**NAME :** CHANDRU A **COLLEGE :** KONGU ENGINEERING COLLEGE

**PL/SQL\_EXERCISES**

# Exercise 1: Control Structures

## ****1. Understanding Control Structures in PL/SQL****

PL/SQL (Procedural Language/SQL) provides control flow mechanisms such as loops, conditional statements, and sequential execution. These structures are essential for handling logic, automating tasks, and processing data efficiently.

In a banking environment, control structures can be used to:

* Process customer data based on certain criteria (e.g., age, balance)
* Perform batch updates (e.g., adjust interest rates, update status)
* Schedule operations like sending reminders for upcoming due dates

**2. Table Setup**

Set Up Tables and Insert Sample Data

🡪Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER(10,2),

IsVIP CHAR(1)

);

🡪Loans Table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

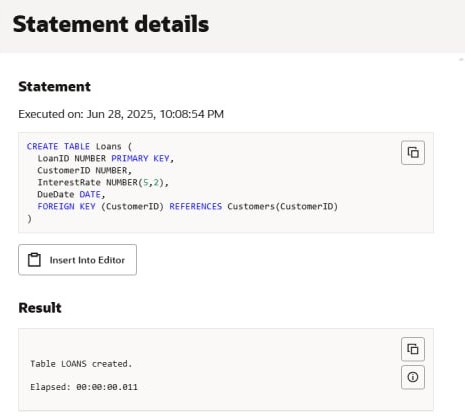
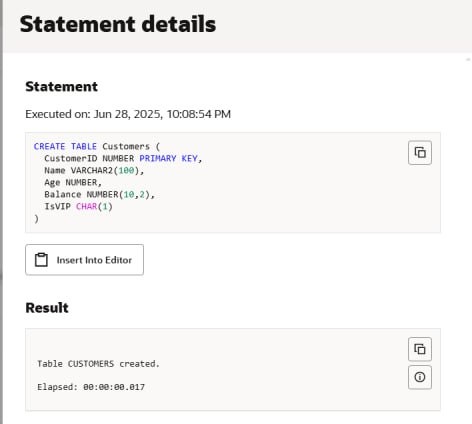
CustomerID NUMBER,

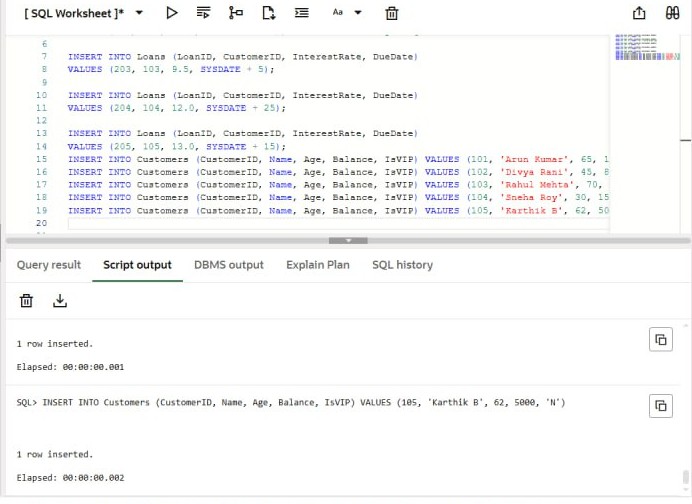
InterestRate NUMBER(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);





**3.Implementation**

**Scenario 1 – Apply 1% Discount to Interest Rates for Customers Aged Over 60**

BEGIN

FOR cust IN (SELECT CustomerID FROM Customers WHERE Age > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

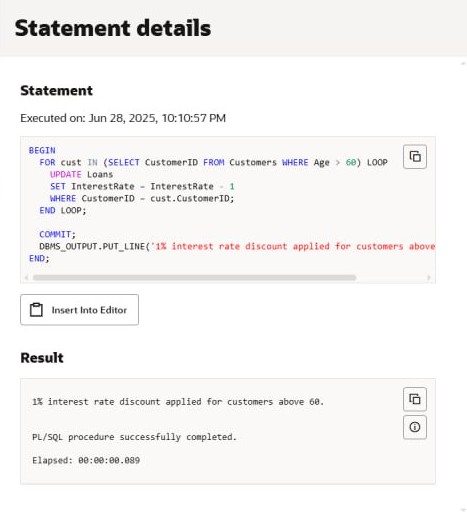
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('1% interest rate discount applied for customers above 60.');

END;

/

****

**Scenario 2 – Promote Customers to VIP Status Based on Balance**

BEGIN

FOR cust IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

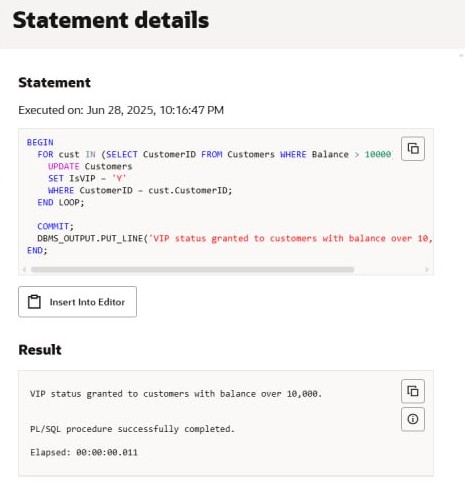
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('VIP status granted to customers with balance over 10,000.');

END;

/



**Scenario 3 – Print Loan Reminders for Loans Due in the Next 30 Days**

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.DueDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

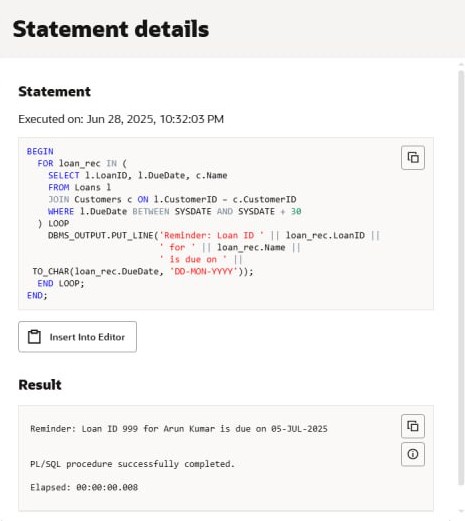
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' loan\_rec.LoanID

' for ' loan\_rec.Name

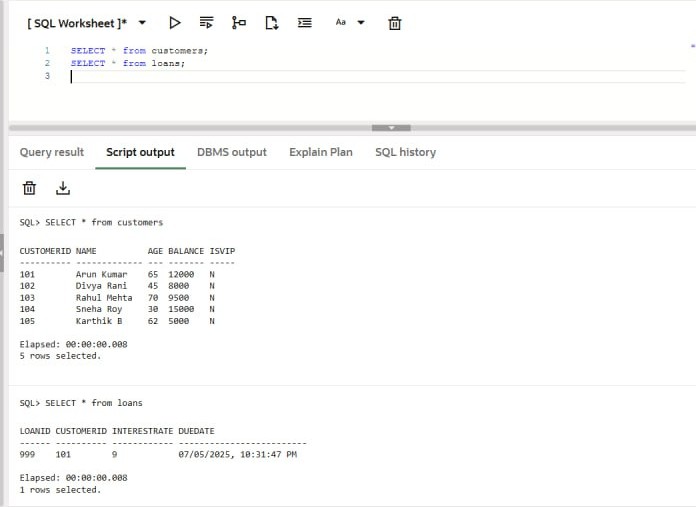
' is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Final Result:**



**Exercise 3: Stored Procedures**

## ****1. Understanding Stored Procedures in PL/SQL****

Stored procedures in PL/SQL are named, precompiled blocks of SQL and procedural statements that perform a specific task. They can be executed repeatedly and are beneficial in database applications to promote code reuse and secure transactional logic.

In a banking environment, stored procedures are used for:

* Automating tasks like interest crediting
* Implementing financial policies (e.g., employee bonus allocation)
* Managing fund transfers between accounts securely.

**2. Table Setup:**

Set Up Tables and Insert Sample Data

🡪Accounts Table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(30),

Balance NUMBER(10, 2)

);

🡪Employees Table

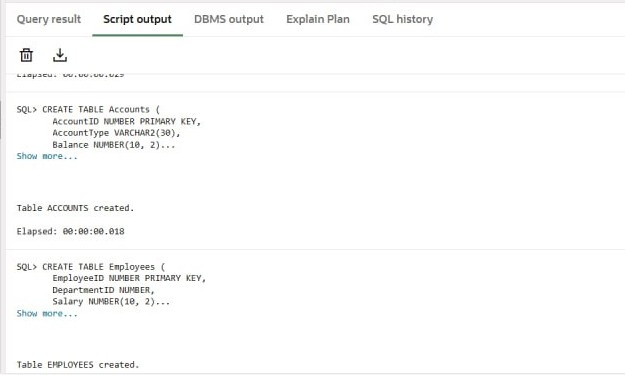
CREATE TABLE Employees (

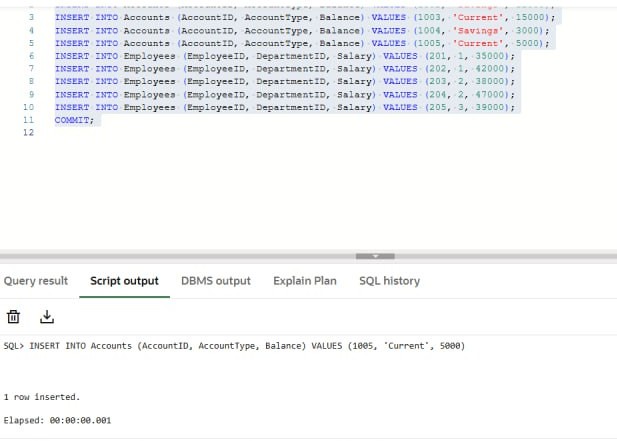
EmployeeID NUMBER PRIMARY KEY,

DepartmentID NUMBER,

Salary NUMBER(10, 2)

);





**3. Implementation:**

**Scenario 1: Credit Monthly Interest to Savings Accounts**

Step 1: Create Stored Procedure and Apply **1% interest** to all **savings accounts.**

CREATE OR REPLACE PROCEDURE CreditMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

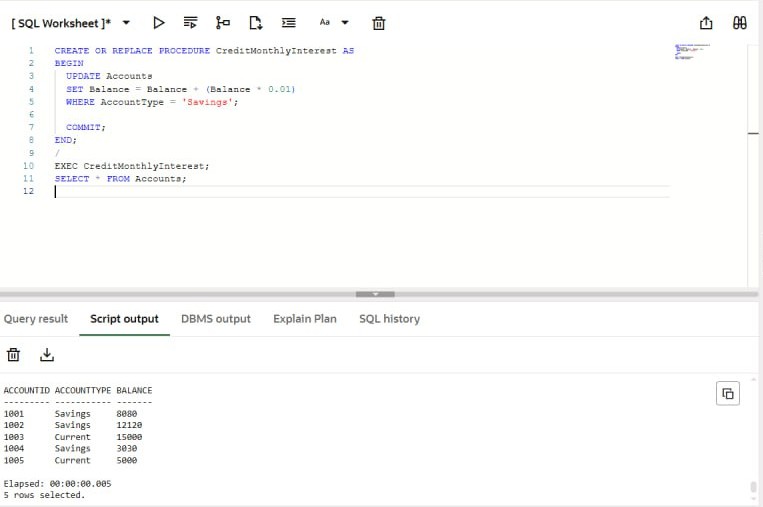
/

Step 2: Execute Procedure

EXEC CreditMonthlyInterest;

Step 3: Check Result

SELECT \* FROM Accounts;



## Scenario 2: Bonus Allocation Based on Department

Step 1: Create Stored Procedure and Update **salaries** by adding a **bonus percentage** for a specified **department**.

CREATE OR REPLACE PROCEDURE AllocateDepartmentBonus (

d\_id IN NUMBER,

bonus\_rate IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_rate / 100)

WHERE DepartmentID = d\_id;

COMMIT;

END;

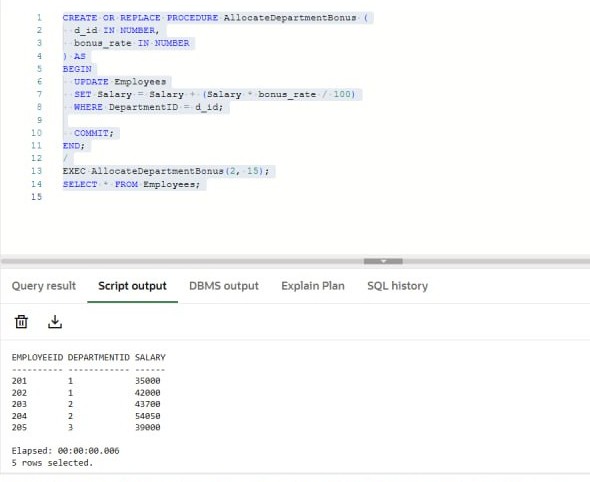
/

Step 2: Execute Procedure

EXEC AllocateDepartmentBonus(2, 15);

Step 3: Check Result

Select \* from Employees;



**Scenario 3: Account-to-Account Fund Transfer**

Transfer funds between two accounts **only if sufficient balance exists** in the source account.

Step 1: Create Stored Procedure

CREATE OR REPLACE PROCEDURE PerformFundTransfer (

sender\_acc IN NUMBER,

receiver\_acc IN NUMBER,

transfer\_amt IN NUMBER

) AS

e\_low\_balance EXCEPTION;

BEGIN

DECLARE

acc\_balance NUMBER;

BEGIN

SELECT Balance INTO acc\_balance FROM Accounts WHERE AccountID = sender\_acc;

IF acc\_balance < transfer\_amt THEN

RAISE e\_low\_balance;

END IF;

UPDATE Accounts

SET Balance = Balance - transfer\_amt

WHERE AccountID = sender\_acc;

UPDATE Accounts

SET Balance = Balance + transfer\_amt

WHERE AccountID = receiver\_acc;

COMMIT;

END;

EXCEPTION

WHEN e\_low\_balance THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred during fund transfer.');

END;

/

Step 2: Execute Fund Transfer

EXEC PerformFundTransfer(1001, 1002, 500);

Step 3: Check Balances

SELECT \* FROM Accounts;

