PL/SQL Programming - Control Structures

# Exercise 1: Control Structures

This document explains the PL/SQL control structure exercises in detail with code examples and outputs.

## Control Structures in PL/SQL

Control structures in PL/SQL control the flow of execution of the program. They allow the program to make decisions, repeat actions, and branch to different parts of the code based on certain conditions.

### Types of Control Structures in PL/SQL:

1. Conditional Control (Decision Making): Allows executing certain blocks of code based on conditions.  
Example:  
IF age > 60 THEN  
 DBMS\_OUTPUT.PUT\_LINE('Senior Citizen');  
END IF;

2. Iterative Control (Loops): Used to repeat a block of statements multiple times.  
Types include:  
- FOR LOOP  
- WHILE LOOP  
- SIMPLE LOOP  
Example:  
FOR i IN 1..5 LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Iteration: ' || i);  
END LOOP;

3. Sequential Control (Branching): Controls the logical flow to jump from one part of the program to another.  
Example:  
GOTO label\_name;  
<<label\_name>>  
DBMS\_OUTPUT.PUT\_LINE('Jumped here!');

### Importance of Control Structures:

- Makes the program dynamic and flexible  
- Helps in decision-making  
- Allows iteration through database records  
- Essential for business rule implementation

## Scenario 1: Apply Discount for Senior Citizens

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.

### PL/SQL Code:

DECLARE  
 CURSOR customer\_cursor IS  
 SELECT CustomerID, Age, LoanInterestRate  
 FROM BankCustomers;  
BEGIN  
 FOR customer\_record IN customer\_cursor LOOP  
 IF customer\_record.Age > 60 THEN  
 UPDATE BankCustomers  
 SET LoanInterestRate = LoanInterestRate - 1  
 WHERE CustomerID = customer\_record.CustomerID;  
 DBMS\_OUTPUT.PUT\_LINE('1% interest discount applied for Customer ID: ' || customer\_record.CustomerID);  
 END IF;  
 END LOOP;  
 COMMIT;  
END;  
/

### Sample Output:

1% interest discount applied for Customer ID: 101  
1% interest discount applied for Customer ID: 105

## Scenario 2: Promote Customers to VIP Based on 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.

### PL/SQL Code:

DECLARE  
 CURSOR vip\_cursor IS  
 SELECT CustomerID, Balance  
 FROM BankCustomers;  
BEGIN  
 FOR vip\_record IN vip\_cursor LOOP  
 IF vip\_record.Balance > 10000 THEN  
 UPDATE BankCustomers  
 SET IsVIP = 'TRUE'  
 WHERE CustomerID = vip\_record.CustomerID;  
 DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || vip\_record.CustomerID || ' promoted to VIP.');  
 END IF;  
 END LOOP;  
 COMMIT;  
END;  
/

### Sample Output:

Customer ID 102 promoted to VIP.  
Customer ID 108 promoted to VIP.

## Scenario 3: Send Loan Due Reminder

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

### PL/SQL Code:

DECLARE  
 CURSOR loan\_cursor IS  
 SELECT LoanID, CustomerID, DueDate  
 FROM Loans  
 WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30;  
BEGIN  
 FOR loan\_record IN loan\_cursor LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_record.LoanID || ' for Customer ID ' || loan\_record.CustomerID || ' is due on ' || TO\_CHAR(loan\_record.DueDate, 'DD-MON-YYYY'));  
 END LOOP;  
END;  
/

### Sample Output:

Reminder: Loan ID L201 for Customer ID 103 is due on 05-JUL-2025  
Reminder: Loan ID L205 for Customer ID 110 is due on 20-JUL-2025

PL/SQL Exercise 3: Stored Procedures

# Introduction to Stored Procedures

A stored procedure in PL/SQL is a reusable, named block of SQL and PL/SQL statements stored in the Oracle Database. It can take input parameters, process data, and perform specific tasks like updating records or generating reports. Stored procedures improve performance, security, and ease of maintenance.

# Scenario 1: Process Monthly Interest for Savings Accounts

## Problem:

The bank wants to apply a 1% monthly interest rate to all savings accounts.

## Table Structure:

CREATE TABLE savings\_account (  
 account\_id NUMBER PRIMARY KEY,  
 account\_holder VARCHAR2(50),  
 balance NUMBER  
);

## Stored Procedure:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 UPDATE savings\_account  
 SET balance = balance + (balance \* 0.01);  
   
 DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');  
END;  
/

## Execution:

BEGIN  
 ProcessMonthlyInterest;  
END;  
/

## Expected Output:

Monthly interest applied to all savings accounts.

# Scenario 2: Employee Bonus Based on Performance

## Problem:

The bank wants to add a bonus percentage to all employees' salaries within a specific department.

## Table Structure:

CREATE TABLE employee (  
 emp\_id NUMBER PRIMARY KEY,  
 emp\_name VARCHAR2(50),  
 department VARCHAR2(50),  
 salary NUMBER  
);

## Stored Procedure:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(  
 dept\_name IN VARCHAR2,  
 bonus\_percent IN NUMBER  
) IS  
BEGIN  
 UPDATE employee  
 SET salary = salary + (salary \* bonus\_percent / 100)  
 WHERE department = dept\_name;  
  
 DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || dept\_name);  
END;  
/

## Execution:

BEGIN  
 UpdateEmployeeBonus('Sales', 10);  
END;  
/

## Expected Output:

Bonus applied to department: Sales

# Scenario 3: Funds Transfer Between Accounts

## Problem:

The bank wants customers to transfer funds between accounts, ensuring sufficient balance exists in the source account.

## Table Structure:

CREATE TABLE bank\_account (  
 account\_id NUMBER PRIMARY KEY,  
 account\_holder VARCHAR2(50),  
 balance NUMBER  
);

## Stored Procedure:

CREATE OR REPLACE PROCEDURE TransferFunds(  
 from\_account IN NUMBER,  
 to\_account IN NUMBER,  
 amount IN NUMBER  
) IS  
 current\_balance NUMBER;  
BEGIN  
 SELECT balance INTO current\_balance FROM bank\_account WHERE account\_id = from\_account;  
  
 IF current\_balance < amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');  
 ELSE  
 UPDATE bank\_account  
 SET balance = balance - amount  
 WHERE account\_id = from\_account;  
  
 UPDATE bank\_account  
 SET balance = balance + amount  
 WHERE account\_id = to\_account;  
  
 DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || amount || ' transferred from account ' || from\_account || ' to account ' || to\_account);  
 END IF;  
END;  
/

## Execution:

BEGIN  
 TransferFunds(1, 2, 500);  
END;  
/

## Expected Output:

Transfer successful: 500 transferred from account 1 to account 2