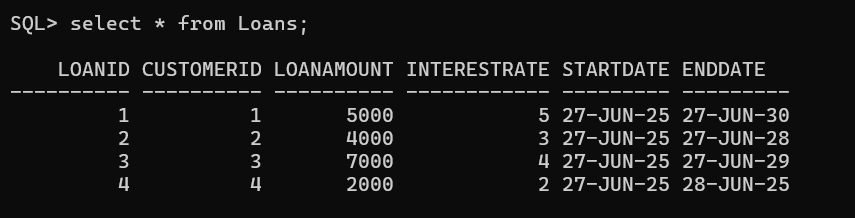
**WEEK - 2**

**PL/SQL**

**Mandatory**

**Exercise 1: Implementing the Singleton Pattern**

**Scenario - 1:**



BEGIN

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

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

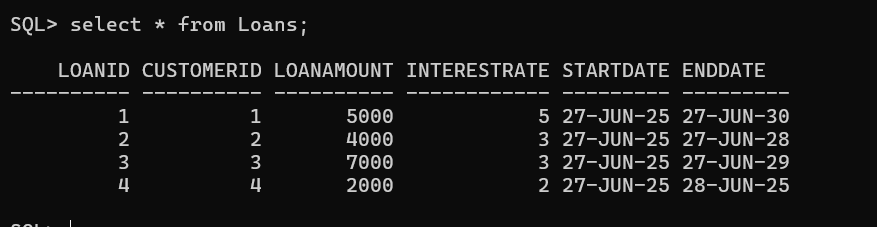
END IF;

END LOOP;

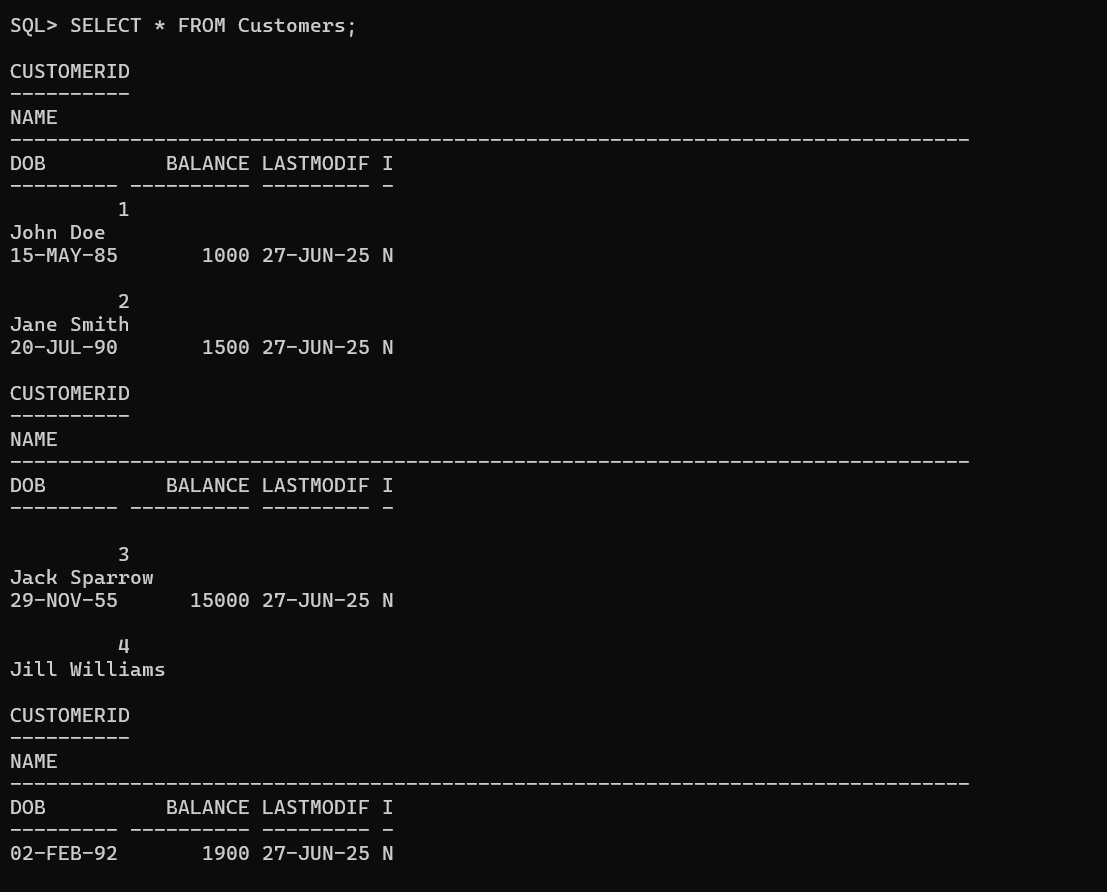
COMMIT;

END;

/



**Scenario - 2:**



ALTER TABLE Customers ADD IsVIP CHAR(1);

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'N'

WHERE CustomerID = cust.CustomerID;

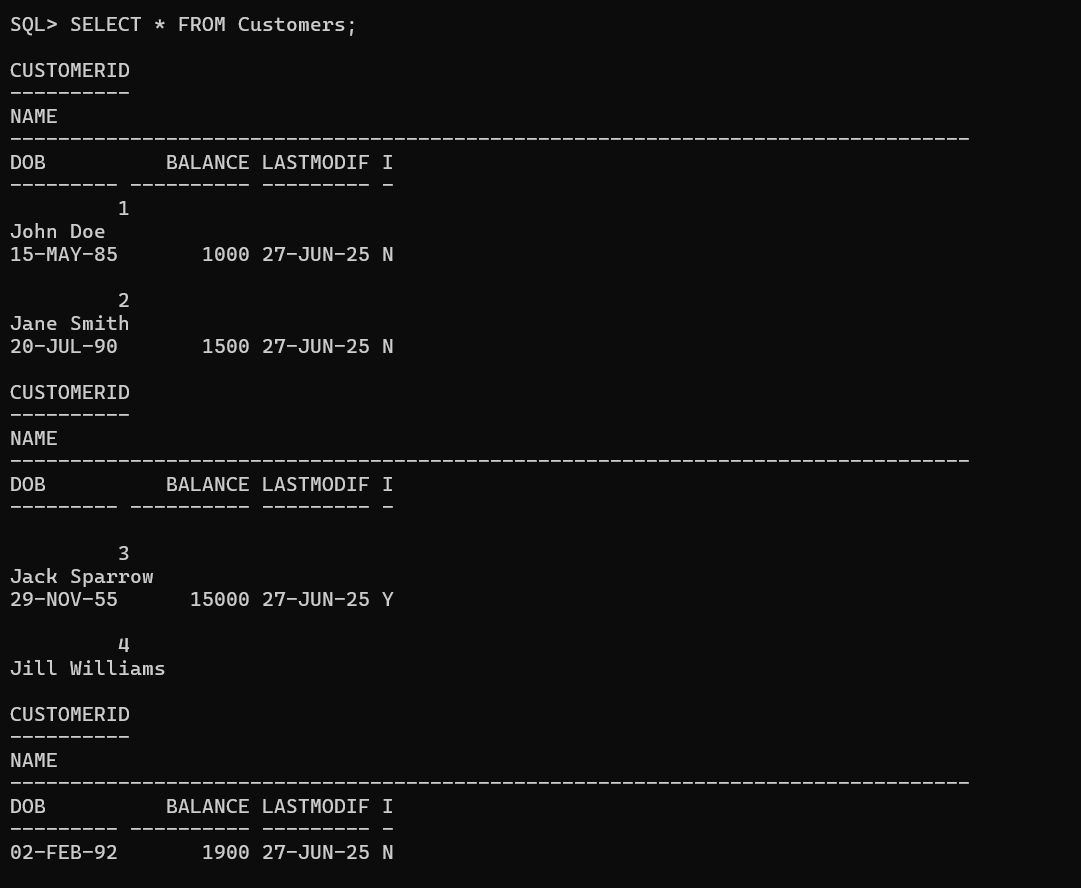
END IF;

END LOOP;

COMMIT;

END;

/



**Scenario - 3:**

SET SERVEROUTPUT ON;

BEGIN

FOR loan IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

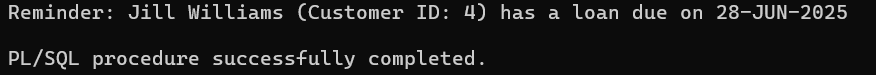
) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan.Name || ' (Customer ID: ' || loan.CustomerID || ') has a loan due on ' || TO\_CHAR(loan.EndDate, 'DD-MON-YYYY'));

END LOOP;

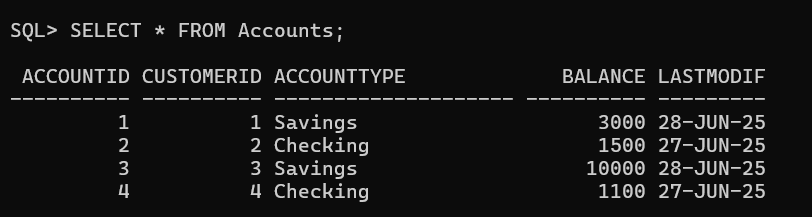
END;

/



**Exercise 3: Stored Procedures**

**Scenario - 1:**



CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = acc.Balance \* 1.01, -- Add 1% interest

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

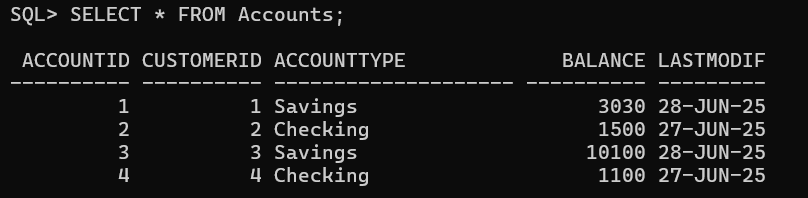
END LOOP;

COMMIT;

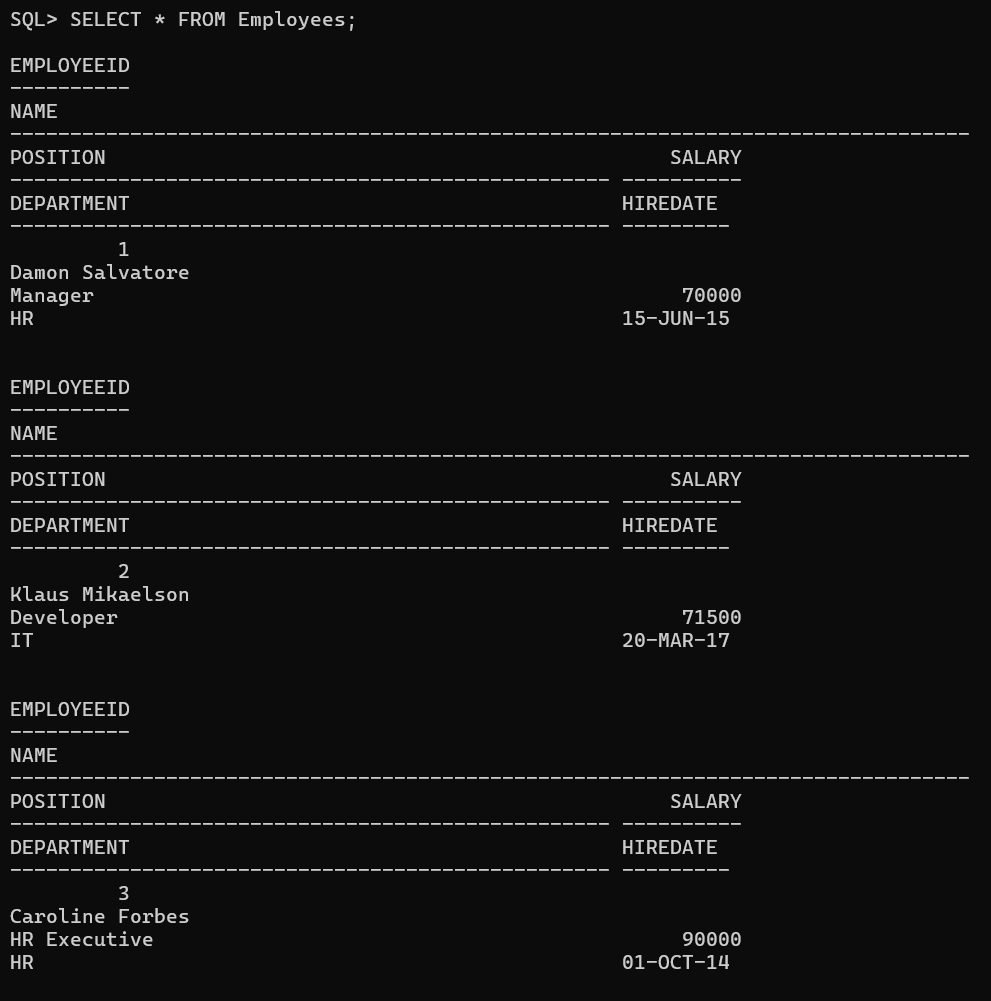
END;

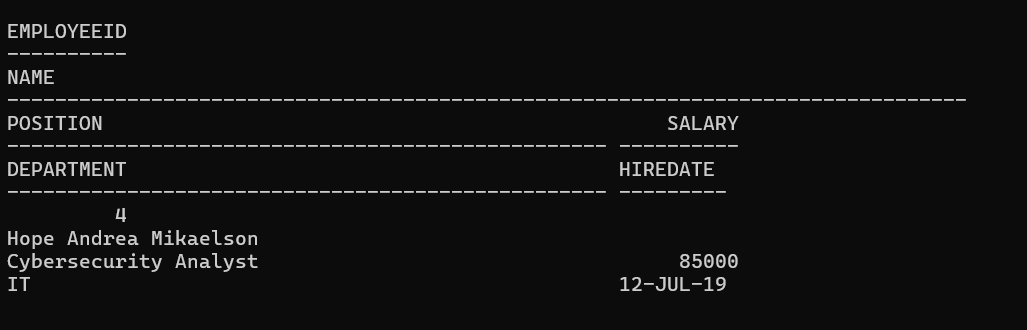
/

EXEC ProcessMonthlyInterest;



**Scenario - 2:**





CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

deptName IN VARCHAR2,

bonusPercent IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

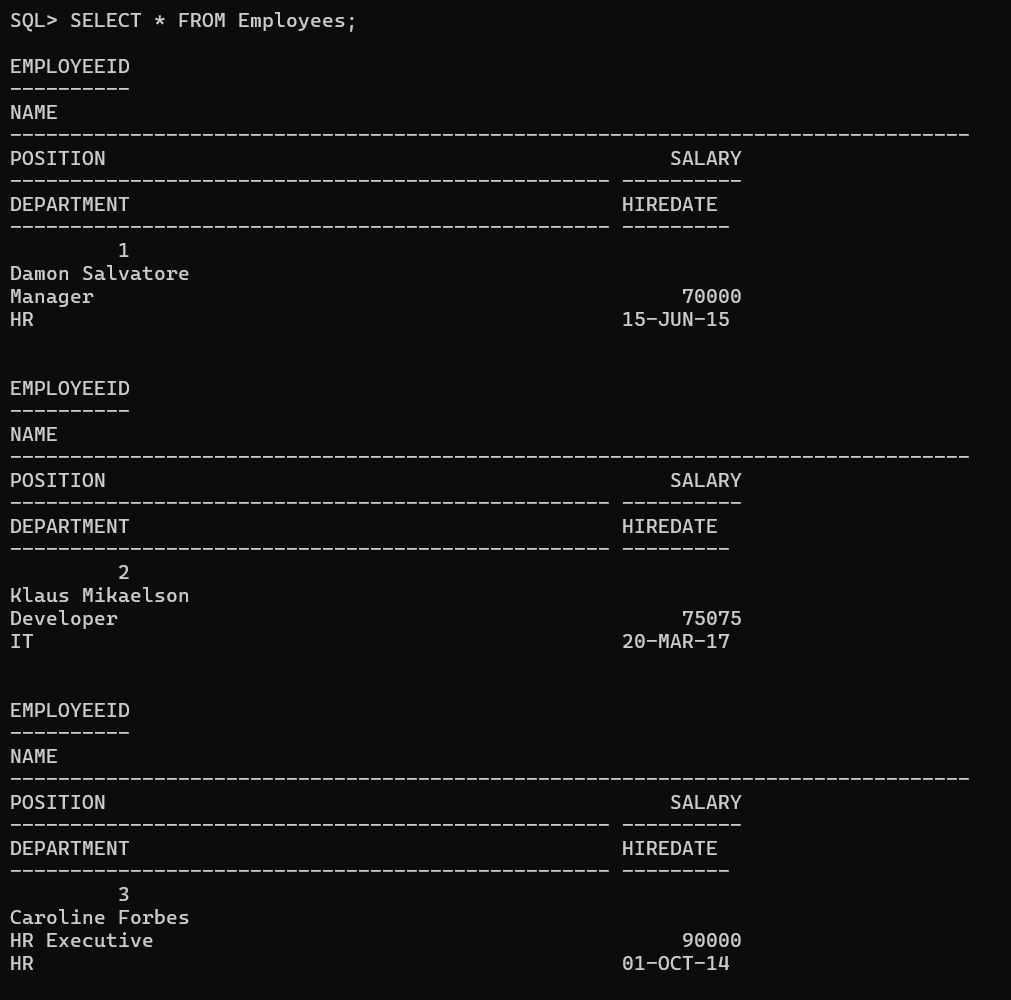
WHERE Department = deptName;

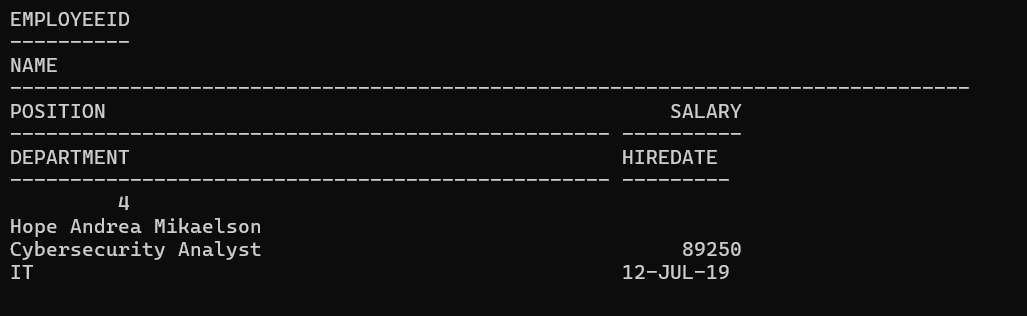
COMMIT;

END;

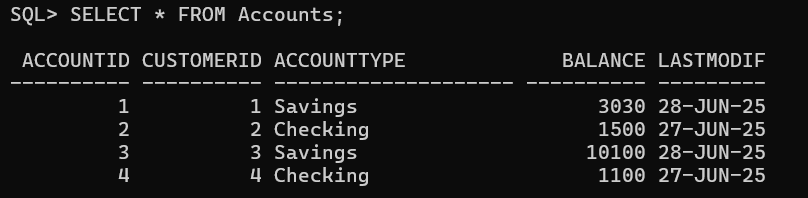
/

EXEC UpdateEmployeeBonus('IT', 5);





**Scenario - 3:**



CREATE OR REPLACE PROCEDURE TransferFunds(

fromAccID IN NUMBER,

toAccID IN NUMBER,

amount IN NUMBER

) AS

fromBal NUMBER;

BEGIN

-- Get balance of source account

SELECT Balance INTO fromBal

FROM Accounts

WHERE AccountID = fromAccID;

-- Check if balance is sufficient

IF fromBal < amount THEN

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

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - amount,

LastModified = SYSDATE

WHERE AccountID = fromAccID;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + amount,

LastModified = SYSDATE

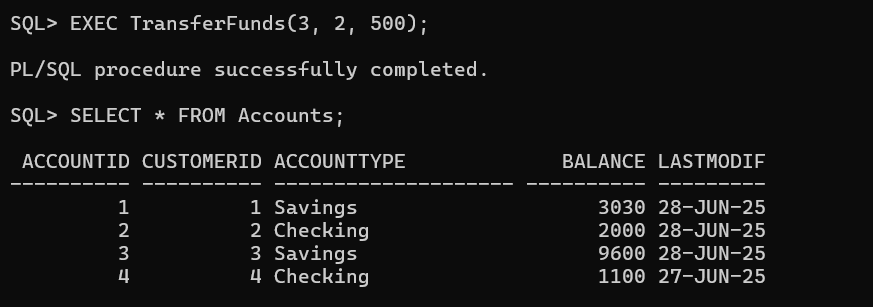
WHERE AccountID = toAccID;

COMMIT;

END;

/

EXEC TransferFunds(3, 2, 500);



**Additional**

**Exercise 2: Error Handling**

**Scenario - 1:**



CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

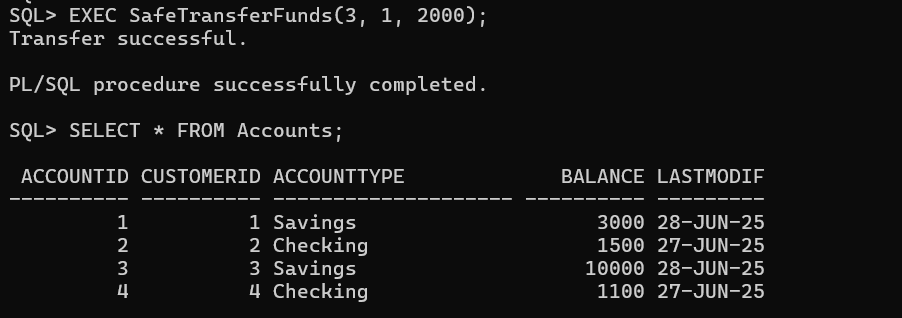
ROLLBACK;

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

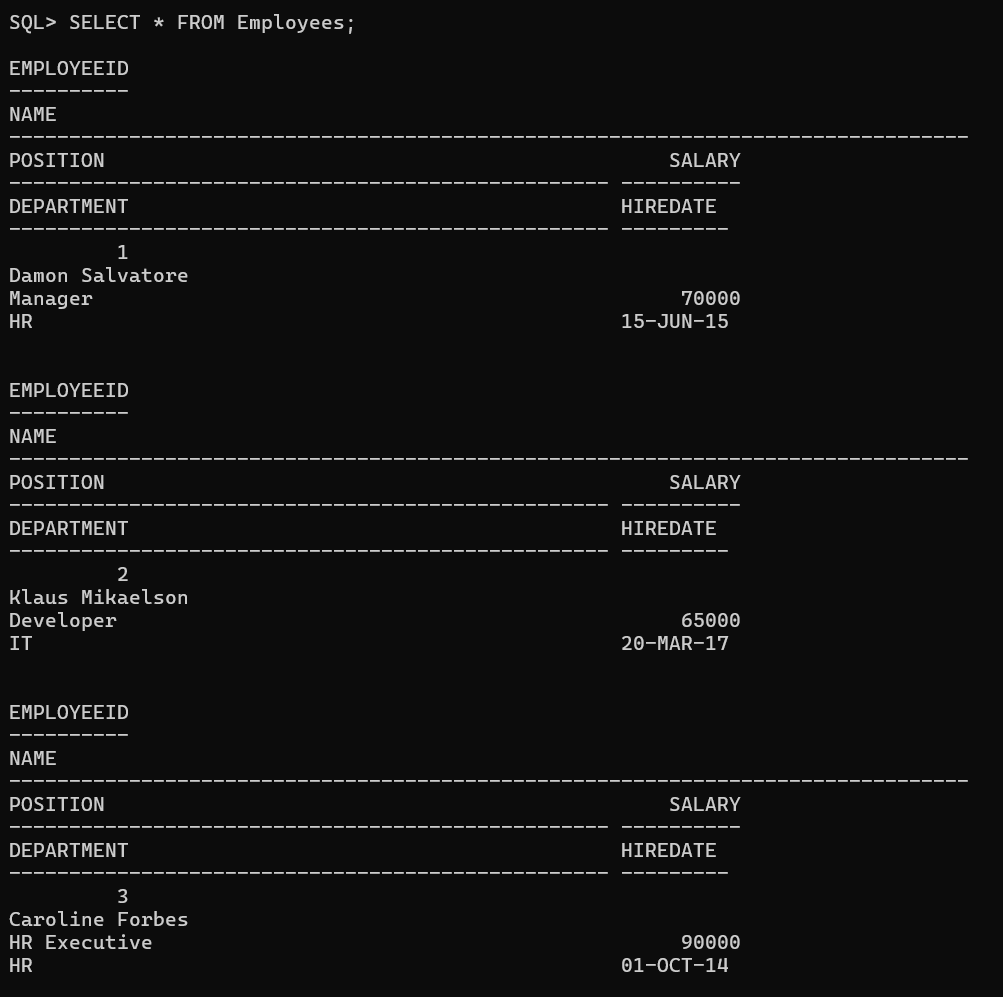
END;

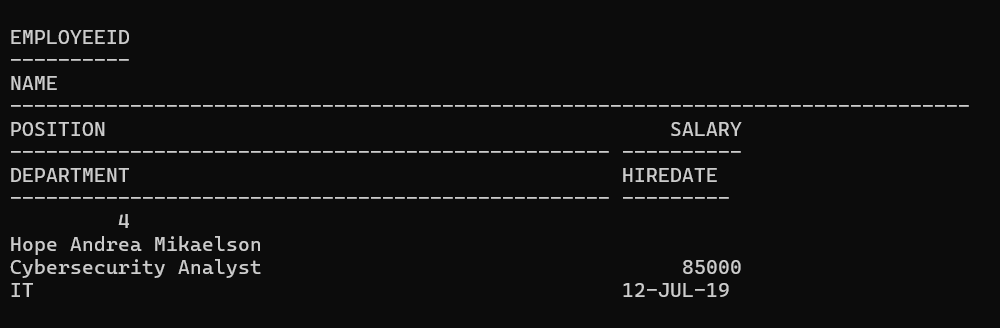
/

EXEC SafeTransferFunds(3, 1, 2000);



**Scenario - 2:**





CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

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

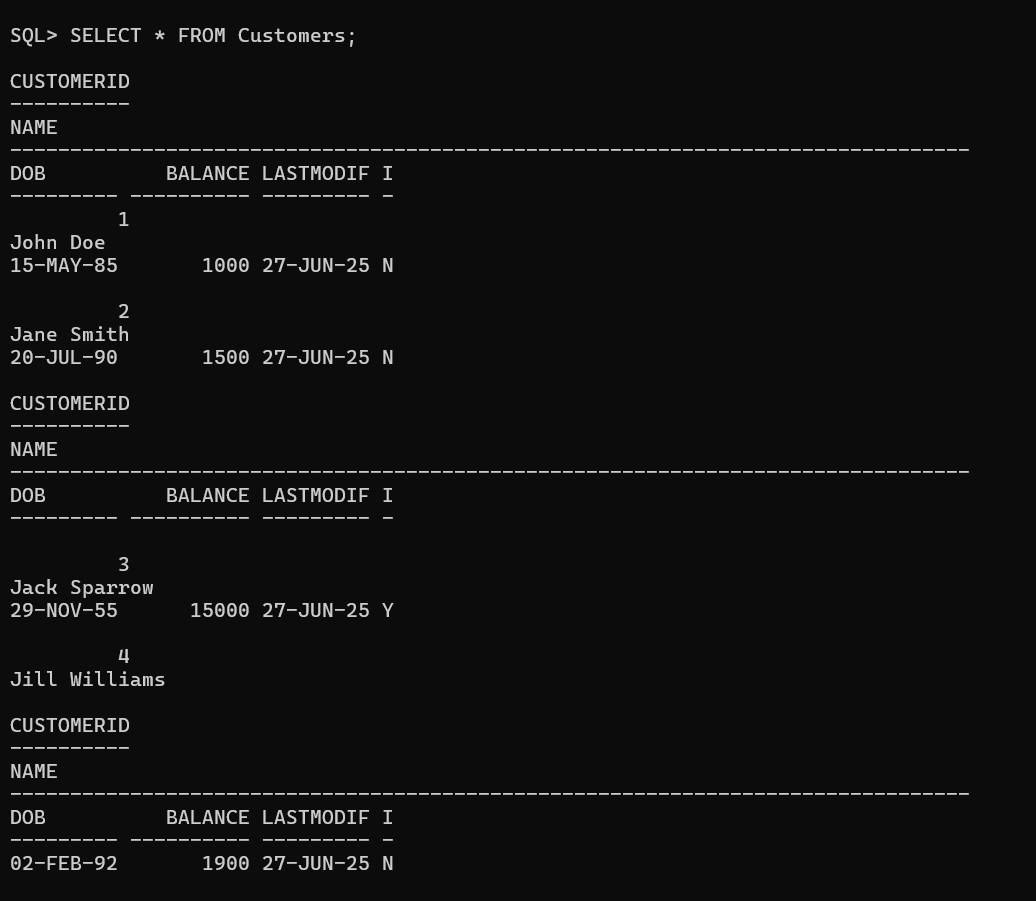
END;

/

EXEC UpdateSalary(2, 10);



**Scenario - 3:**



CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_isvip IN CHAR DEFAULT 'N'

) AS

BEGIN

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

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

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

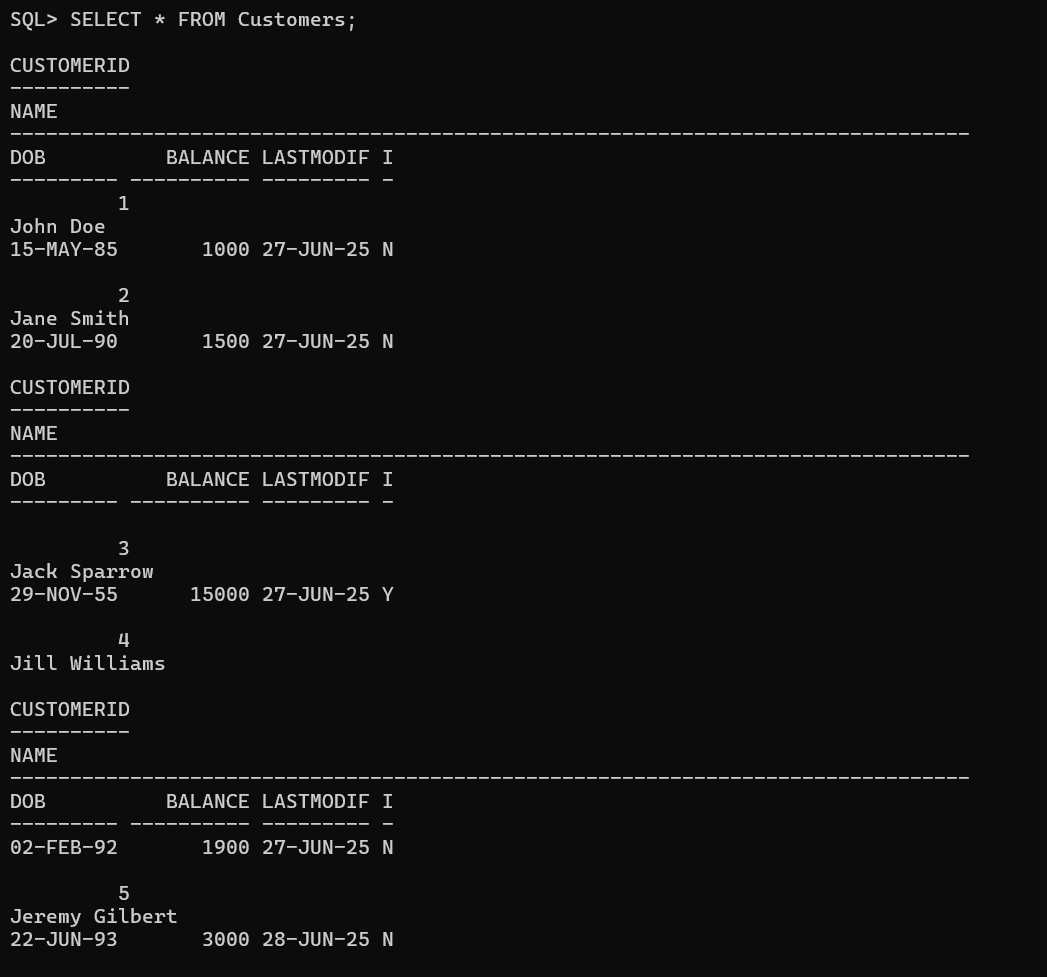
WHEN OTHERS THEN

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

END;

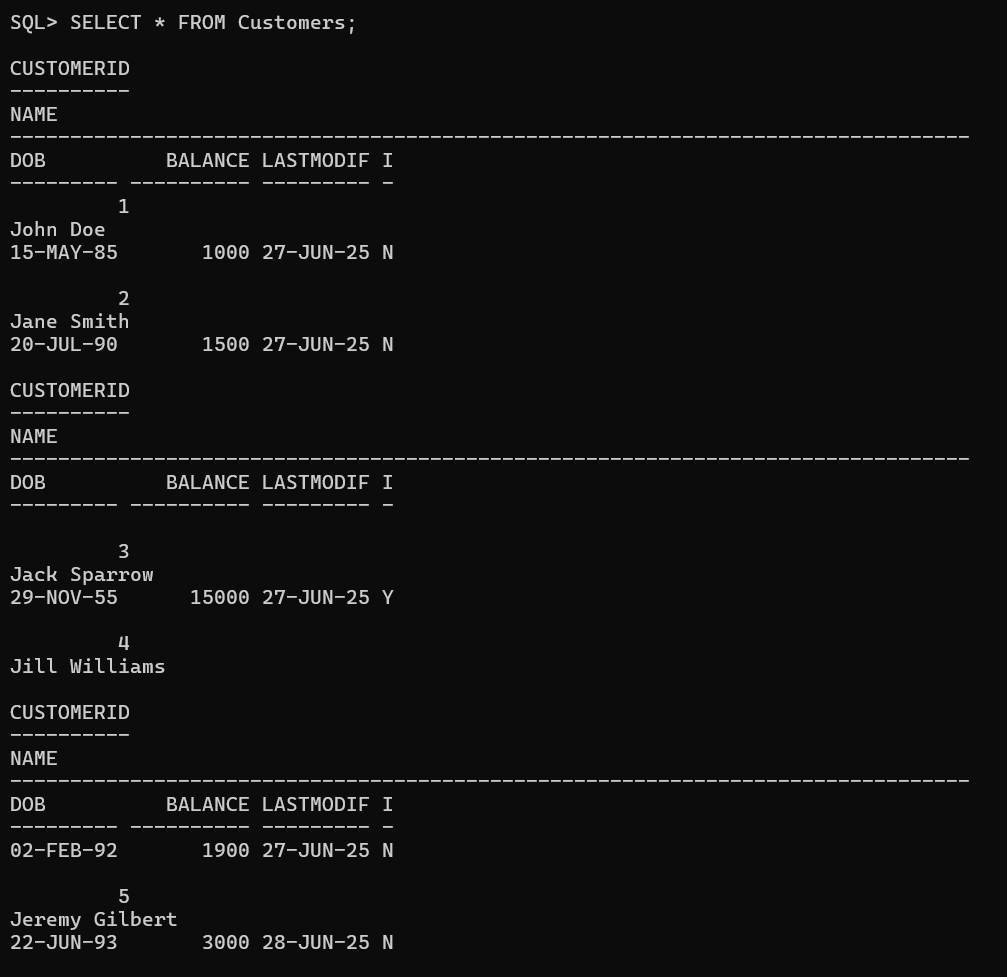
/

EXEC AddNewCustomer(5, 'Elena Gilbert', TO\_DATE('1993-06-22', 'YYYY-MM-DD'), 3000, 'N');



**Exercise 4: Functions**

**Scenario - 1:**



CREATE OR REPLACE FUNCTION CalculateAge(dob DATE)

RETURN NUMBER IS

age NUMBER;

BEGIN

age := FLOOR(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

RETURN age;

END;

/

SELECT

CustomerID,

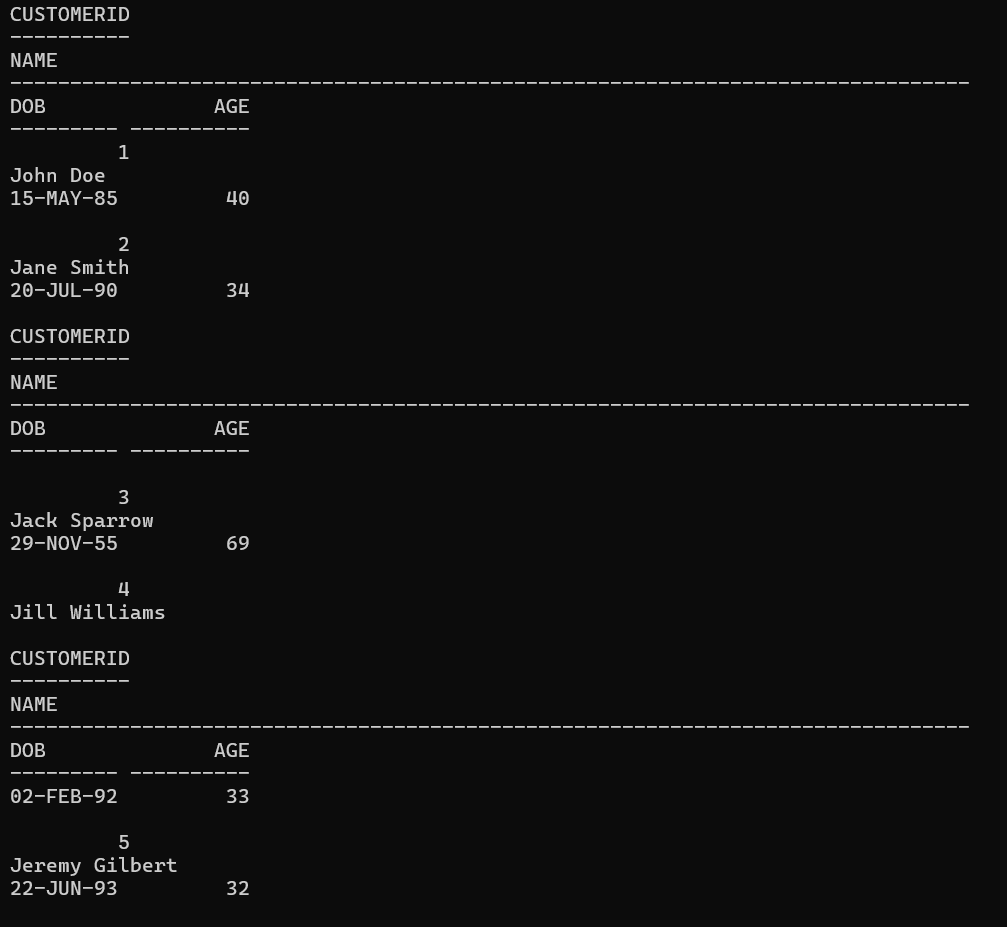
Name,

DOB,

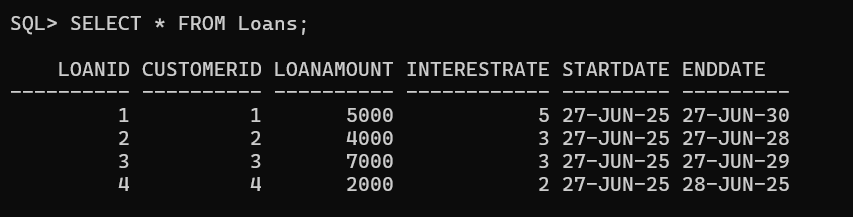
CalculateAge(DOB) AS Age

FROM

Customers;



**Scenario - 2:**



CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loanAmount NUMBER,

interestRate NUMBER,

durationYears NUMBER

)

RETURN NUMBER IS

monthlyInstallment NUMBER;

r NUMBER := interestRate / (12 \* 100); -- Monthly interest rate

n NUMBER := durationYears \* 12; -- Total months

BEGIN

monthlyInstallment := loanAmount \* r / (1 - POWER(1 + r, -n));

RETURN ROUND(monthlyInstallment, 2);

END;

/

SELECT

LoanID,

CustomerID,

LoanAmount,

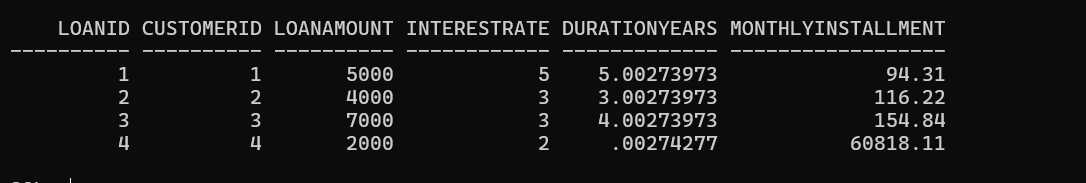
InterestRate,

(ENDDATE - STARTDATE)/365 AS DurationYears,

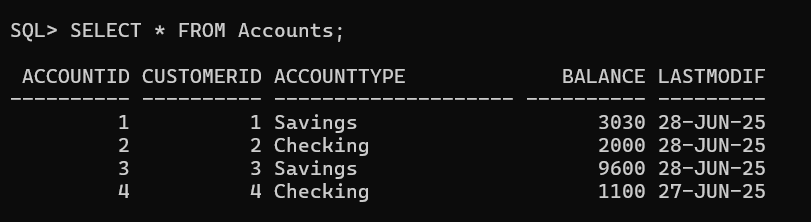
CalculateMonthlyInstallment(LoanAmount, InterestRate, (ENDDATE - STARTDATE)/365) AS MonthlyInstallment

FROM

Loans;



**Scenario - 3:**



CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

) RETURN VARCHAR2 IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_AccountID;

IF v\_Balance >= p\_Amount THEN

RETURN 'Y';

ELSE

RETURN 'N';

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'N';

END;

/

SELECT

AccountID,

CustomerID,

AccountType,

Balance,

HasSufficientBalance(AccountID, 2000) AS HasEnoughBalance

FROM

Accounts;



**Exercise 5: Triggers**

**Scenario - 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

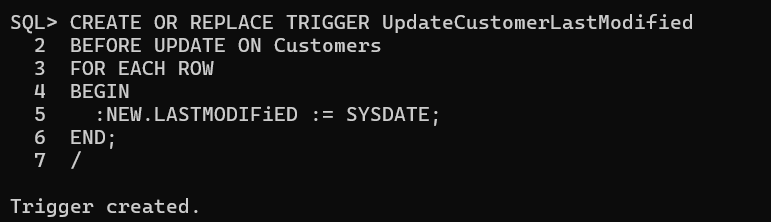
FOR EACH ROW

BEGIN

:NEW.LASTMODIFIED := SYSDATE;

END;

/



**Scenario - 2:**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

Action VARCHAR2(50),

LogDate DATE DEFAULT SYSDATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, Action)

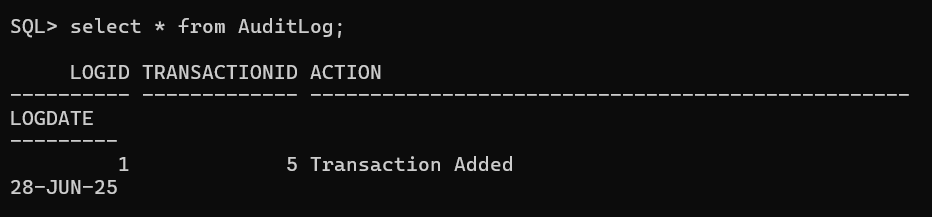
VALUES (:NEW.TransactionID, 'Transaction Added');

END;

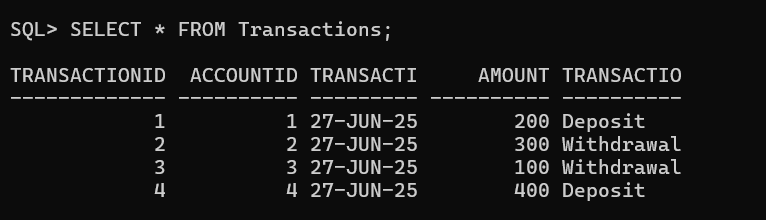
/

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

VALUES (5, 2, SYSDATE, 500, 'Deposit');



**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 = 'Deposit' AND :NEW.Amount <= 0 THEN

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

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

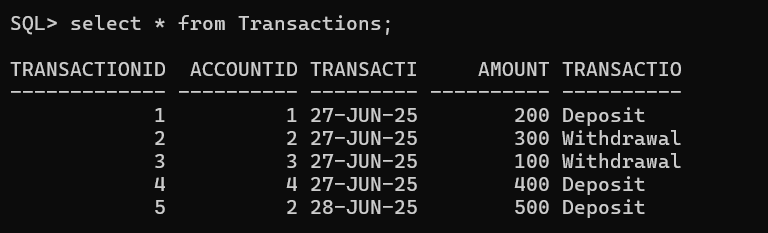
RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

END;

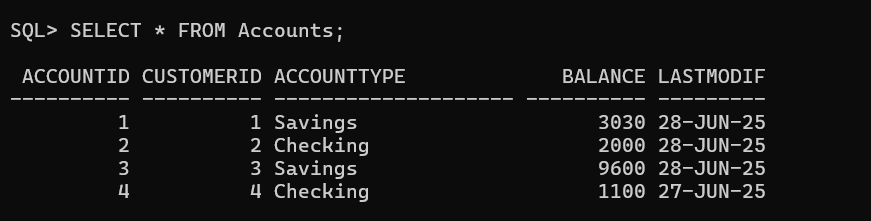
/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (5, 2, SYSDATE, 500, 'Deposit');



**Exercise 6: Cursors**

**Scenario - 1:**



BEGIN

FOR rec IN (

SELECT t.AccountID, t.TransactionDate, t.Amount, t.TransactionType

FROM Transactions t

WHERE TO\_CHAR(t.TransactionDate, 'MM-YYYY') = TO\_CHAR(SYSDATE, 'MM-YYYY')

ORDER BY t.AccountID, t.TransactionDate

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.AccountID ||

', Date: ' || TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY') ||

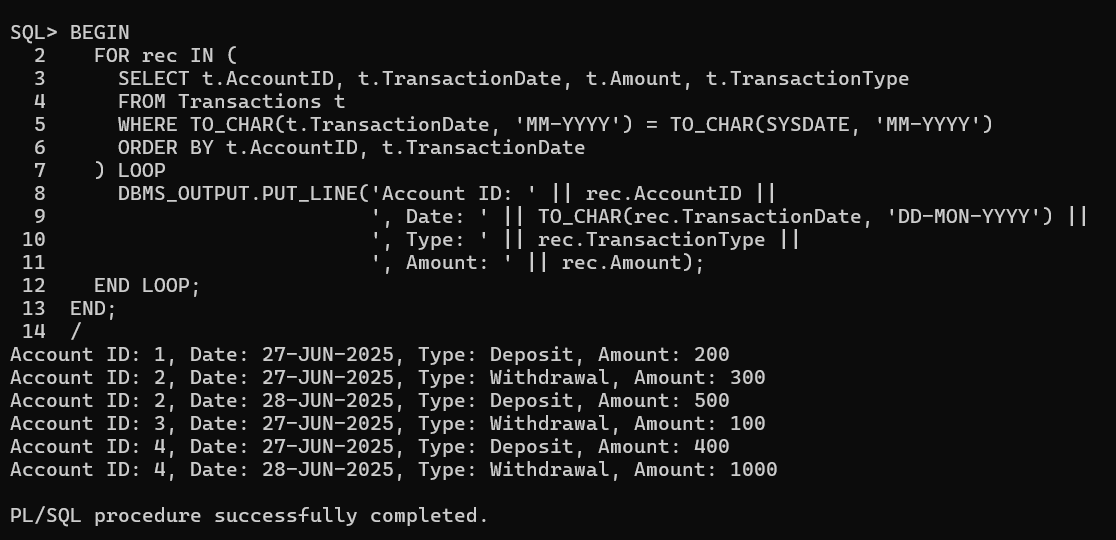
', Type: ' || rec.TransactionType ||

', Amount: ' || rec.Amount);

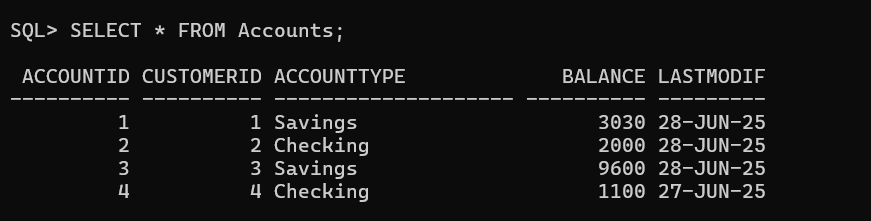
END LOOP;

END;

/



**Scenario - 2:**



DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_fee CONSTANT NUMBER := 100;

BEGIN

FOR rec IN acc\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - v\_fee,

LastModif = SYSDATE

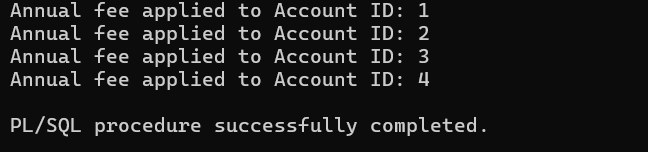
WHERE AccountID = rec.AccountID;

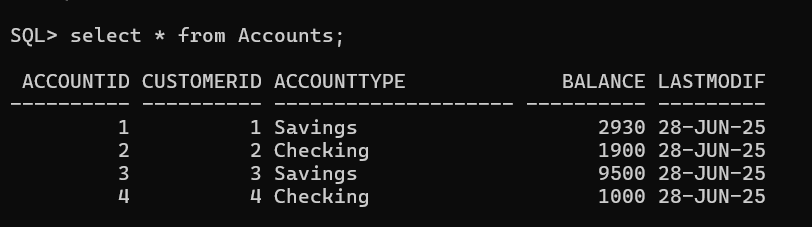
DBMS\_OUTPUT.PUT\_LINE('Annual fee applied to Account ID: ' || rec.AccountID);

END LOOP;

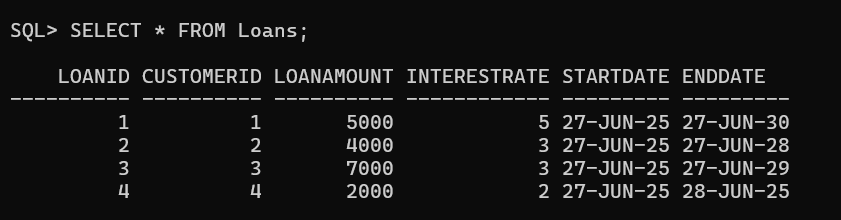
END;

/





**Scenario - 3:**



DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, LoanAmount, InterestRate FROM Loans;

v\_new\_rate NUMBER;

BEGIN

FOR rec IN loan\_cursor LOOP

IF rec.LoanAmount < 5000 THEN

v\_new\_rate := rec.InterestRate + 1;

ELSE

v\_new\_rate := rec.InterestRate + 0.5;

END IF;

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = rec.LoanID;

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

', New Interest Rate: ' || v\_new\_rate);

END LOOP;

END;

/

