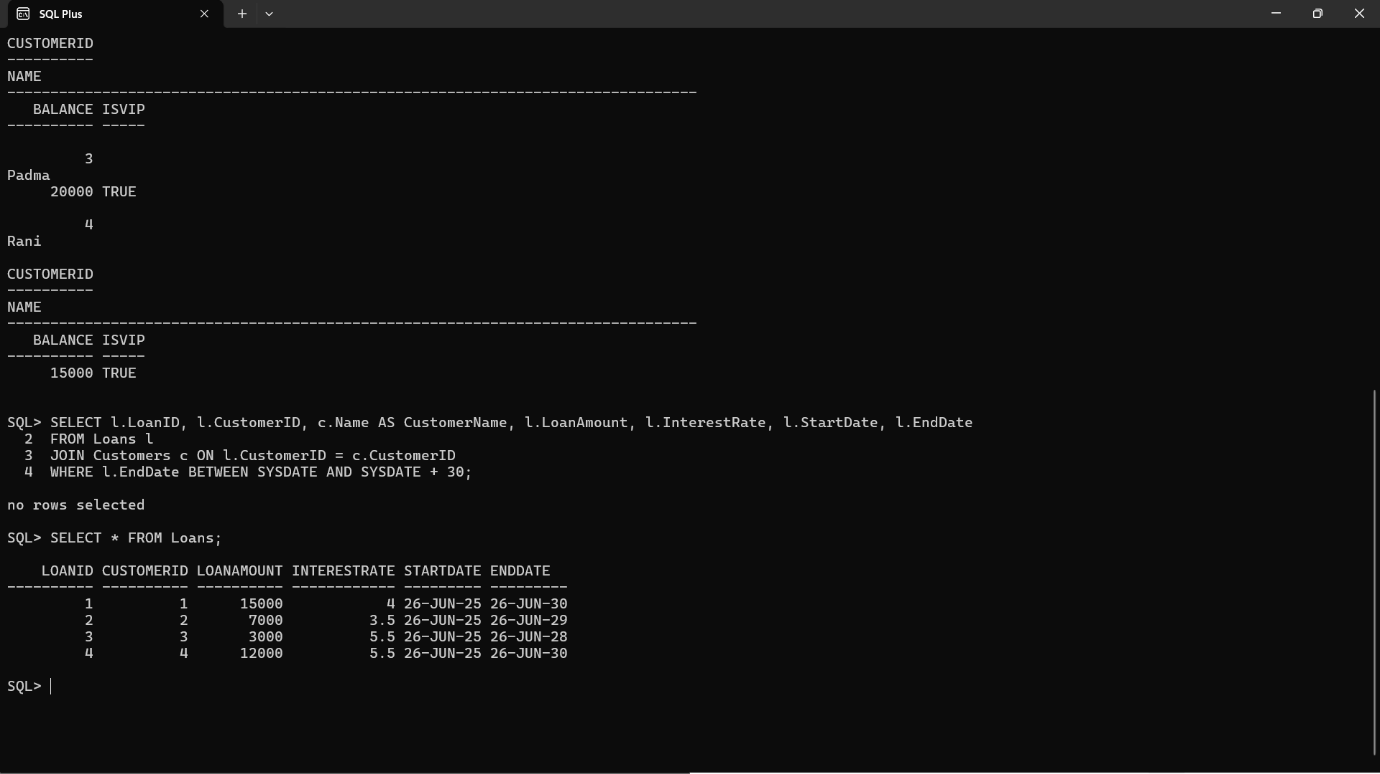
COMMIT;

END;

/



iii) Remainder Message



BEGIN

FOR rec IN (

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

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: Loan ID ' || rec.LoanID ||

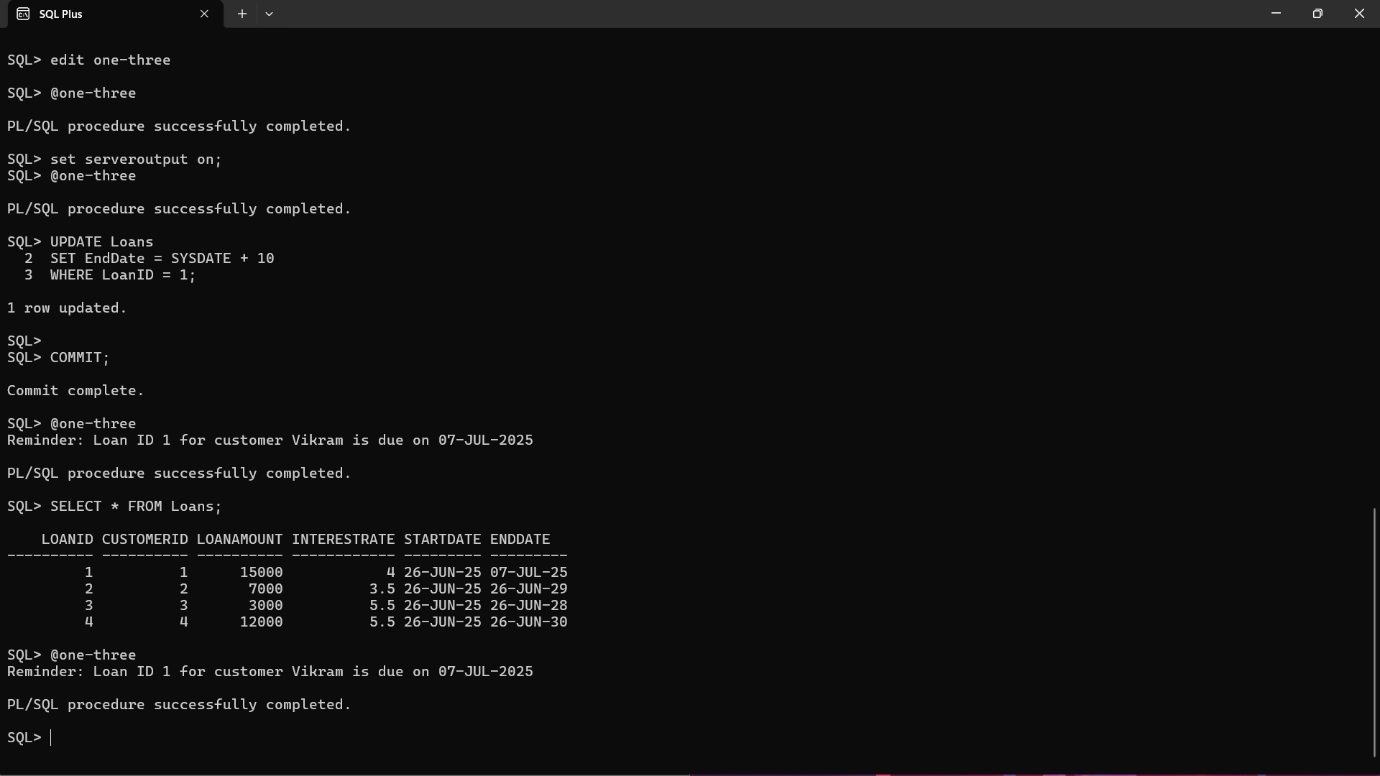
' for customer ' || rec.Name ||

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

END LOOP;

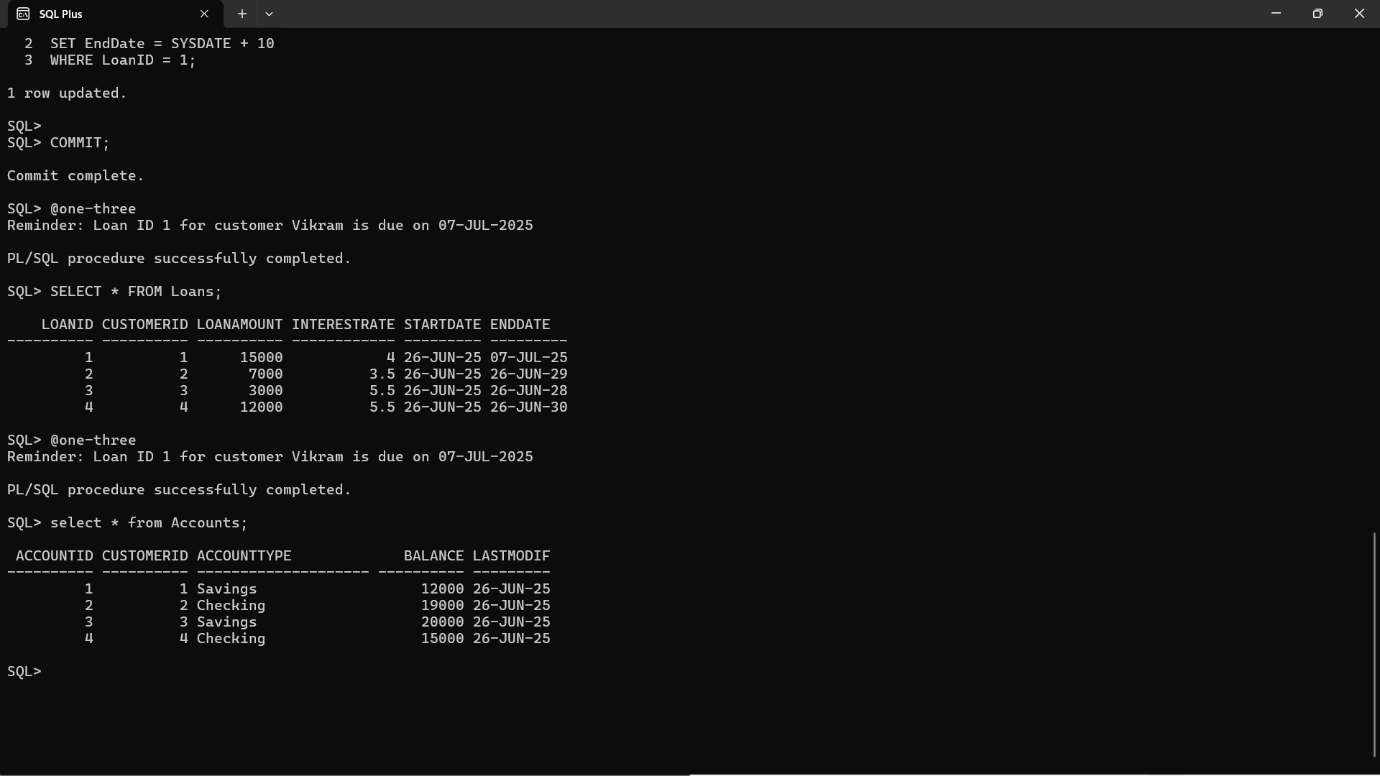
END;

/



3) Stored Procedures

i) ProcessMonthlyInterest



CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

FOR UPDATE

) LOOP

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest of 1% applied to all savings accounts.');

EXCEPTION

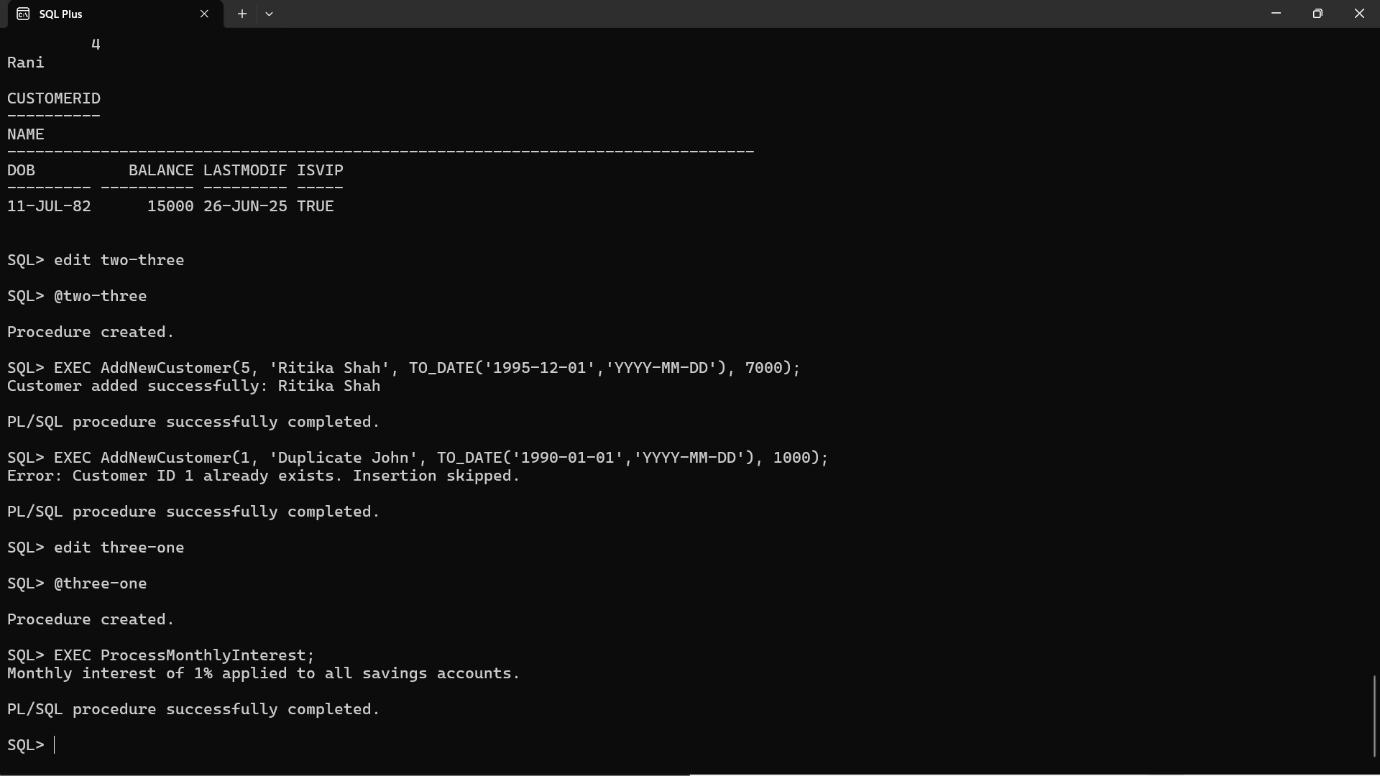
WHEN OTHERS THEN

ROLLBACK;

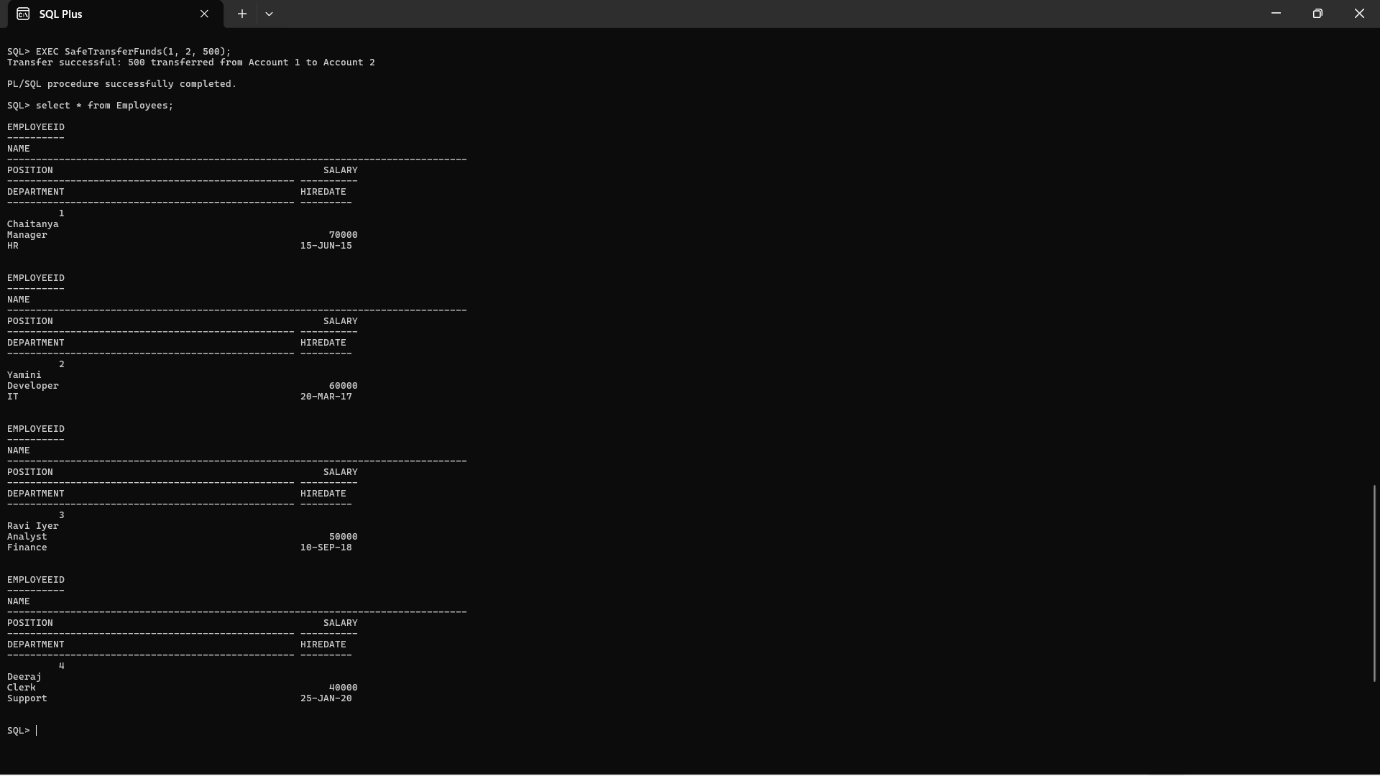
DBMS\_OUTPUT.PUT\_LINE('Error applying interest: ' || SQLERRM);

END;

/



ii) UpdateEmployeeBonus



CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

v\_Count NUMBER := 0;

BEGIN

-- Step 1: Update salaries

UPDATE Employees

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

WHERE Department = p\_Department;

-- Step 2: Check how many rows were updated

v\_Count := SQL%ROWCOUNT;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_BonusPercent || '% applied to ' || v\_Count || ' employee(s) in department: ' || p\_Department);

EXCEPTION

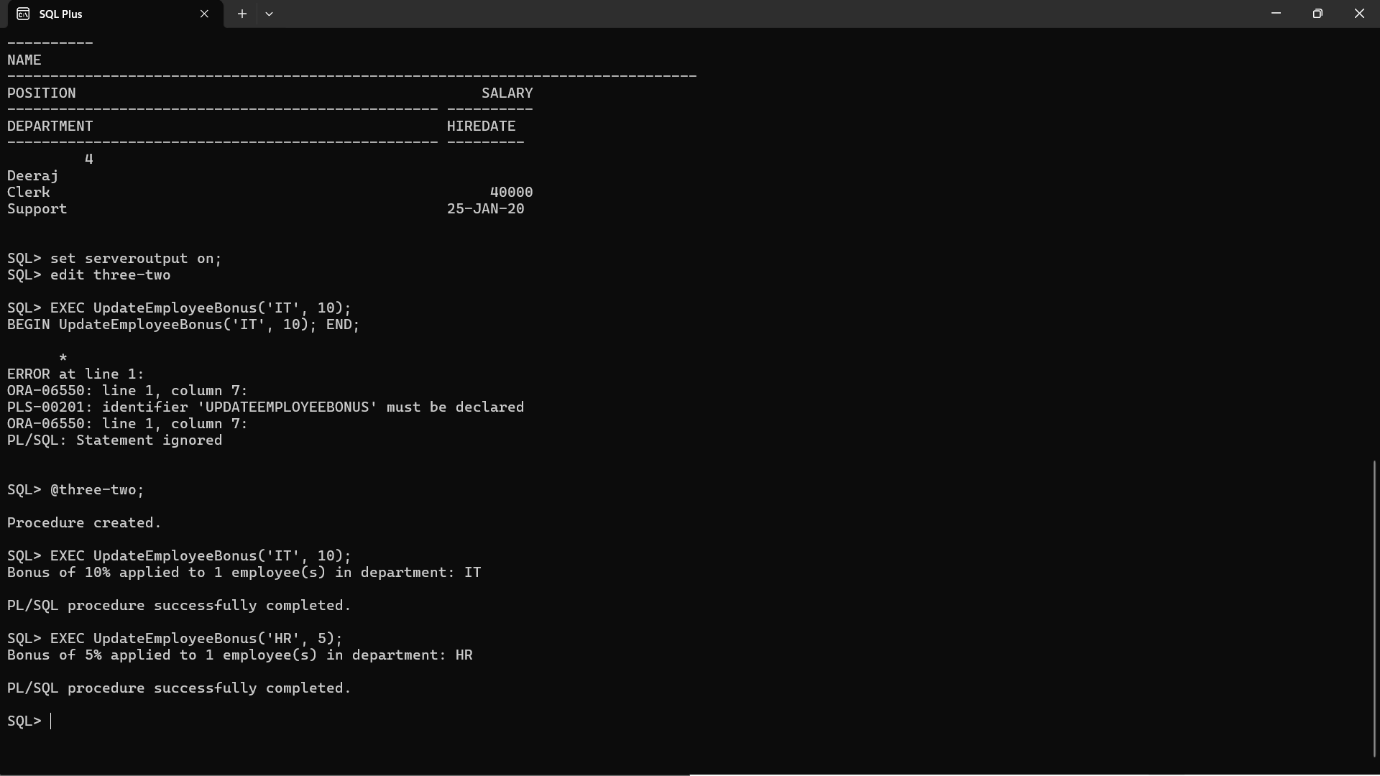
WHEN OTHERS THEN

ROLLBACK;

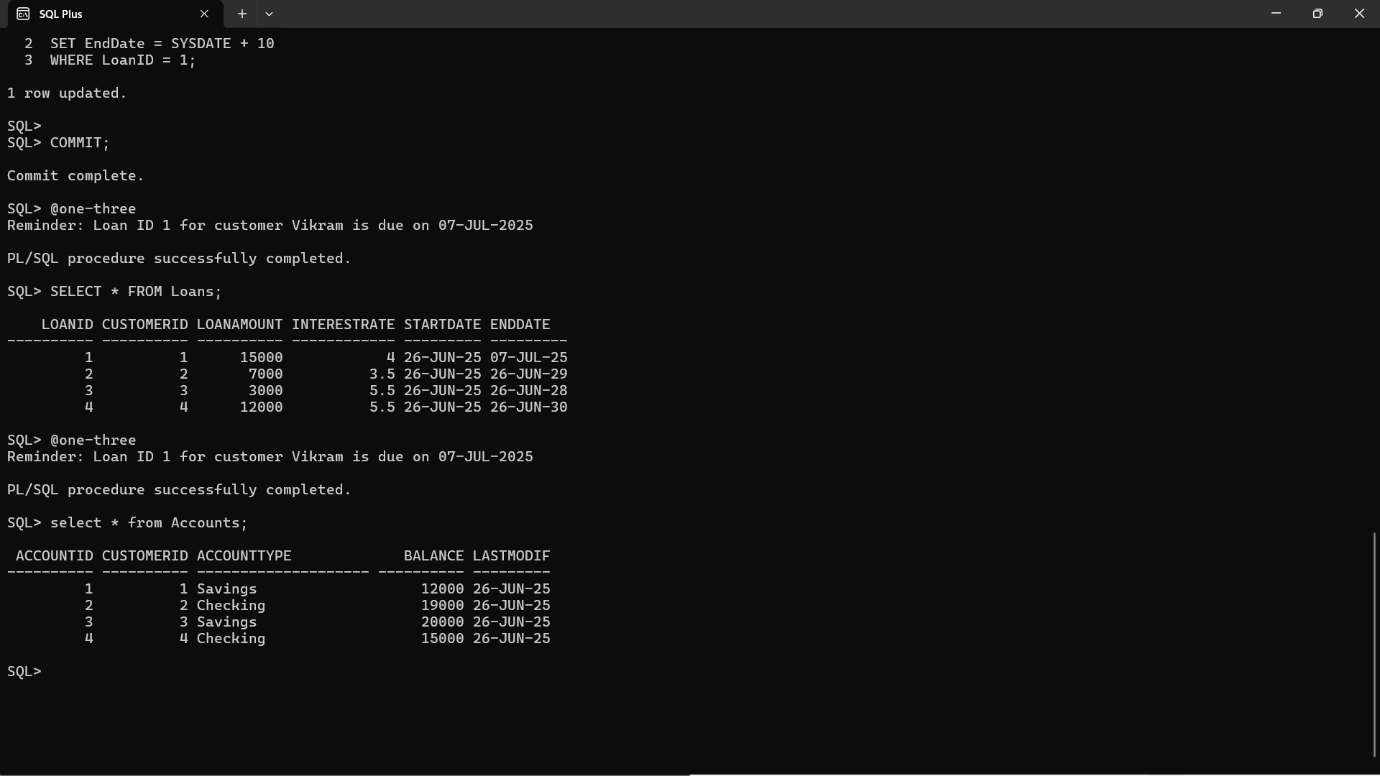
DBMS\_OUTPUT.PUT\_LINE('Error updating bonus: ' || SQLERRM);

END;

/



iii) TransferFunds



CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_SourceBalance NUMBER;

BEGIN

-- Step 1: Get source balance

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

-- Step 2: Check for sufficient balance

IF v\_SourceBalance < p\_Amount THEN

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

END IF;

-- Step 3: Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_FromAccountID;

-- Step 4: Add to destination

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_ToAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Successfully transferred ' || p\_Amount ||

' from Account ' || p\_FromAccountID ||

' to Account ' || p\_ToAccountID);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: One of the accounts does not exist.');

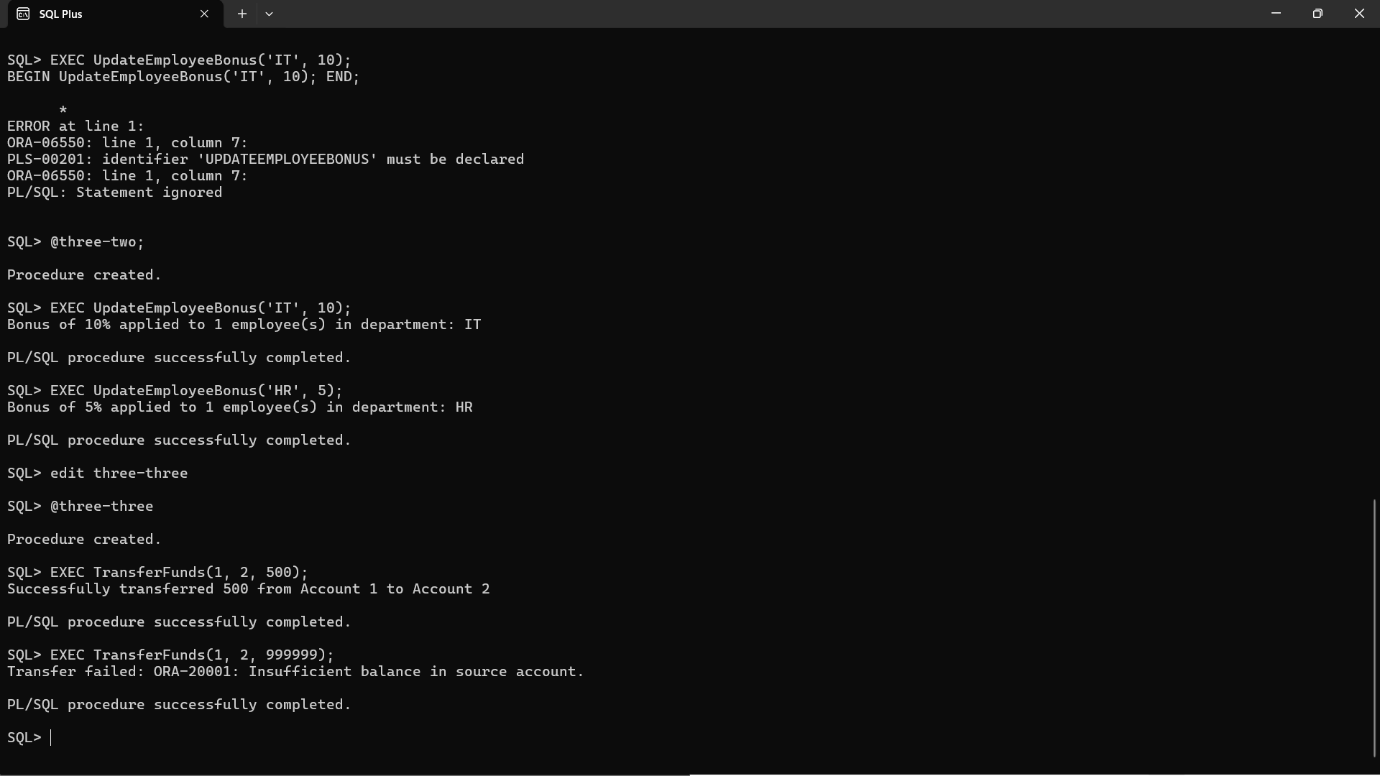
WHEN OTHERS THEN

ROLLBACK;

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

END;

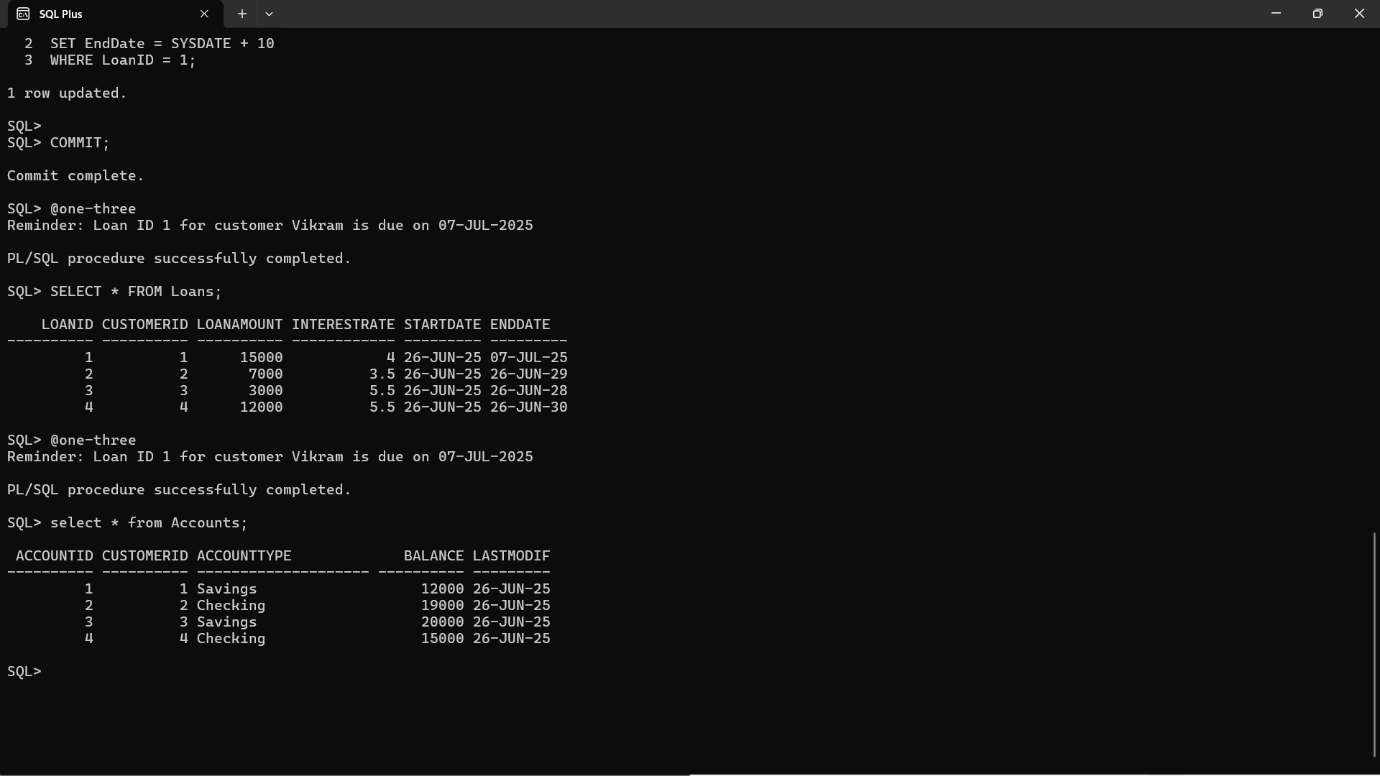
/



**PRACTICE EXERCISES**

2) ERROR HANDLING

i) SafeTransferFunds



CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_SourceBalance NUMBER;

BEGIN

-- Step 1: Get source account balance

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

-- Step 2: Check for sufficient funds

IF v\_SourceBalance < p\_Amount THEN

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

END IF;

-- Step 3: Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Step 4: Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

-- Step 5: Commit if all went well

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || p\_Amount || ' transferred from Account ' ||

p\_FromAccountID || ' to Account ' || p\_ToAccountID);

EXCEPTION

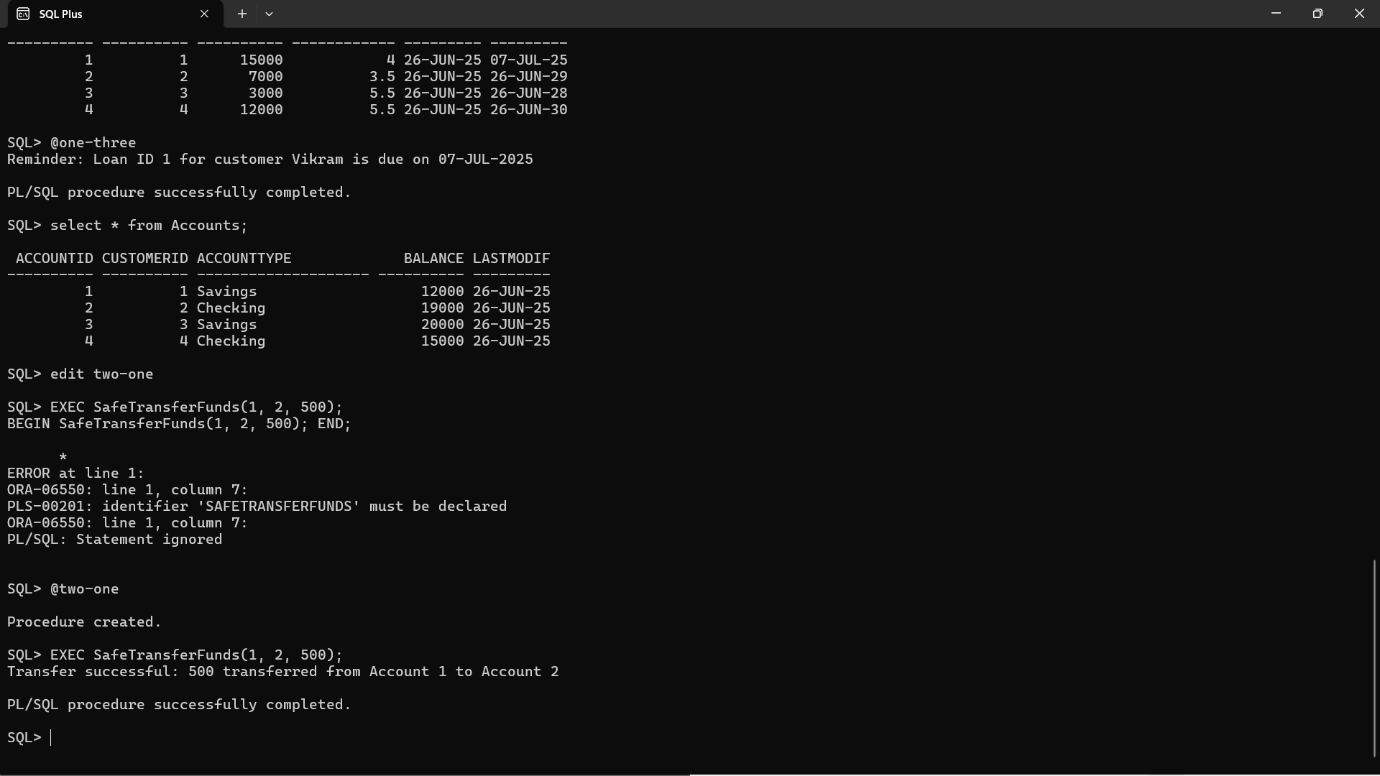
WHEN OTHERS THEN

ROLLBACK;

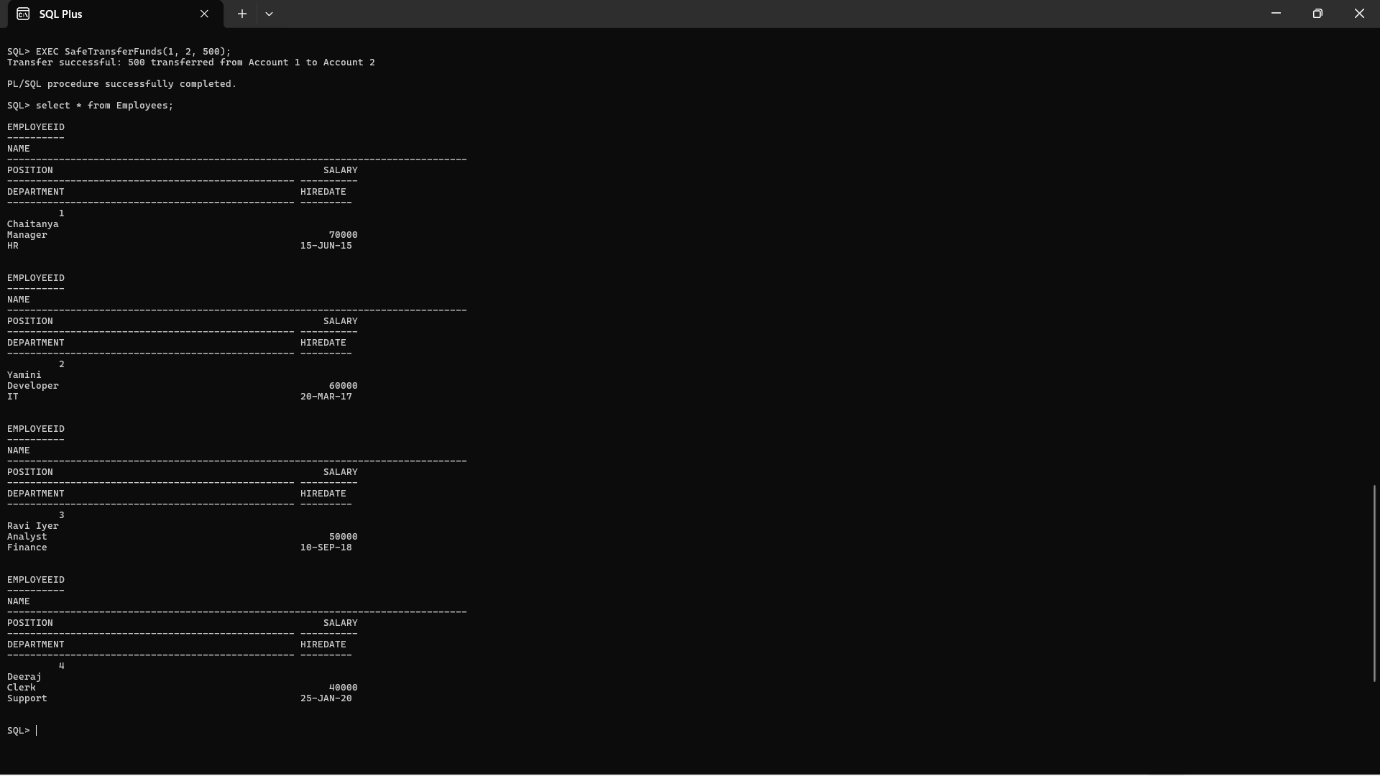
DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

/



ii) UpdateSalary



CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_PercentIncrease IN NUMBER

) IS

v\_CurrentSalary NUMBER;

BEGIN

-- Step 1: Get current salary

SELECT Salary INTO v\_CurrentSalary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

-- Step 2: Update salary by given percent

UPDATE Employees

SET Salary = Salary + (v\_CurrentSalary \* p\_PercentIncrease / 100)

WHERE EmployeeID = p\_EmployeeID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID ' || p\_EmployeeID);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

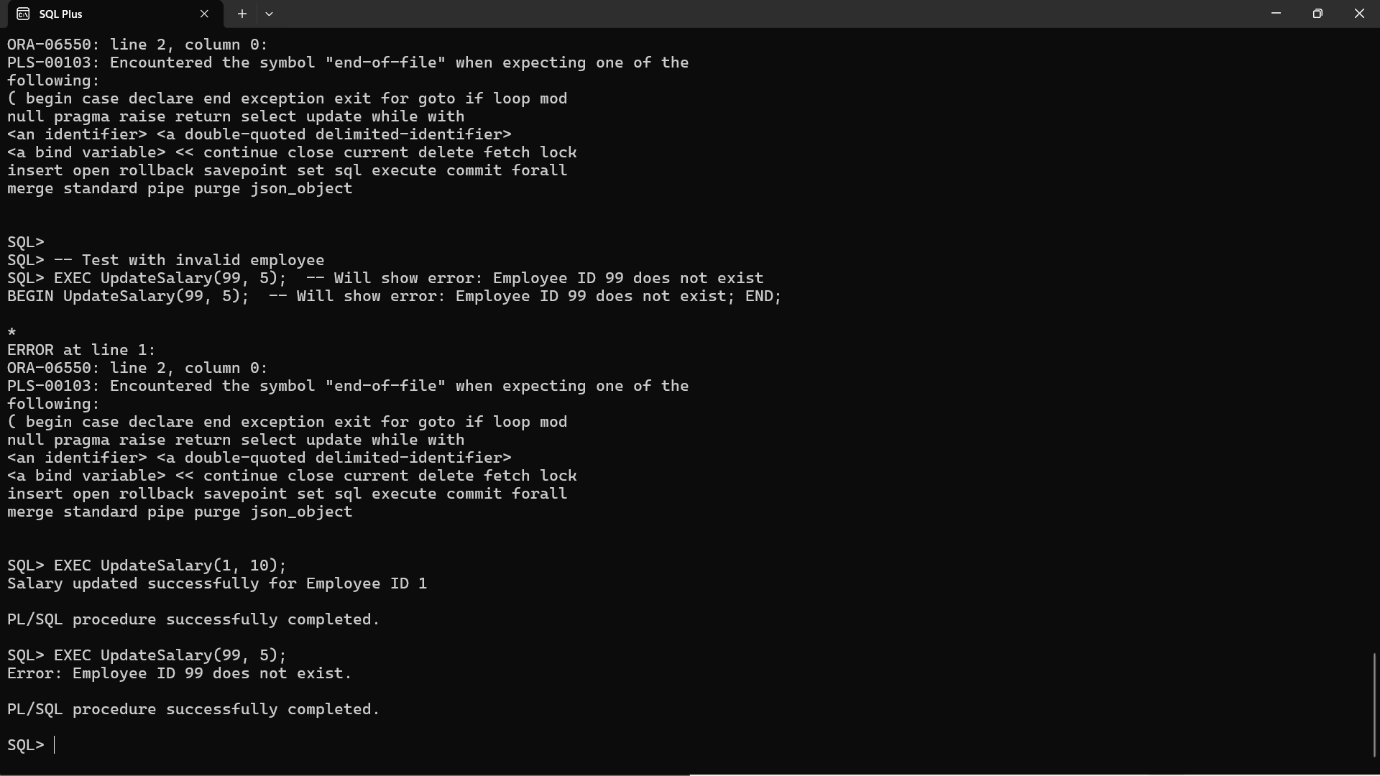
DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' does not exist.');

WHEN OTHERS THEN

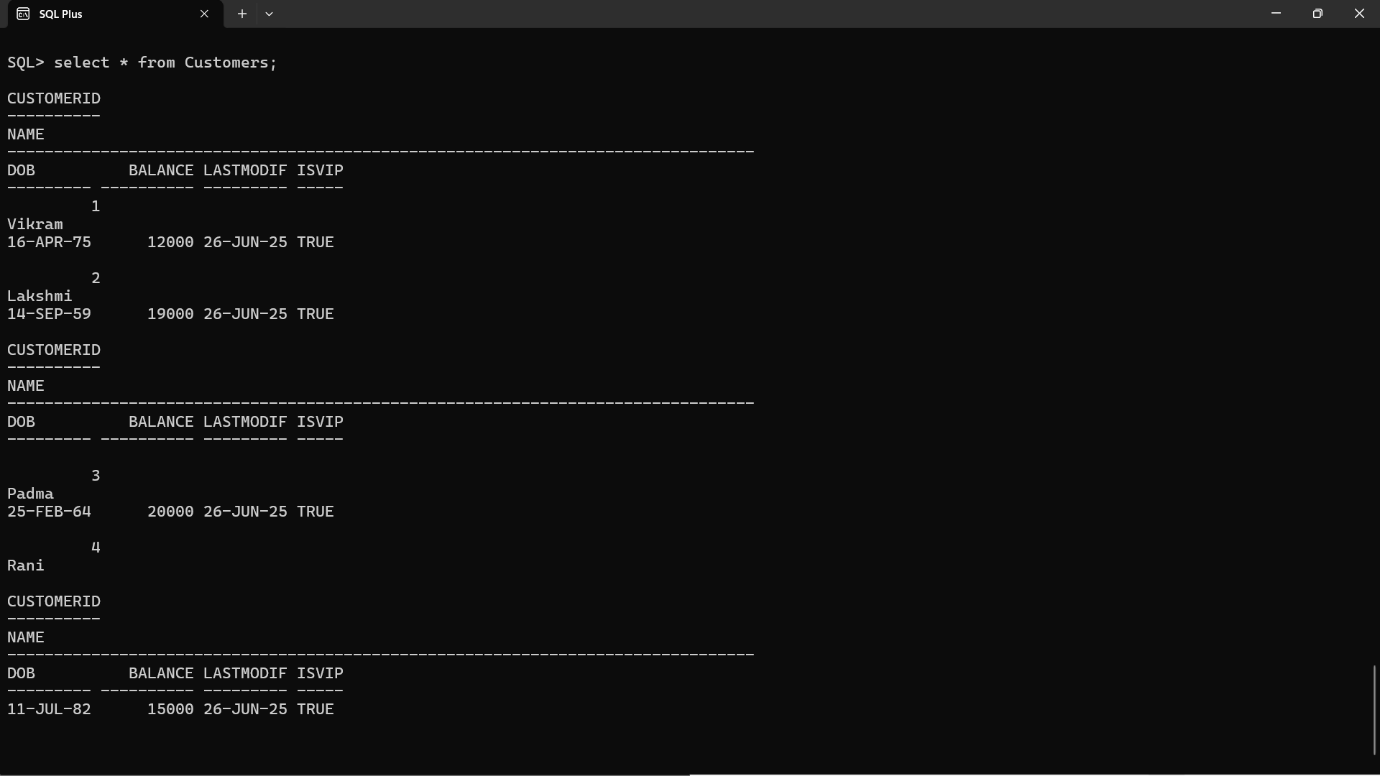
DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

/



iii) AddNewCustomer



CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID 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\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully: ' || p\_Name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

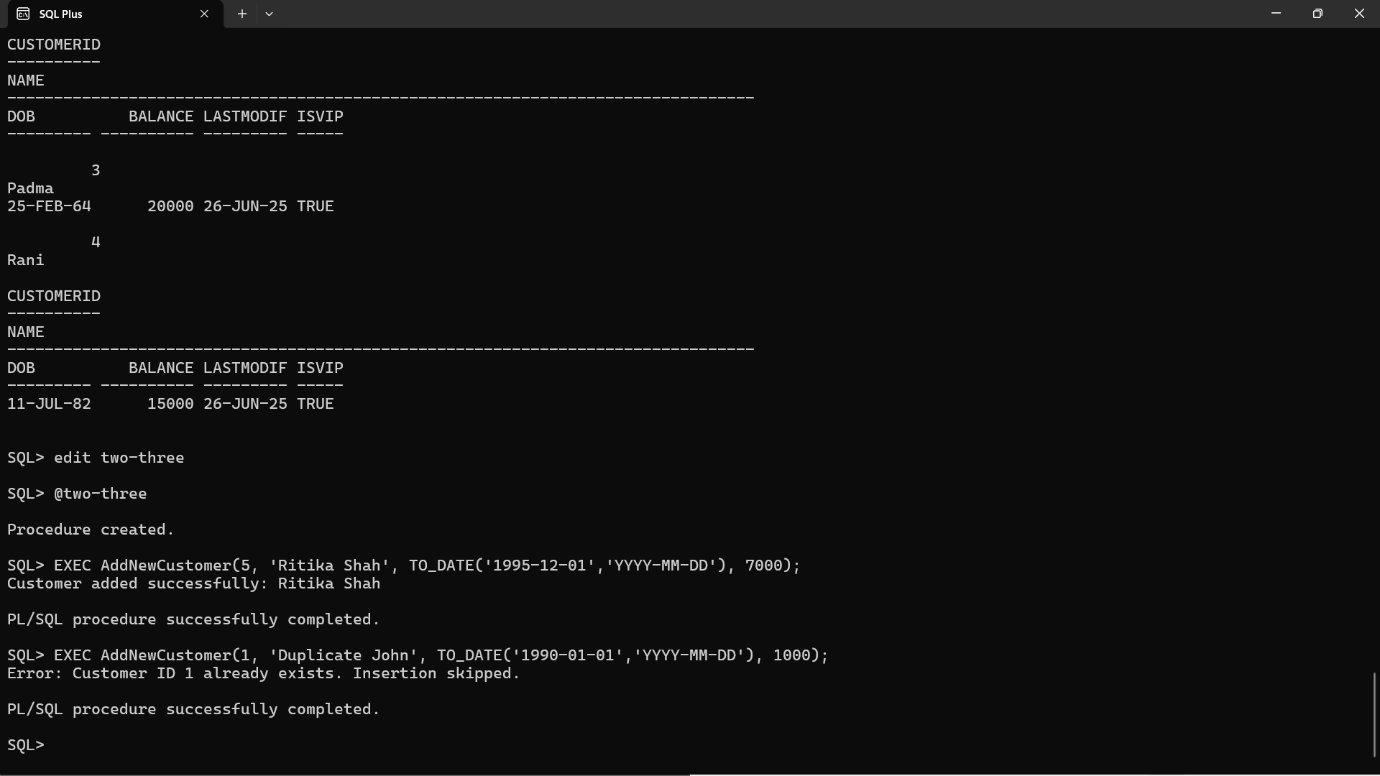
DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_CustomerID || ' already exists. Insertion skipped.');

WHEN OTHERS THEN

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

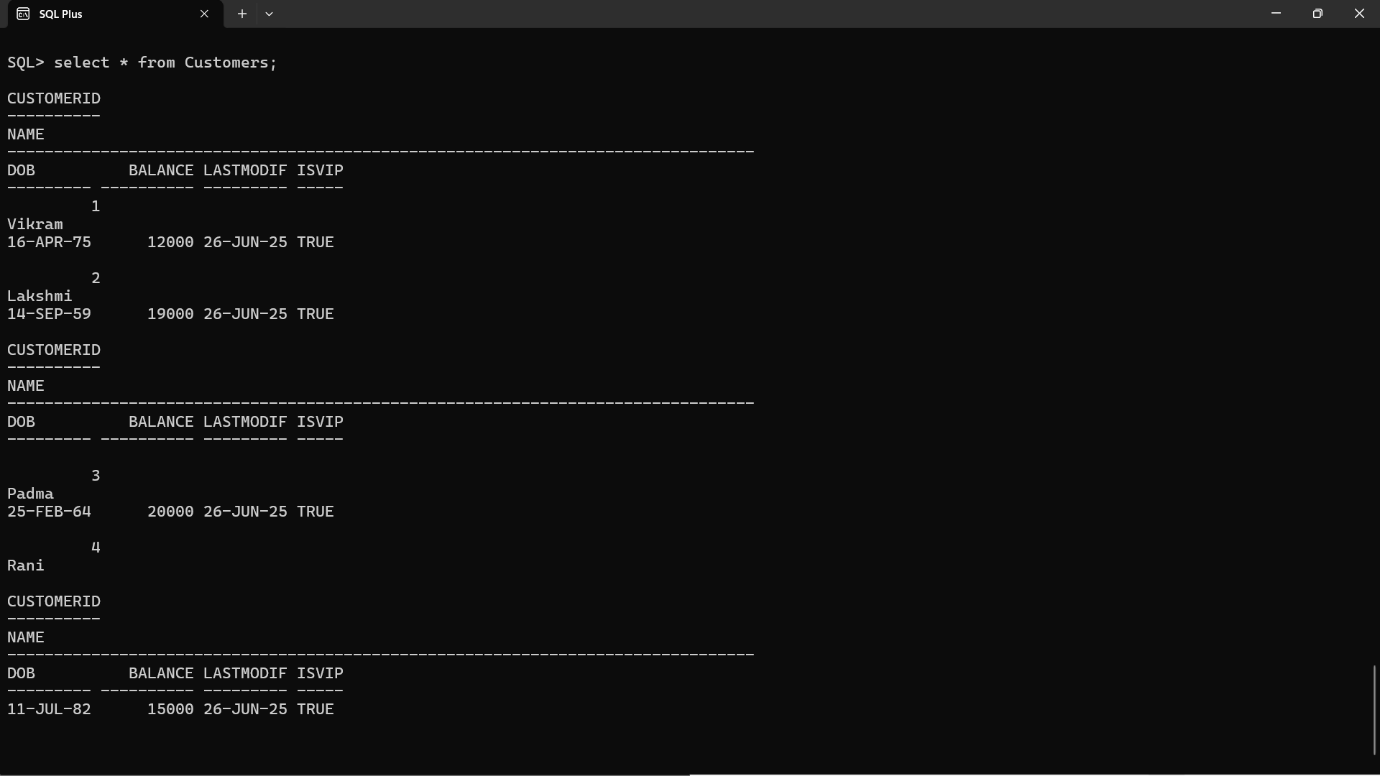
END;

/



4) FUNCTIONS

i) Calculate Age

  
CREATE OR REPLACE FUNCTION CalculateAge (

p\_DOB IN DATE

) RETURN NUMBER IS

v\_Age NUMBER;

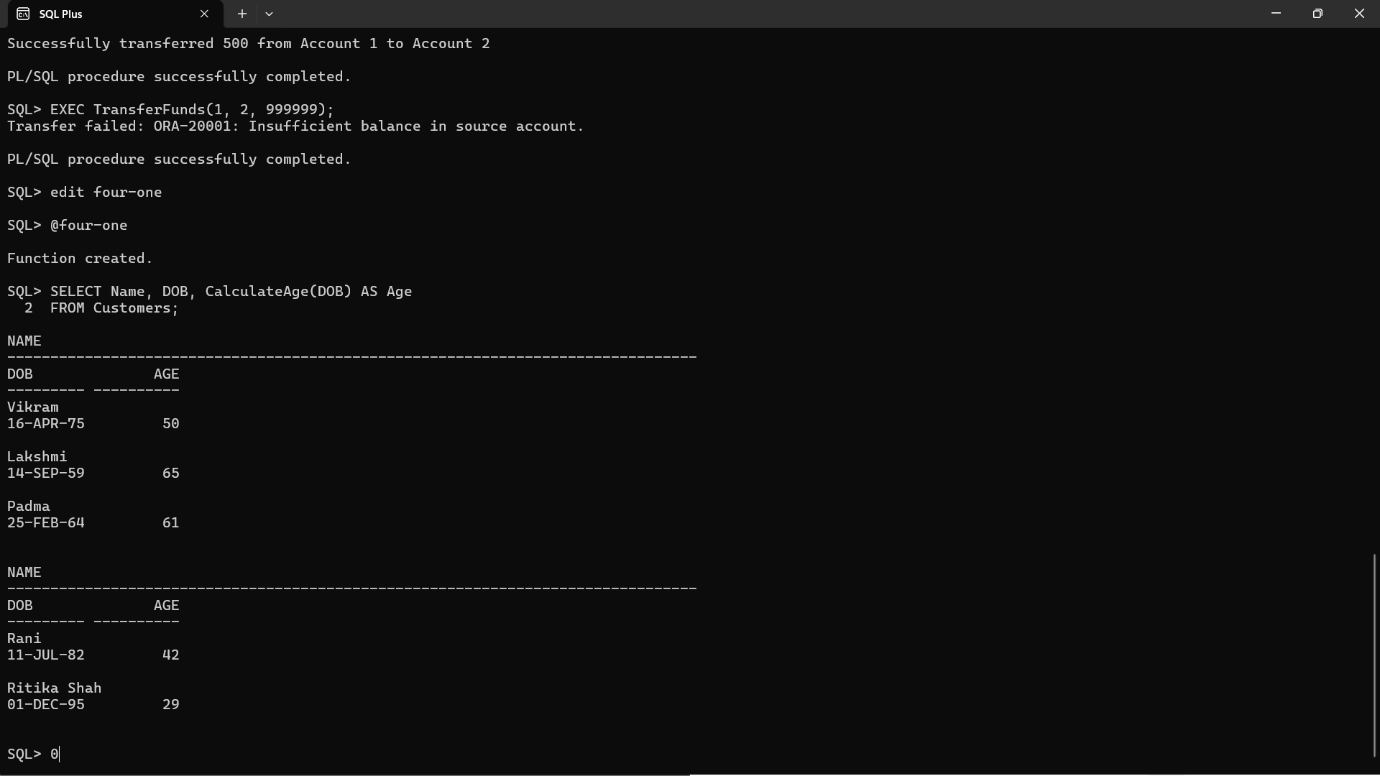
BEGIN

v\_Age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

/



ii) Calculate Monthly Installment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount IN NUMBER,

p\_InterestRate IN NUMBER,

p\_DurationYears IN NUMBER

) RETURN NUMBER IS

v\_MonthlyRate NUMBER;

v\_TotalMonths NUMBER;

v\_EMI NUMBER;

BEGIN

v\_MonthlyRate := p\_InterestRate / 12 / 100;

v\_TotalMonths := p\_DurationYears \* 12;

v\_EMI := (p\_LoanAmount \* v\_MonthlyRate \* POWER(1 + v\_MonthlyRate, v\_TotalMonths)) /

(POWER(1 + v\_MonthlyRate, v\_TotalMonths) - 1);

RETURN ROUND(v\_EMI, 2);

EXCEPTION

WHEN ZERO\_DIVIDE THEN

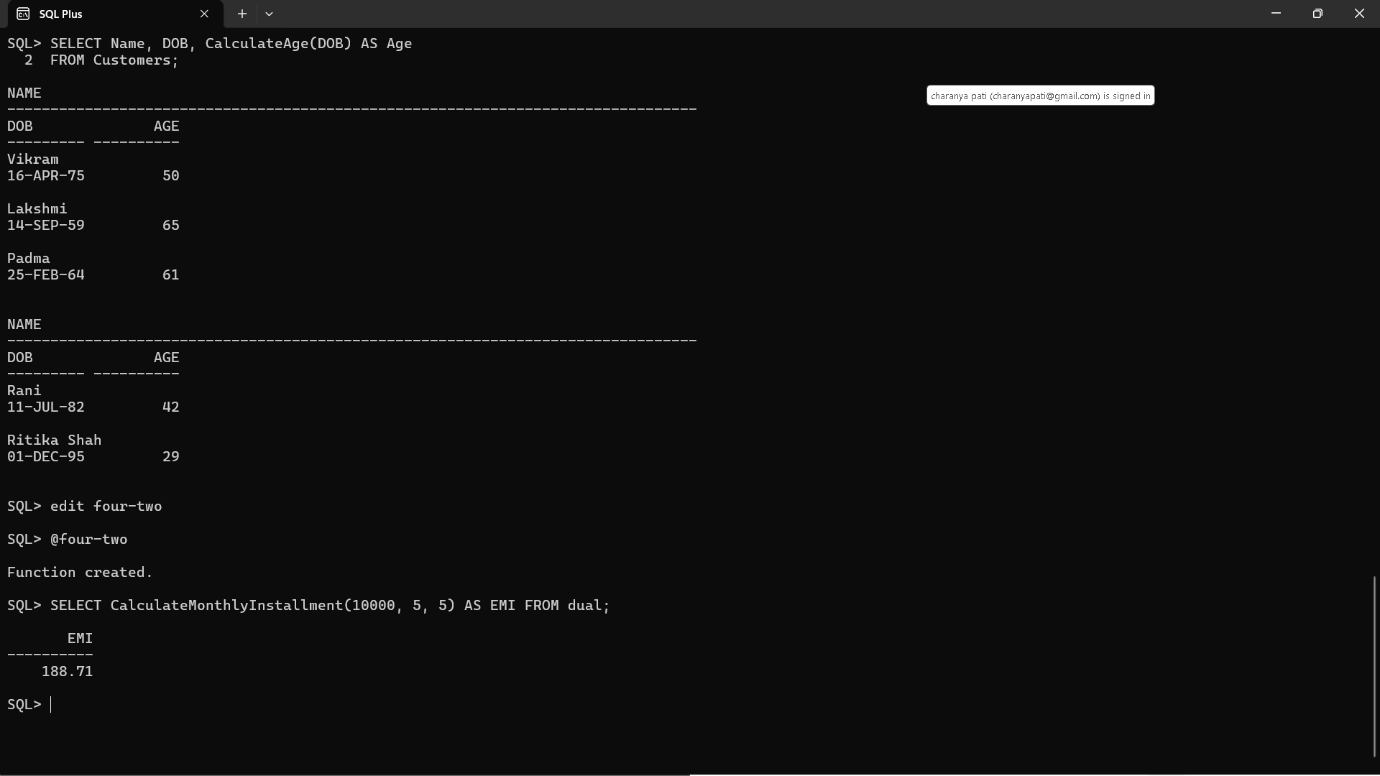
RETURN NULL;

WHEN OTHERS THEN

RETURN NULL;

END;

/



iii) HasSufficientBalance

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

) RETURN BOOLEAN IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

RETURN v\_Balance >= p\_Amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID ' || p\_AccountID || ' not found.');

RETURN FALSE;

WHEN OTHERS THEN

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

RETURN FALSE;

END;

/

DECLARE

v\_Result BOOLEAN;

BEGIN

v\_Result := HasSufficientBalance(1, 5000); -- Change account ID and amount as needed

IF v\_Result THEN

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

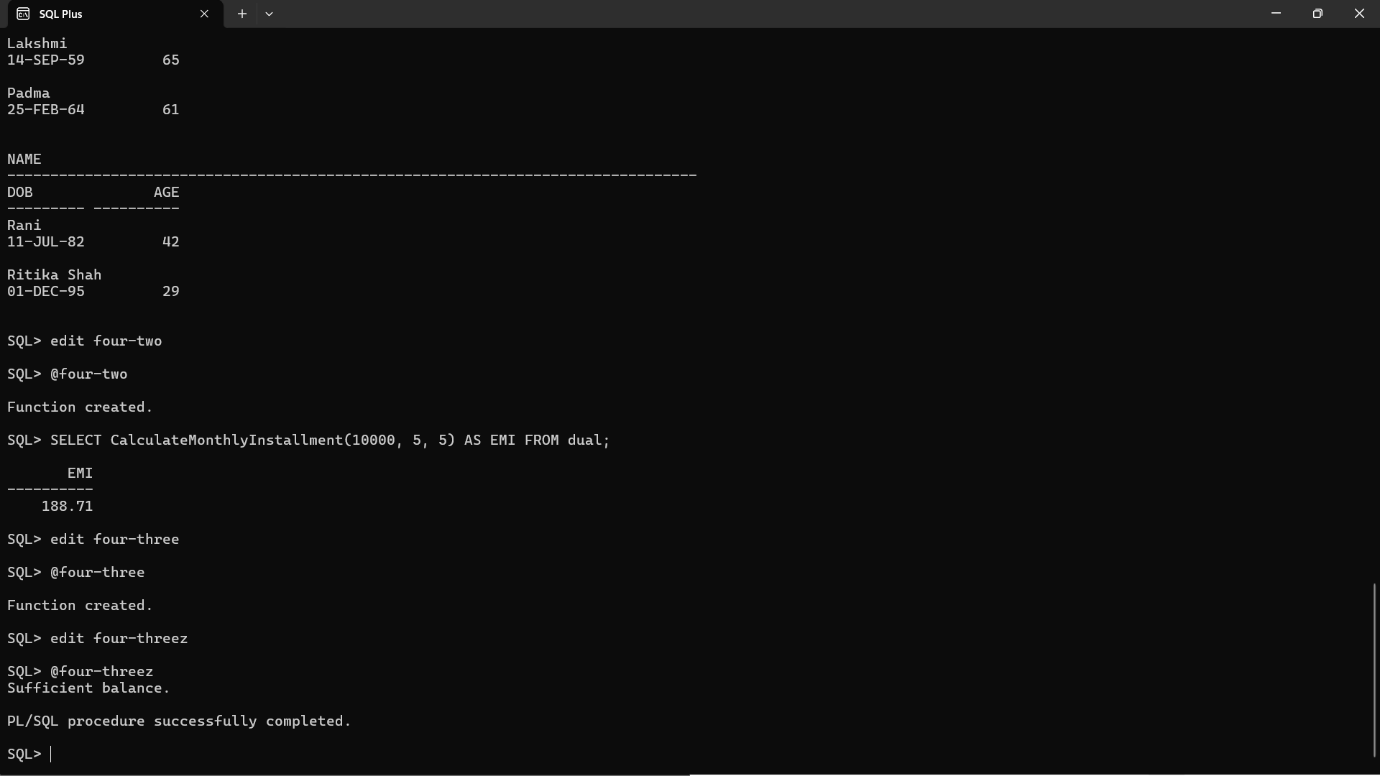
ELSE

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

END IF;

END;

/



5) TRIGGERS

i) UpdateCustomerLastModified

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

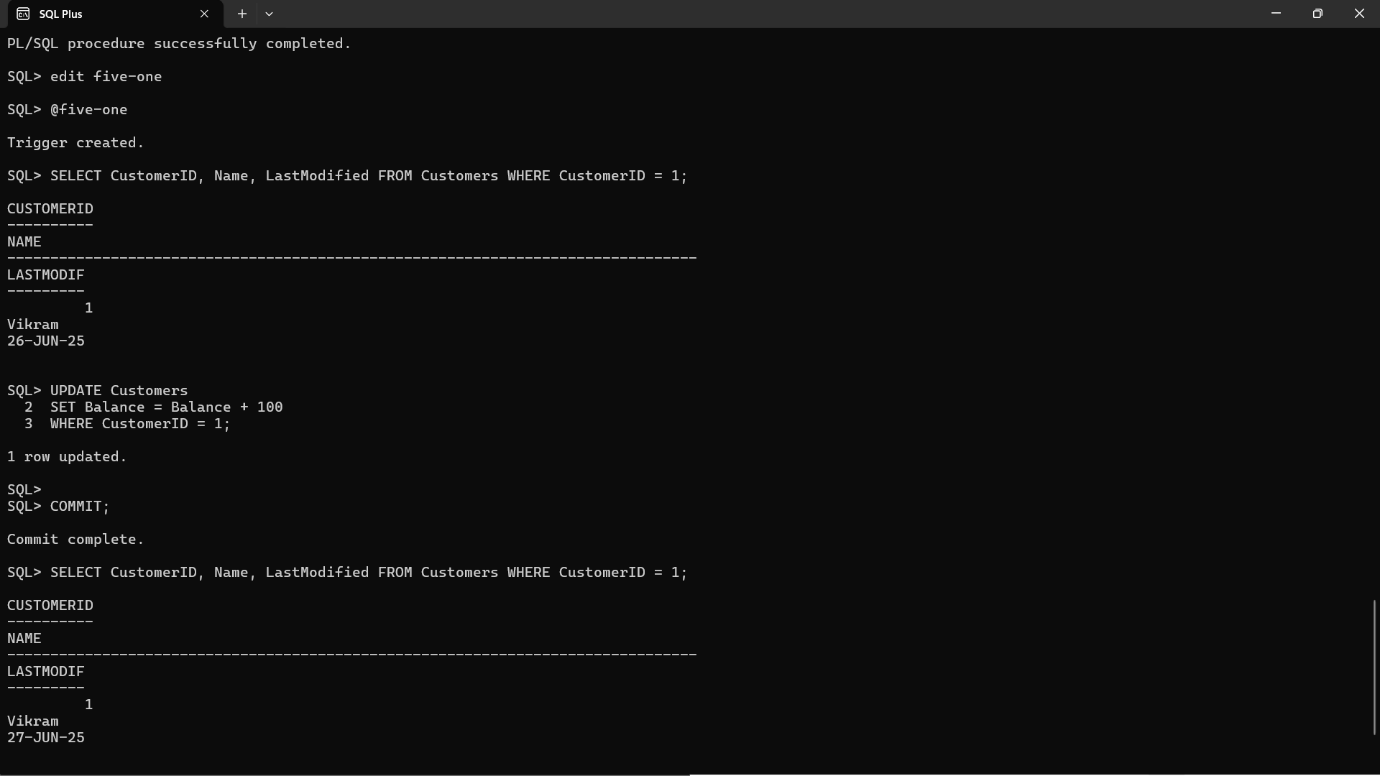
FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/



ii) LogTransaction

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (

TransactionID,

AccountID,

Action,

LogDate

) VALUES (

:NEW.TransactionID,

:NEW.AccountID,

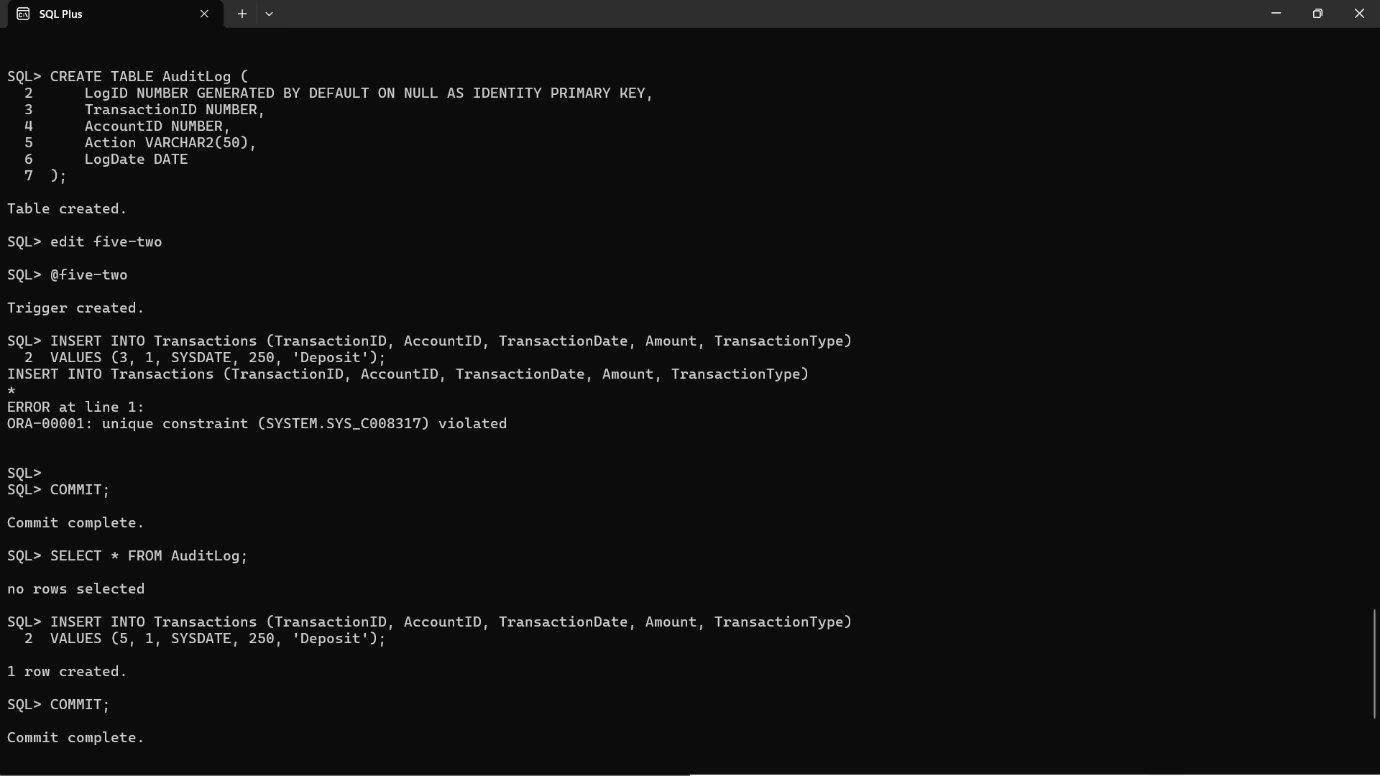
'INSERTED TRANSACTION',

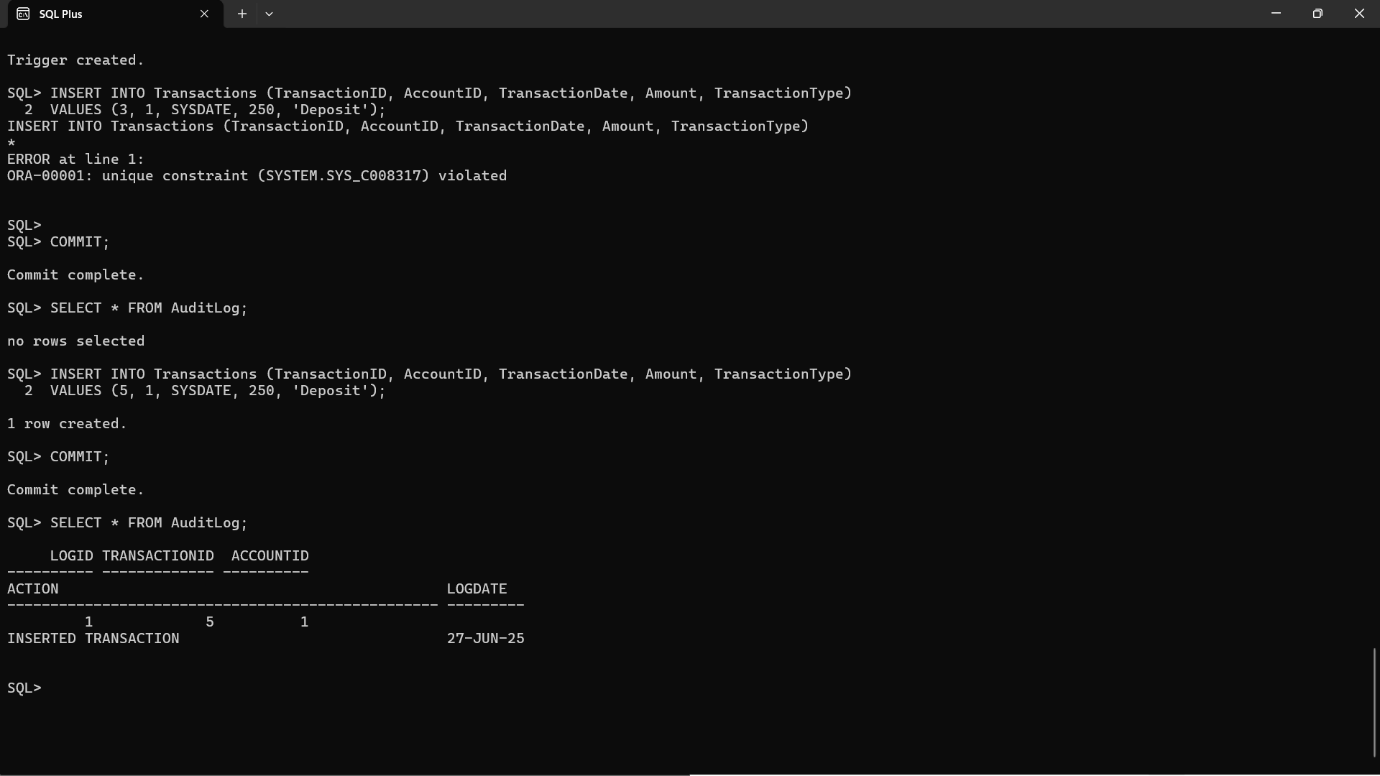
SYSDATE

);

END;

/





iii) CheckTransactionRules

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_Balance NUMBER;

BEGIN

-- Get the current balance of the account

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

-- Rule 1: Deposit must be positive

IF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Deposit amount must be greater than 0.');

END IF;

-- Rule 2: Withdrawal must not exceed balance

ELSIF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > v\_Balance THEN

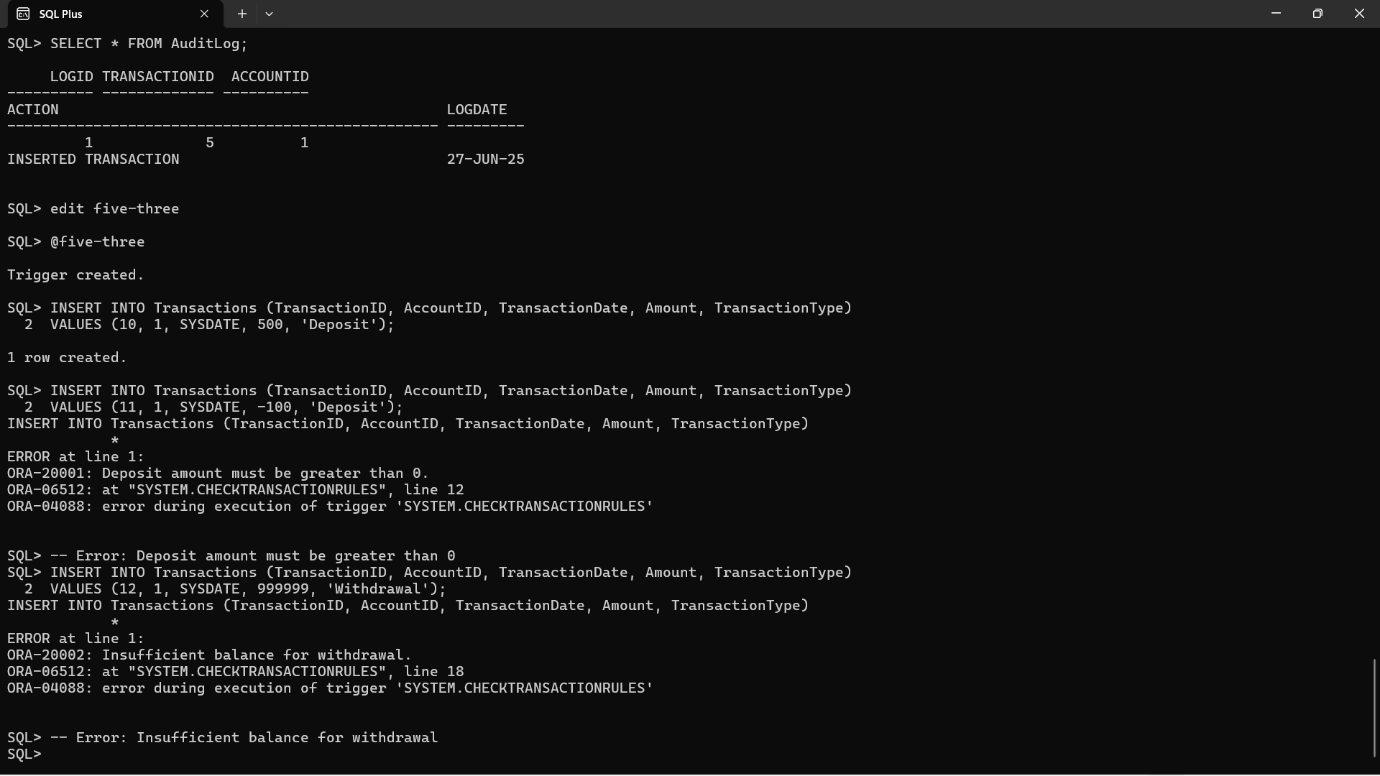
RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

END IF;

END;

/



6) CURSORS

i) GeneratemonthlyStatements

SET SERVEROUTPUT ON;

DECLARE

CURSOR txn\_cursor IS

SELECT

t.TransactionID,

t.AccountID,

c.CustomerID,

c.Name AS CustomerName,

t.TransactionDate,

t.Amount,

t.TransactionType

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

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

AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE)

ORDER BY c.CustomerID, t.TransactionDate;

v\_LastCustomerID NUMBER := NULL;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Monthly Transaction Statements ---');

FOR txn IN txn\_cursor LOOP

-- Header for each new customer

IF v\_LastCustomerID IS NULL OR v\_LastCustomerID != txn.CustomerID THEN

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || txn.CustomerName || ' (ID: ' || txn.CustomerID || ')');

DBMS\_OUTPUT.PUT\_LINE('Transactions:');

v\_LastCustomerID := txn.CustomerID;

END IF;

-- Transaction line

DBMS\_OUTPUT.PUT\_LINE(' [' || TO\_CHAR(txn.TransactionDate, 'DD-MON') || '] ' ||

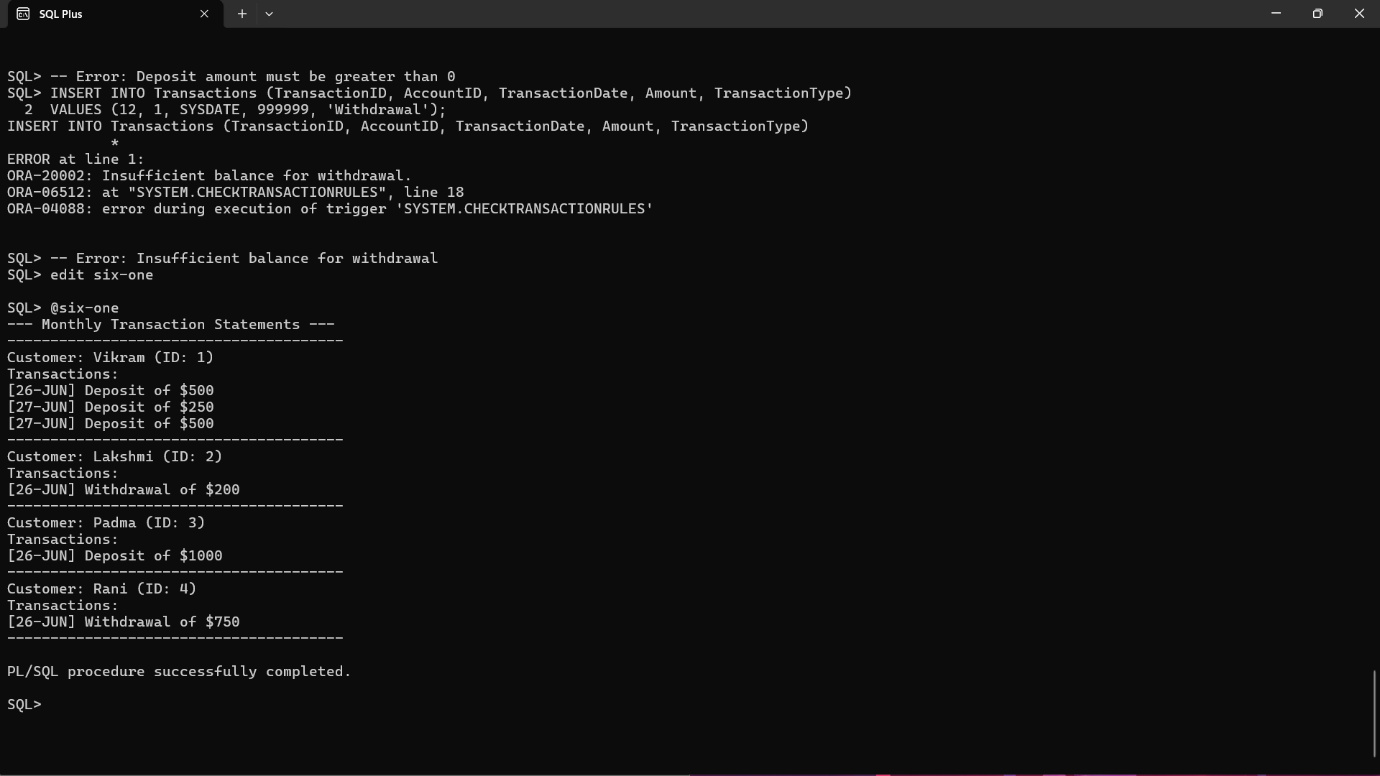
txn.TransactionType || ' of $' || txn.Amount);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------');

END;

/



ii) ApplyAnnualFee

SET SERVEROUTPUT ON;

DECLARE

-- Define the annual maintenance fee

v\_AnnualFee NUMBER := 100;

-- Cursor to loop through all accounts

CURSOR acc\_cursor IS

SELECT AccountID, Balance

FROM Accounts

FOR UPDATE;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Applying Annual Maintenance Fee ---');

-- Loop through each account

FOR acc IN acc\_cursor LOOP

-- Deduct fee and update LastModified

UPDATE Accounts

SET Balance = Balance - v\_AnnualFee,

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

-- Output confirmation

DBMS\_OUTPUT.PUT\_LINE('Annual fee of $' || v\_AnnualFee ||

' deducted from Account ID ' || acc.AccountID);

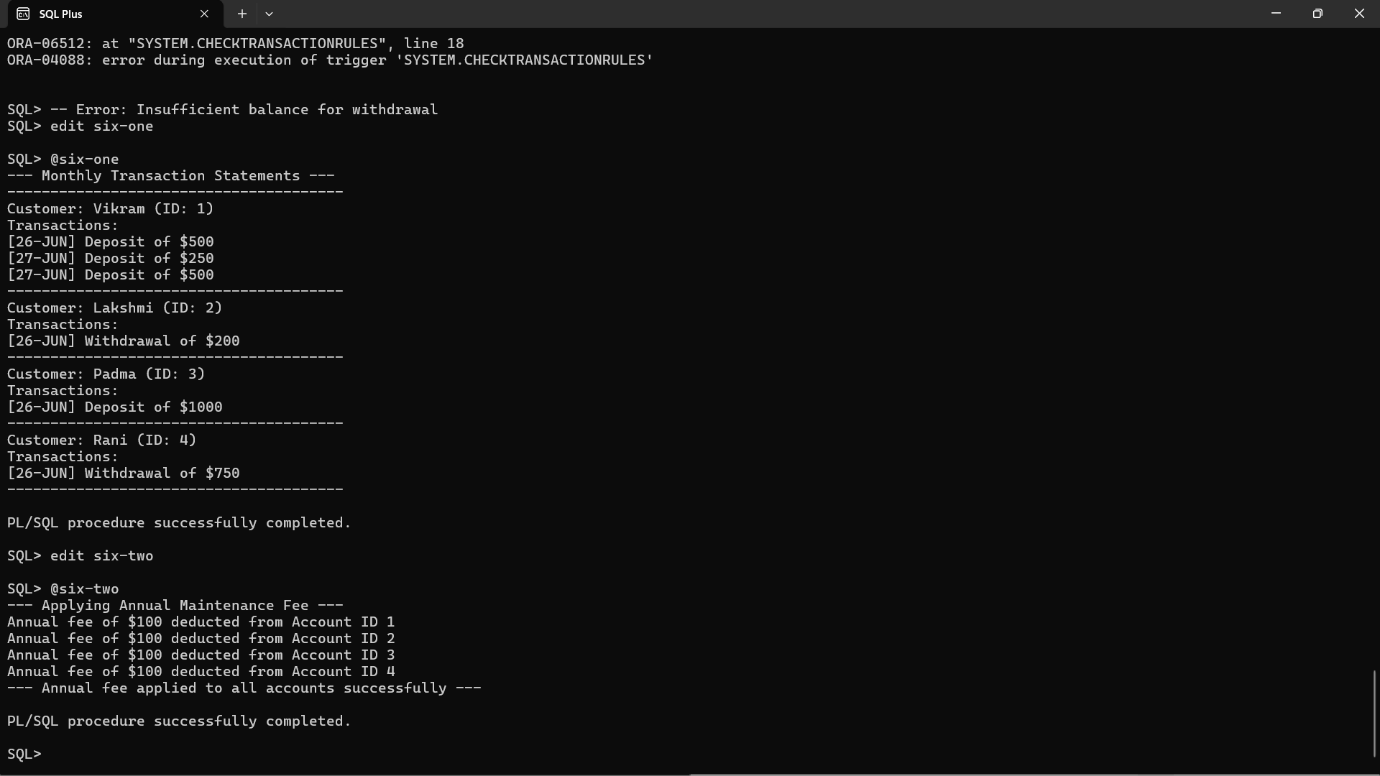
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Annual fee applied to all accounts successfully ---');

END;

/



iii) UpdateLoanInterestRates

SET SERVEROUTPUT ON;

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, LoanAmount, InterestRate

FROM Loans

FOR UPDATE;

v\_NewRate NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Updating Loan Interest Rates Based on Policy ---');

FOR loan\_rec IN loan\_cursor LOOP

-- Apply new policy rules

IF loan\_rec.LoanAmount > 10000 THEN

v\_NewRate := loan\_rec.InterestRate - 0.5;

ELSIF loan\_rec.LoanAmount BETWEEN 5000 AND 10000 THEN

v\_NewRate := loan\_rec.InterestRate - 0.25;

ELSE

v\_NewRate := loan\_rec.InterestRate;

END IF;

-- Update loan if rate changed

IF v\_NewRate != loan\_rec.InterestRate THEN

UPDATE Loans

SET InterestRate = v\_NewRate

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID ' || loan\_rec.LoanID ||

' updated from ' || loan\_rec.InterestRate || '% to ' || v\_NewRate || '%');

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Loan interest rate updates completed ---');

END;

/

