

Exercise 1: Control Structures

Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

- **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.

Scenario 2: A customer can be promoted to VIP status based on their balance.

- **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.

Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.

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

Procedure:

Step 1: Create Tables

```
DROP TABLE loans CASCADE CONSTRAINTS;
```

```
DROP TABLE customers CASCADE CONSTRAINTS;
```

```
CREATE TABLE customers (  
    customer_id  NUMBER PRIMARY KEY,  
    name        VARCHAR2(50),  
    age         NUMBER,  
    balance     NUMBER(10, 2),  
    is_vip      VARCHAR2(5) DEFAULT 'FALSE'  
);
```

```
CREATE TABLE loans (  
    loan_id     NUMBER PRIMARY KEY,  
    customer_id NUMBER REFERENCES customers(customer_id),  
    interest_rate NUMBER(5, 2),  
    due_date    DATE  
);
```

Step 2: Insert Sample Data

```
INSERT INTO customers VALUES (1, 'Alice', 65, 15000, 'N');
```

```
INSERT INTO customers VALUES (2, 'Bob', 55, 8000, 'N');
```

```
INSERT INTO customers VALUES (3, 'Charlie', 70, 12000, 'N');
```

```
INSERT INTO loans VALUES (101, 1, 7.5, SYSDATE + 10);
INSERT INTO loans VALUES (102, 2, 8.0, SYSDATE + 40);
INSERT INTO loans VALUES (103, 3, 6.5, SYSDATE + 5);
```

```
COMMIT;
```

Step 3: Scenario 1

Program:

```
BEGIN
    FOR rec IN (
        SELECT l.LoanID, l.InterestRate
        FROM Loans l
        JOIN Customers c ON l.CustomerID = c.CustomerID
        WHERE c.Age > 60
    ) LOOP
        UPDATE Loans
        SET InterestRate = rec.InterestRate - 1
        WHERE LoanID = rec.LoanID;
    END LOOP;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Scenario 1: Discount applied to senior customers.');
```

END;

/

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL HISTORY  TASK MONITOR

SQL> SELECT l.LoanID,c.Name,l.DueDate
2  FROM Loans l
3  JOIN Customers c ON l.CustomerID=c.CustomerID
4* WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE+30;

  LOANID NAME      DUEDATE
-----
    101 Alice      08-07-25
    103 Charlie    03-07-25
```

Step 4: Scenario 2

Program:

BEGIN

```
FOR rec IN (  
    SELECT CustomerID FROM Customers WHERE Balance > 10000  
) LOOP  
    UPDATE Customers  
    SET IsVIP = 'Y'  
    WHERE CustomerID = rec.CustomerID;  
END LOOP;
```

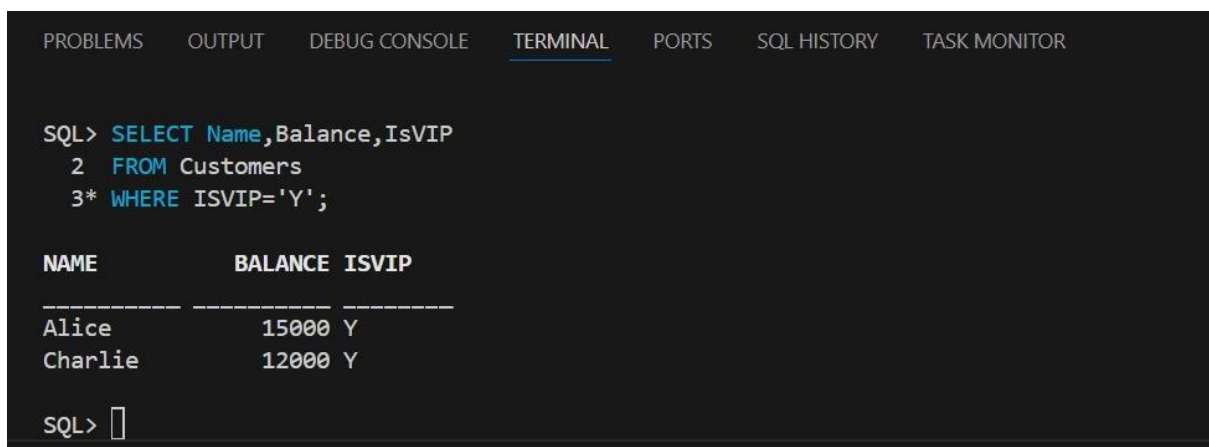
COMMIT;

DBMS_OUTPUT.PUT_LINE('Scenario 2: VIP status updated.');

END;

/

Out Put:



The screenshot shows a SQL Developer terminal window with the following tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), PORTS, SQL HISTORY, and TASK MONITOR. The terminal displays the following SQL query and its results:

```
SQL> SELECT Name,Balance,IsVIP  
2 FROM Customers  
3* WHERE ISVIP='Y';
```

NAME	BALANCE	ISVIP
Alice	15000	Y
Charlie	12000	Y

The terminal also shows the prompt SQL> with a cursor.

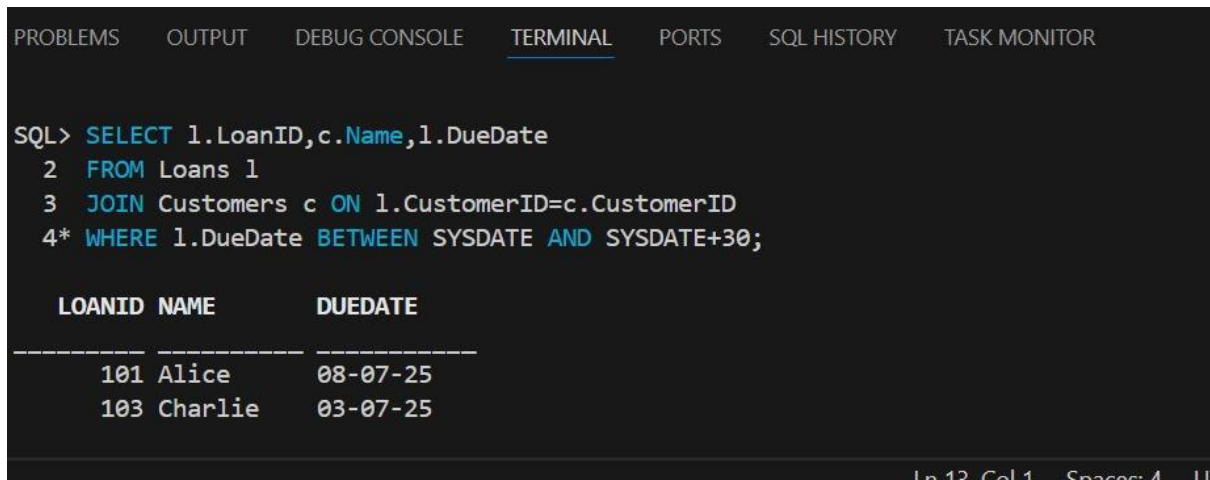
Step 5: Scenario 3

BEGIN

```
FOR rec IN (  
    SELECT l.LoanID, c.Name, l.DueDate  
    FROM Loans l  
    JOIN Customers c ON l.CustomerID = c.CustomerID  
    WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30  
) LOOP
```

```
        DBMS_OUTPUT.PUT_LINE('Reminder: Loan #' || rec.LoanID || ' for ' || rec.Name ||  
                               ' is due on ' || TO_CHAR(rec.DueDate, 'DD-MON-YYYY'));  
    END LOOP;  
END;  
  
/
```

Out put:



The screenshot shows a SQL IDE interface with a terminal window. The terminal displays the following SQL query and its results:

```
SQL> SELECT 1.LoanID,c.Name,1.DueDate  
2 FROM Loans l  
3 JOIN Customers c ON l.CustomerID=c.CustomerID  
4* WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE+30;
```

LOANID	NAME	DUE DATE
101	Alice	08-07-25
103	Charlie	03-07-25

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), PORTS, SQL HISTORY, and TASK MONITOR. The status bar at the bottom right shows 'Ln 13, Col 1, Spaces: 4, LF'.

Exercise 3: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

- **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.

Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

- **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.

Scenario 3: Customers should be able to transfer funds between their accounts.

- **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.

Procedure:

Step 1: Create Tables

```
CREATE TABLE Accounts (  
    AccountID NUMBER PRIMARY KEY,  
    AccountType VARCHAR2(20),  
    Balance NUMBER  
);  
  
CREATE TABLE Employees (  
    EmployeeID NUMBER PRIMARY KEY,  
    Name VARCHAR2(100),  
    Department VARCHAR2(50),  
    Salary NUMBER  
);
```

Step 2: Insert Table data

```
-- Insert Accounts  
  
INSERT INTO Accounts VALUES (101, 'Savings', 1000);  
INSERT INTO Accounts VALUES (102, 'Savings', 1500);
```

```
INSERT INTO Accounts VALUES (103, 'Checking', 2000);
```

```
-- Insert Employees
```

```
INSERT INTO Employees VALUES (1, 'Alice', 'HR', 50000);
```

```
INSERT INTO Employees VALUES (2, 'Bob', 'Finance', 60000);
```

```
INSERT INTO Employees VALUES (3, 'Charlie', 'HR', 55000);
```

```
COMMIT;
```

Step 3: Process monthly interest

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
```

```
BEGIN
```

```
    UPDATE Accounts
```

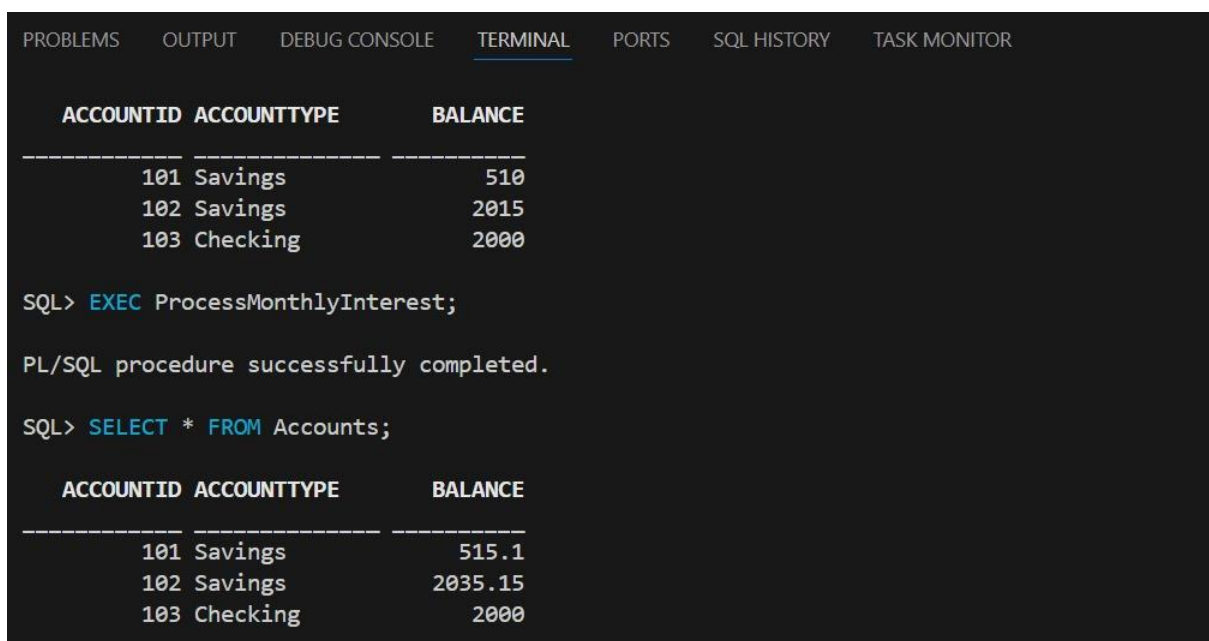
```
    SET Balance = Balance + (Balance * 0.01)
```

```
    WHERE AccountType = 'Savings';
```

```
COMMIT;
```

```
END;
```

```
/
```



The screenshot shows a SQL terminal window with the following content:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL HISTORY  TASK MONITOR
```


ACCOUNTID	ACCOUNTTYPE	BALANCE
101	Savings	510
102	Savings	2015
103	Checking	2000


```
SQL> EXEC ProcessMonthlyInterest;
```

PL/SQL procedure successfully completed.

```
SQL> SELECT * FROM Accounts;
```

ACCOUNTID	ACCOUNTTYPE	BALANCE
101	Savings	515.1
102	Savings	2035.15
103	Checking	2000

Step 4: UpdateEmployeeBonus

```
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;
/

```

Out put:

The screenshot shows a SQL terminal window with the following content:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL HISTORY  TASK MONITOR

SQL> SELECT * FROM Employees;

  EMPLOYEEID NAME      DEPARTMENT      SALARY
-----
1 Alice      HR              55000
2 Bob        Finance         60000
3 Charlie    HR              60500

SQL> -
SQL> EXEC UpdateEmployeeBonus('HR',10);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM Employees;

  EMPLOYEEID NAME      DEPARTMENT      SALARY
-----
1 Alice      HR              60500
2 Bob        Finance         60000
3 Charlie    HR              66550

```

Step 5:

```

CREATE OR REPLACE PROCEDURE TransferFunds(
    fromAccount IN NUMBER,
    toAccount IN NUMBER,

```

```

    amount IN NUMBER
) AS
    insufficient_balance EXCEPTION;
BEGIN
    -- Check balance

    DECLARE

        src_balance NUMBER;
    BEGIN

        SELECT Balance INTO src_balance FROM Accounts WHERE AccountID = fromAccount;

        IF src_balance < amount THEN

            RAISE insufficient_balance;

        END IF;


        -- Perform transfer

        UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAccount;

        UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAccount;


        COMMIT;

    END;

EXCEPTION

    WHEN insufficient_balance THEN

        DBMS_OUTPUT.PUT_LINE('Insufficient balance in source account. ');

        ROLLBACK;

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);

        ROLLBACK;

    END;

/

Out Put:

```


SQL> **SELECT** * **FROM** Accounts;

ACCOUNTID	ACCOUNTTYPE	BALANCE
101	Savings	515.1
102	Savings	2035.15
103	Checking	2000

SQL> **EXEC** TransferFunds(101, 102, 500);

PL/SQL procedure successfully completed.

SQL> **SELECT** * **FROM** Accounts;

ACCOUNTID	ACCOUNTTYPE	BALANCE
101	Savings	15.1
102	Savings	2535.15
103	Checking	2000