

# Week 2

## PL/SQL Programming \_HandsOn

### Control Structures

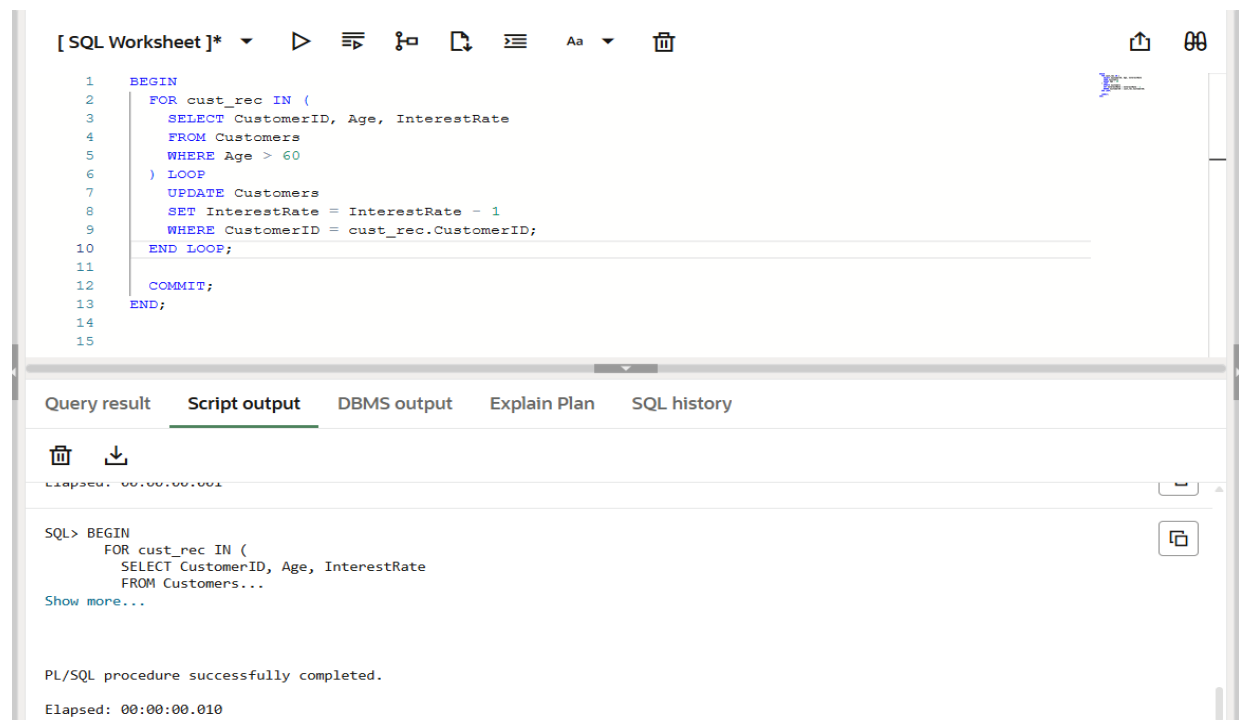
**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**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.

#### CODE:

```
BEGIN
  FOR cust_rec IN (
    SELECT CustomerID, Age, InterestRate
    FROM Customers
    WHERE Age > 60
  ) LOOP
    UPDATE Customers
    SET InterestRate = InterestRate - 1
    WHERE CustomerID = cust_rec.CustomerID;
  END LOOP;
  COMMIT;
END;
```

## OUTPUT :



The screenshot shows a SQL Worksheet interface. The top toolbar includes icons for running, saving, and other functions. The main editor contains a PL/SQL procedure:

```
1 BEGIN
2   FOR cust_rec IN (
3     SELECT CustomerID, Age, InterestRate
4     FROM Customers
5     WHERE Age > 60
6   ) LOOP
7     UPDATE Customers
8     SET InterestRate = InterestRate - 1
9     WHERE CustomerID = cust_rec.CustomerID;
10  END LOOP;
11
12  COMMIT;
13 END;
```

Below the editor, the 'Script output' tab is active, showing the execution results:

```
SQL> BEGIN
      FOR cust_rec IN (
        SELECT CustomerID, Age, InterestRate
        FROM Customers...
      ) LOOP
        UPDATE Customers
        SET InterestRate = InterestRate - 1
        WHERE CustomerID = cust_rec.CustomerID;
      END LOOP;
      COMMIT;
    END;
```

The output indicates that the PL/SQL procedure was successfully completed. The elapsed time is 00:00:00.010.

**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.**

## CODE :

```
BEGIN

FOR cust_rec IN (

  SELECT CustomerID, Balance

  FROM Customers

  WHERE Balance > 10000

) LOOP

  UPDATE Customers

  SET IsVIP = 'TRUE'
```

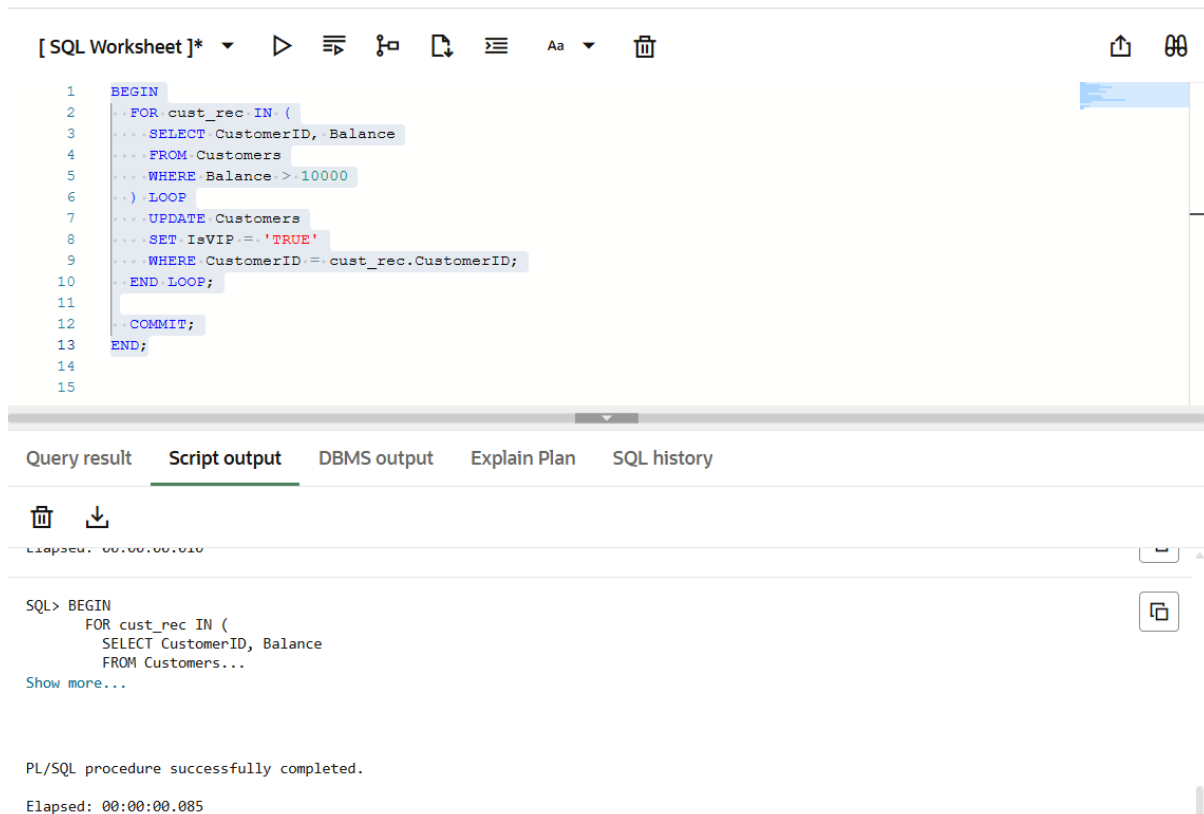
```
WHERE CustomerID = cust_rec.CustomerID;

END LOOP;

COMMIT;

END;

OUTPUT :
```



The screenshot displays a SQL worksheet interface. The top section contains a PL/SQL block with the following code:

```
1 BEGIN
2   FOR cust_rec IN (
3     SELECT CustomerID, Balance
4     FROM Customers
5     WHERE Balance > 10000
6   ) LOOP
7     UPDATE Customers
8     SET IsVIP = 'TRUE'
9     WHERE CustomerID = cust_rec.CustomerID;
10  END LOOP;
11
12  COMMIT;
13
14 END;
```

Below the code editor, the 'Script output' tab is active, showing the execution results:

```
SQL> BEGIN
      FOR cust_rec IN (
        SELECT CustomerID, Balance
        FROM Customers...
Show more...
```

At the bottom, a status message indicates: "PL/SQL procedure successfully completed." followed by the elapsed time: "Elapsed: 00:00:00.085".

**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.**

**CODE :**

```
DECLARE
```

```
v_due_date DATE := SYSDATE + 30;
```

```
BEGIN
```

```

FOR loan_rec IN (
    SELECT LoanID, CustomerID, DueDate
    FROM Loans
    WHERE DueDate <= v_due_date
) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ID ' || loan_rec.LoanID ||
        ' for Customer ID ' || loan_rec.CustomerID ||
        ' is due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-
YYYYY'));
END LOOP;
END;

```

## OUTPUT:

The screenshot shows an SQL worksheet interface with a toolbar at the top containing icons for running, saving, and other functions. The main area displays a PL/SQL script with line numbers 1 through 15. The script declares a variable `v_due_date` as `DATE` and sets it to `SYSDATE + 30`. It then enters a `FOR` loop over a query that selects `LoanID`, `CustomerID`, and `DueDate` from the `Loans` table where `DueDate` is less than or equal to `v_due_date`. Inside the loop, it uses `DBMS_OUTPUT.PUT_LINE` to print a reminder message for each loan. The output pane at the bottom shows the results of the script execution, including the reminder messages and a confirmation that the procedure completed successfully.

```

[ SQL Worksheet ]*  ▶  ⚙  🔍  📄  ⌵  Aa  🗑
1  DECLARE
2  | v_due_date DATE := SYSDATE + 30;
3  BEGIN
4  | FOR loan_rec IN (
5  |     SELECT LoanID, CustomerID, DueDate
6  |     FROM Loans
7  |     WHERE DueDate <= v_due_date
8  | ) LOOP
9  |     DBMS_OUTPUT.PUT_LINE('Reminder: Loan ID ' || loan_rec.LoanID ||
10 |         ' for Customer ID ' || loan_rec.CustomerID ||
11 |         ' is due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-YYYYY'));
12 | END LOOP;
13 END;
14
15
Query result  Script output  DBMS output  Explain Plan  SQL history
🗑  ⬇
SQL> DECLARE
v_due_date DATE := SYSDATE + 30;
BEGIN
    FOR loan_rec IN (...
Show more...

Reminder: Loan ID 103 for Customer ID 3 is due on 02-JUL-2025
Reminder: Loan ID 101 for Customer ID 1 is due on 07-JUL-2025

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.022

```

## 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.**

**CODE :**

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
```

```
BEGIN
```

```
    UPDATE Accounts
```

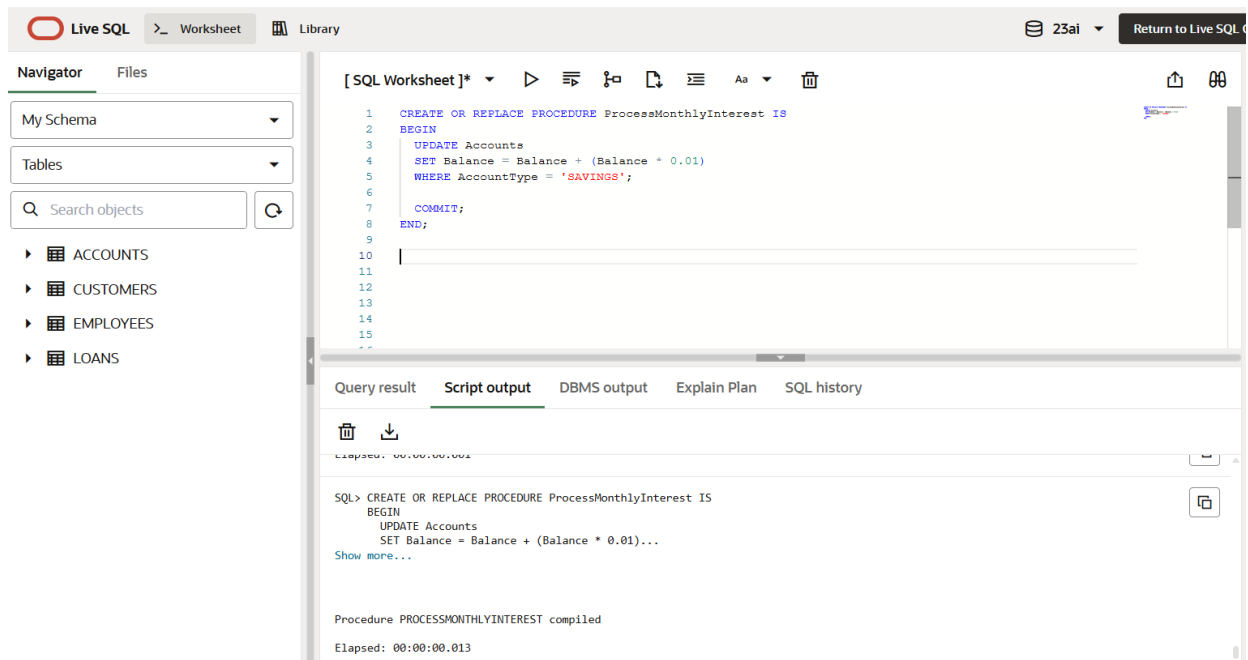
```
    SET Balance = Balance + (Balance * 0.01)
```

```
    WHERE AccountType = 'SAVINGS';
```

```
COMMIT;
```

```
END;
```

**OUTPUT :**



**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.**

**CODE :**

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (  
    p_DepartmentID IN NUMBER,  
    p_BonusPercent IN NUMBER  
) IS  
BEGIN  
    UPDATE Employees  
    SET Salary = Salary + (Salary * p_BonusPercent / 100)  
    WHERE DepartmentID = p_DepartmentID;  
    COMMIT;  
END;
```

**OUTPUT :**

The screenshot displays a web-based SQL editor interface. On the left, a 'Navigator' pane shows a schema named 'My Schema' with tables 'ACCOUNTS', 'CUSTOMERS', 'EMPLOYEES', and 'LOANS'. The main editor area contains the SQL code for the 'UpdateEmployeeBonus' procedure. Below the editor, the 'Script output' tab is active, showing the successful execution of the procedure. The output text reads: 'SQL> CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus ( p\_DepartmentID IN NUMBER, p\_BonusPercent IN NUMBER ) IS... Show more...' followed by 'Procedure UPDATEEMPLOYEEBONUS compiled' and 'Elapsed: 00:00:00.026'.

**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.**

**CODE :**

```
CREATE OR REPLACE PROCEDURE TransferFunds (  
    p_FromAccountID IN NUMBER,  
    p_ToAccountID   IN NUMBER,  
    p_Amount        IN NUMBER  
) IS  
    v_FromBalance NUMBER;  
BEGIN  
    -- Check balance of source account  
    SELECT Balance INTO v_FromBalance  
    FROM Accounts  
    WHERE AccountID = p_FromAccountID  
    FOR UPDATE;  
  
    IF v_FromBalance < p_Amount THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source  
account.');
```

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

END;

**OUTPUT :**

The screenshot displays the Live SQL web application interface. On the left, a 'Navigator' pane shows a schema named 'My Schema' with tables 'ACCOUNTS', 'CUSTOMERS', 'EMPLOYEES', and 'LOANS'. The main editor area, titled '[ SQL Worksheet ]\*', contains a PL/SQL procedure named 'TransferFunds'. The procedure has three parameters: 'p\_FromAccountID IN NUMBER', 'p\_ToAccountID IN NUMBER', and 'p\_Amount IN NUMBER'. The code includes a 'BEGIN' block with a comment '-- Check balance of source account', a 'SELECT' statement to fetch the balance into a variable 'v\_FromBalance', a 'WHERE' clause to lock the source account for update, and an 'IF' statement that raises an application error if the balance is insufficient. The 'Script output' tab at the bottom shows the successful execution of the procedure, indicating it was compiled and took 00:00:00.015 seconds to run.

```
1 CREATE OR REPLACE PROCEDURE TransferFunds (  
2   p_FromAccountID IN NUMBER,  
3   p_ToAccountID   IN NUMBER,  
4   p_Amount         IN NUMBER  
5 ) IS  
6   v_FromBalance NUMBER;  
7 BEGIN  
8   -- Check balance of source account  
9   SELECT Balance INTO v_FromBalance  
10  FROM Accounts  
11  WHERE AccountID = p_FromAccountID  
12  FOR UPDATE;  
13  
14  IF v_FromBalance < p_Amount THEN  
15    RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account.');
```

Query result Script output DBMS output Explain Plan SQL history

SQL> CREATE OR REPLACE PROCEDURE TransferFunds (  
 p\_FromAccountID IN NUMBER,  
 p\_ToAccountID IN NUMBER,  
 p\_Amount IN NUMBER...  
Show more...

Procedure TRANSFERFUNDS compiled  
Elapsed: 00:00:00.015