**Week-II**

**PL SQL Exercises**

# Database Schema & Sample Data

Customers Table:

CREATE TABLE Customers (  
 CustomerID NUMBER PRIMARY KEY,  
 Name VARCHAR2(50),  
 Age NUMBER,  
 Balance NUMBER(10,2),  
 IsVIP VARCHAR2(5),  
 LoanInterestRate NUMBER(5,2)  
);

Accounts Table:

CREATE TABLE Accounts (  
 AccountID NUMBER PRIMARY KEY,  
 CustomerID NUMBER,  
 Balance NUMBER(10,2),  
 AccountType VARCHAR2(20),  
 FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);

Loans Table:

CREATE TABLE Loans (  
 LoanID NUMBER PRIMARY KEY,  
 CustomerID NUMBER,  
 DueDate DATE,  
 FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);

Employees Table:

CREATE TABLE Employees (  
 EmpID NUMBER PRIMARY KEY,  
 Name VARCHAR2(50),  
 Department VARCHAR2(30),  
 Salary NUMBER(10,2),  
 Performance VARCHAR2(20)  
);

Sample Data:

-- Customers  
INSERT INTO Customers VALUES (101, 'Alice', 65, 15000, 'FALSE', 10.5);  
INSERT INTO Customers VALUES (102, 'Bob', 45, 9500, 'FALSE', 9.8);  
INSERT INTO Customers VALUES (103, 'Charlie', 62, 12000, 'FALSE', 11.2);  
INSERT INTO Customers VALUES (104, 'Daisy', 30, 8000, 'FALSE', 10.0);  
  
-- Accounts  
INSERT INTO Accounts VALUES (1, 101, 5000, 'Savings');  
INSERT INTO Accounts VALUES (2, 102, 8000, 'Current');  
INSERT INTO Accounts VALUES (3, 101, 12000, 'Savings');  
INSERT INTO Accounts VALUES (4, 103, 3000, 'Savings');  
  
-- Loans  
INSERT INTO Loans VALUES (201, 101, SYSDATE + 10);  
INSERT INTO Loans VALUES (202, 102, SYSDATE + 35);  
INSERT INTO Loans VALUES (203, 103, SYSDATE + 5);  
INSERT INTO Loans VALUES (204, 104, SYSDATE + 40);  
  
-- Employees  
INSERT INTO Employees VALUES (1, 'John', 'HR', 50000, 'Excellent');  
INSERT INTO Employees VALUES (2, 'Alice', 'IT', 60000, 'Good');  
INSERT INTO Employees VALUES (3, 'Bob', 'HR', 55000, 'Average');  
INSERT INTO Employees VALUES (4, 'Mary', 'IT', 62000, 'Excellent');  
  
COMMIT;

**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.

**Code**:

begin

for cust in (select CustomerID, Age, LoanInterestRate from Customers) loop

if cust.Age > 60 then

update Customers

set LoanInterestRate = LoanInterestRate - 1

where CustomerID = cust.CustomerID;

end if;

end loop;

commit;

end;

/

**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 in (select CustomerID, Balance from Customers) loop

if cust.Balance > 10000 then

update Customers

set IsVIP = 'TRUE'

where CustomerID = cust.CustomerID;

dbms\_output.put\_line('CustomerID: ' || cust.CustomerID ||

' promoted to VIP. Balance: ' || cust.Balance);

end if;

end loop;

commit;

end;

/

**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:**

begin

for loan\_rec in (

select L.LoanID, L.CustomerID, L.DueDate, C.Name

from Loans L

join Customers C on L.CustomerID = C.CustomerID

where L.DueDate <= sysdate + 30

) loop

dbms\_output.put\_line(

'Reminder: Loan ID ' || loan\_rec.LoanID ||

' for customer ' || loan\_rec.Name ||

' (CustomerID: ' || loan\_rec.CustomerID ||

') is due on ' || to\_char(loan\_rec.DueDate, 'DD-MON-YYYY')

);

end loop;

end;

/

**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.

**Code:**

create or replace procedure ProcessMonthlyInterest is

begin

for acc in (

select AccountID, Balance

from Accounts

where AccountType = 'Savings'

) loop

update Accounts

set Balance = acc.Balance + (acc.Balance \* 0.01)

where AccountID = acc.AccountID;

dbms\_output.put\_line('Interest applied to Account ID ' || acc.AccountID ||

': New Balance = ' || to\_char(acc.Balance + (acc.Balance \* 0.01), '99999.99'));

end loop;

commit;

end;

/

**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\_dept in varchar2,

p\_bonus in number -- Pass as percentage like 10 for 10%

) is

begin

for emp\_rec in (

select EmpID, Salary

from Employees

where Department = p\_dept

) loop

update Employees

set Salary = emp\_rec.Salary + (emp\_rec.Salary \* p\_bonus / 100)

where EmpID = emp\_rec.EmpID;

dbms\_output.put\_line('Bonus applied to Employee ID ' || emp\_rec.EmpID ||

': New Salary = ' || to\_char(emp\_rec.Salary + (emp\_rec.Salary \* p\_bonus / 100), '999999.99'));

end loop;

commit;

end;

/

**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\_from\_acct in number,

p\_to\_acct in number,

p\_amount in number

) is

v\_from\_balance Accounts.Balance%type;

begin

-- Step 1: Get balance of source account

select Balance into v\_from\_balance

from Accounts

where AccountID = p\_from\_acct;

-- Step 2: Check sufficient balance

if v\_from\_balance < p\_amount then

dbms\_output.put\_line('Insufficient balance in Account ID ' || p\_from\_acct);

return;

end if;

-- Step 3: Deduct from source

update Accounts

set Balance = Balance - p\_amount

where AccountID = p\_from\_acct;

-- Step 4: Add to destination

update Accounts

set Balance = Balance + p\_amount

where AccountID = p\_to\_acct;

dbms\_output.put\_line('Transferred ' || p\_amount || ' from Account ID ' ||

p\_from\_acct || ' to Account ID ' || p\_to\_acct);

commit;

end;

/