PL/SQL programming

Exercise 1: Control Structures

Creating customers and loans table

-- Create customers table

CREATE TABLE customers (

  customer\_id   NUMBER PRIMARY KEY,

  name          VARCHAR2(20),

  age           NUMBER,

  balance       NUMBER,

  isvip         VARCHAR2(5)

);

-- Create loans table

CREATE TABLE loans (

  loan\_id       NUMBER PRIMARY KEY,

  customer\_id   NUMBER REFERENCES customers(customer\_id),

  due\_date      DATE,

  interest\_rate NUMBER

);

-- Insert sample customer data

INSERT INTO customers VALUES (1, 'John Doe', 65, 15000, 'FALSE');

INSERT INTO customers VALUES (2, 'Jane Smith', 45, 9500, 'FALSE');

INSERT INTO customers VALUES (3, 'Emily Davis', 70, 20000, 'FALSE');

INSERT INTO customers VALUES (4, 'Mark Thomas', 55, 12000, 'FALSE');

INSERT INTO customers VALUES (5, 'Asha Menon', 62, 8000, 'FALSE');

-- Insert sample loan data

INSERT INTO loans VALUES (101, 1, SYSDATE + 10, 9.5);

INSERT INTO loans VALUES (102, 2, SYSDATE + 40, 10.0);

INSERT INTO loans VALUES (103, 3, SYSDATE + 5, 8.5);

INSERT INTO loans VALUES (104, 4, SYSDATE + 15, 9.0);

INSERT INTO loans VALUES (105, 5, SYSDATE + 25, 9.8);

COMMIT;

-- All customers

SELECT \* FROM customers;

-- All loans

SELECT \* FROM loans;

CUSTOMER\_ID NAME AGE BALANCE ISVIP

1 John Doe 65 15000 FALSE

2 Jane Smith 45 9500 FALSE

3 Emily Davis 70 20000 FALSE

4 Mark Thomas 55 12000 FALSE

5 Asha Menon 62 8000 FALSE

*Customers Table Customers table*

*Customers table*

*Customers table*

LOAN\_ID CUSTOMER\_ID DUE\_DATE INTEREST\_RATE

101 1 06/07/25 9.5

102 2 05/08/25 10

103 3 01/07/25 8.5

104 4 11/07/25 9

105 5 21/07/25 9.8

*Loans Table*

**Scenario 1:**

The bank wants to apply a discount to loan interest rates for customers above 60 years old.

SET SERVEROUTPUT ON;

--Scenario 1: Apply 1% discount to interest for customers over 60

BEGIN

  FOR cust IN (SELECT customer\_id FROM customers WHERE age > 60) LOOP

    UPDATE loans

    SET interest\_rate = interest\_rate - 1

    WHERE customer\_id = cust.customer\_id;

  END LOOP;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Scenario 1: Interest rate discount applied to customers over 60.');

END;

/

SELECT

  c.name AS customer\_name,

  c.age,

  l.loan\_id,

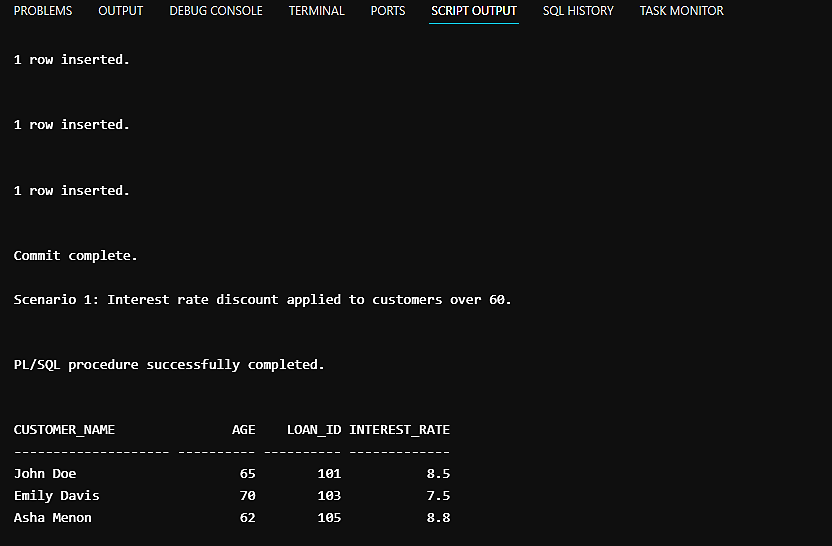
  l.interest\_rate

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id

WHERE c.age > 60;

OUTPUT:



**Scenario 2:**

A customer can be promoted to VIP status based on their balance.

-- Scenario 2: Set IsVIP flag for customers with balance > 10,000

BEGIN

  FOR cust IN (SELECT customer\_id FROM customers WHERE balance > 10000) LOOP

    UPDATE customers

    SET isvip = 'TRUE'

    WHERE customer\_id = cust.customer\_id;

  END LOOP;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Scenario 2: VIP flag set for high balance customers.');

END;

SELECT

  customer\_id,

  name,

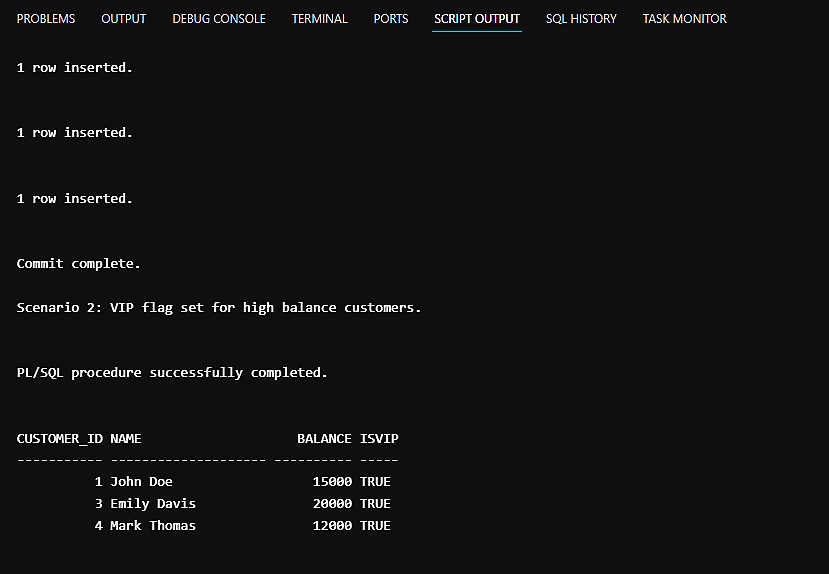
  balance,

  isvip

FROM customers

WHERE balance > 10000;

OUTPUT:



**Scenario 3:**

The bank wants to send reminders to customers whose loans are due within the next 30 days.

--Scenario 3: Send reminders for loans due in next 30 days

BEGIN

  FOR loan\_rec IN (

    SELECT l.loan\_id, l.due\_date, c.name

    FROM loans l

    JOIN customers c ON l.customer\_id = c.customer\_id

    WHERE l.due\_date <= SYSDATE + 30

  ) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.loan\_id ||

                         ' for ' || loan\_rec.name ||

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

  END LOOP;

END;

/

SELECT

  l.loan\_id,

  c.name AS customer\_name,

  l.due\_date,

  ROUND(l.due\_date - SYSDATE) AS days\_left

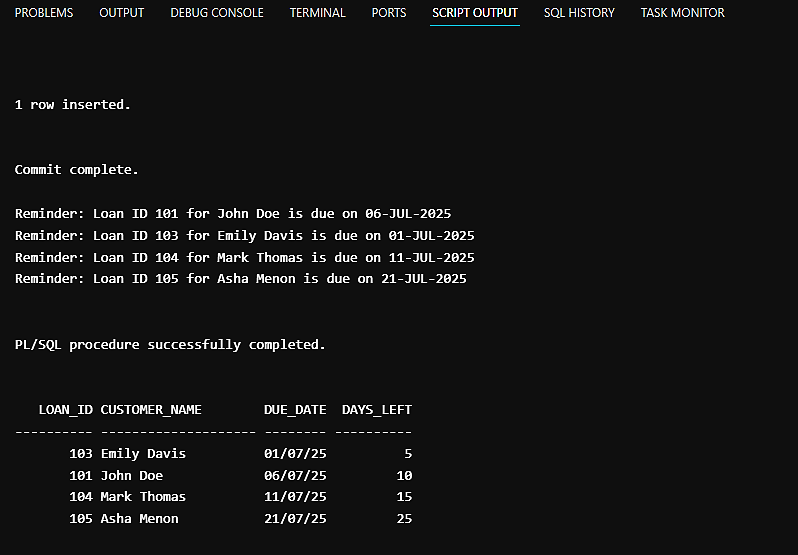
FROM loans l

JOIN customers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

ORDER BY l.due\_date;

OUTPUT:



Exercise 3: Stored Procedures

Creating accounts and employees table

CREATE TABLE accounts (

  account\_id NUMBER PRIMARY KEY,

  account\_type VARCHAR2(20),

  balance NUMBER

);

CREATE TABLE employees (

  employee\_id NUMBER PRIMARY KEY,

  department\_id NUMBER,

  salary NUMBER

);

-- Sample data for accounts

INSERT INTO accounts VALUES (1, 'SAVINGS', 1000);

INSERT INTO accounts VALUES (2, 'SAVINGS', 2000);

INSERT INTO accounts VALUES (3, 'CURRENT', 5000);

-- Sample data for employees

INSERT INTO employees VALUES (101, 10, 50000);

INSERT INTO employees VALUES (102, 10, 60000);

INSERT INTO employees VALUES (103, 20, 55000);

COMMIT;

-- All employees

SELECT \* FROM employees;

-- All accounts

SELECT \* FROM accounts;

ACCOUNT\_ID ACCOUNT\_TYPE BALANCE

1 SAVINGS 1010

2 SAVINGS 1520

3 CURRENT 5500

*Accounts Table Customers table*

*Customers table*

*Customers table*

EMPLOYEE\_ID DEPARTMENT\_ID SALARY

101 10 55000

102 10 66000

103 20 55000

*Employees Table Customers table*

*Customers table*

*Customers table*

**Scenario 1:**

The bank needs to process monthly interest for all savings accounts.

-- Scenario 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  UPDATE accounts

  SET balance = balance \* 1.01

  WHERE account\_type = 'SAVINGS';

  COMMIT;

END;

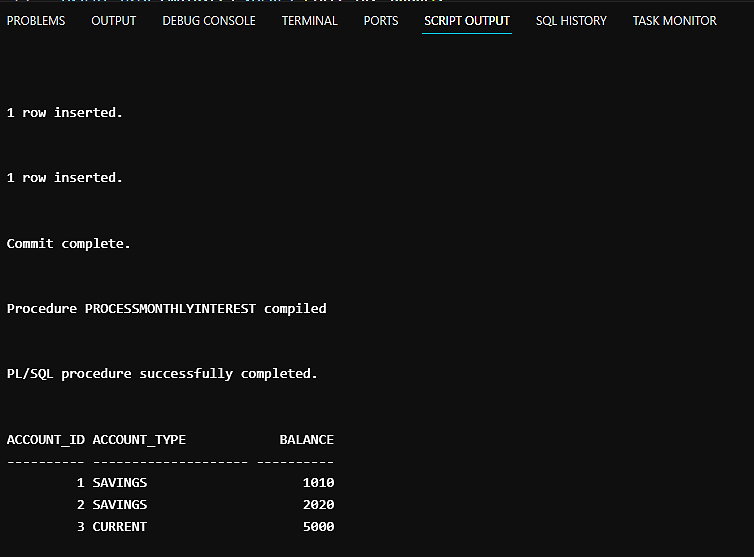
/

EXEC ProcessMonthlyInterest;

-- View updated balances

SELECT \* FROM accounts;

OUTPUT:



**Scenario 2:**

The bank wants to implement a bonus scheme for employees based on their performance.

--Scenario 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

  p\_department\_id IN NUMBER,

  p\_bonus\_percent IN NUMBER

) IS

BEGIN

  UPDATE employees

  SET salary = salary + (salary \* p\_bonus\_percent / 100)

  WHERE department\_id = p\_department\_id;

  COMMIT;

  END;

/

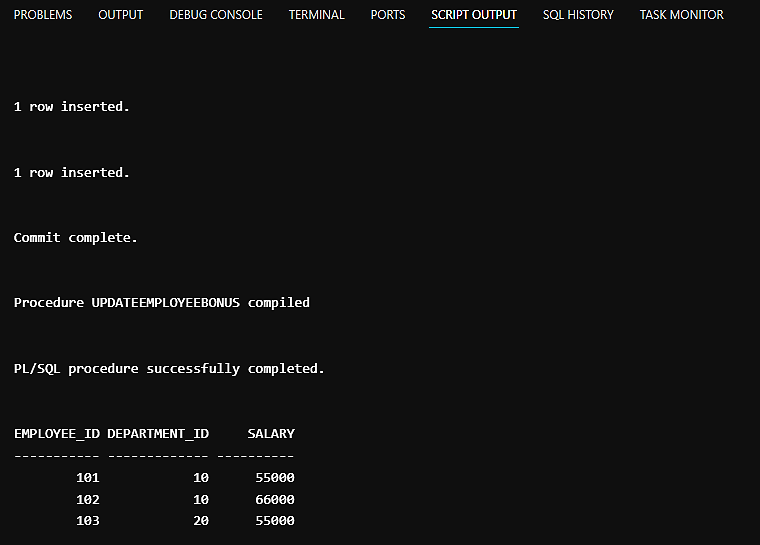
-- 10% bonus for dept 10

 EXEC UpdateEmployeeBonus(10, 10);

-- View updated salaries

SELECT \* FROM employees;

OUTPUT:



**Scenario 3:**

Customers should be able to transfer funds between their accounts.

-- Scenario 3: TransferFunds

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 account\_id = p\_from\_account\_id;

  IF v\_balance < p\_amount THEN

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

  END IF;

  UPDATE accounts

  SET balance = balance - p\_amount

  WHERE account\_id = p\_from\_account\_id;

  UPDATE accounts

  SET balance = balance + p\_amount

  WHERE account\_id = p\_to\_account\_id;

  COMMIT;

END;

/

-- Transfer 500 from acc 2 to acc 3

EXEC TransferFunds(2, 3, 500);

-- View updated balances

SELECT \* FROM accounts;

OUTPUT:

