# PL/SQL EXERCISES

### Schema to be Created:

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

### INSERTED VALUE:

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Elder Thomas', TO\_DATE('1950-04-10', 'YYYY-MM-DD'), 2000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Grace Walker', TO\_DATE('1958-09-25', 'YYYY-MM-DD'), 1800, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 3, 15000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

# 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.

**--Data insertion**

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 4, 20000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

SET SERVEROUTPUT ON;

DECLARE

v\_age NUMBER;

v\_old\_rate NUMBER;

v\_new\_rate NUMBER;

BEGIN

FOR cust\_rec IN (

SELECT c.CustomerID, c.Name, c.DOB, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

) LOOP

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

IF v\_age > 60 THEN

v\_old\_rate := cust\_rec.InterestRate;

v\_new\_rate := v\_old\_rate - 1;

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = cust\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || cust\_rec.Name ||

' | Age: ' || v\_age ||

' | Old Interest Rate: ' || v\_old\_rate ||

' | New Interest Rate: ' || v\_new\_rate);

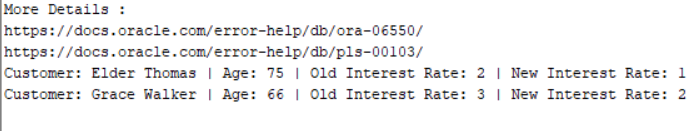
END IF;

END LOOP;

COMMIT;

END;

### OUTPUT:



### 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.

**--New column added**

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

**--Data insertion**

INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

VALUES (5, 'George', TO\_DATE('1993-07-16', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

VALUES (6, 'Martin', TO\_DATE('1998-08-19', 'YYYY-MM-DD'), 60000, SYSDATE);

SET SERVEROUTPUT ON;

BEGIN

FOR cust\_rec IN (SELECT CustomerID, Name, Balance FROM Customers) LOOP

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || cust\_rec.Name ||

' | Balance: $' || cust\_rec.Balance ||

' | ==>: You are eligle for VIP');

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || cust\_rec.Name ||

' | Balance: $' || cust\_rec.Balance ||

' | ==>: You are not eligle for VIP');

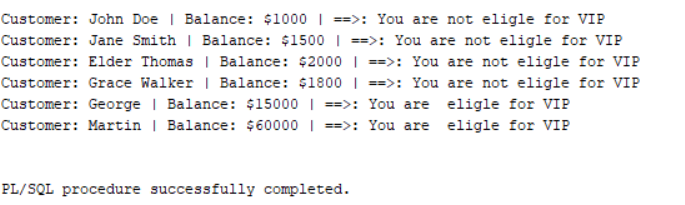
END IF;

END LOOP;

COMMIT;

END;

### OUTPUT:



### 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.

--Data insertion:

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (4, 1, 3000, 6, SYSDATE, SYSDATE + 10);

SET SERVEROUTPUT ON;

BEGIN

FOR loan\_rec IN (

SELECT c.Name, c.CustomerID, l.LoanID, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

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

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

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

END LOOP;

END;

### OUTPUT:



# 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.

**Function:**

CREATE FUNCTION CalculateInterest(p\_balance NUMBER) RETURN NUMBER IS

BEGIN

RETURN p\_balance + (p\_balance \* 0.01); -- 1% interest

END;

**Stored procedure:**

CREATE PROCEDURE ProcessMonthlyInterest IS

v\_old\_balance NUMBER;

v\_new\_balance NUMBER;

BEGIN

FOR acc\_rec IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

v\_old\_balance := acc\_rec.Balance;

v\_new\_balance := CalculateInterest(v\_old\_balance);

UPDATE Accounts

SET Balance = v\_new\_balance,

LastModified = SYSDATE

WHERE AccountID = acc\_rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || acc\_rec.AccountID ||

' | Old Balance: ' || v\_old\_balance ||

' | New Balance: ' || v\_new\_balance);

END LOOP;

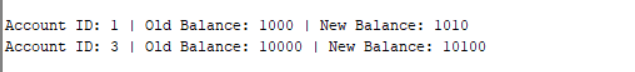
COMMIT;

END;

SET SERVEROUTPUT ON;

EXEC ProcessMonthlyInterest;

### 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.

**Stored procedure:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

v\_old\_salary NUMBER;

v\_new\_salary NUMBER;

BEGIN

FOR emp\_rec IN (

SELECT EmployeeID, Name, Salary

FROM Employees

WHERE Department = p\_department

) LOOP

v\_old\_salary := emp\_rec.Salary;

v\_new\_salary := v\_old\_salary + (v\_old\_salary \* p\_bonus\_percent / 100);

UPDATE Employees

SET Salary = v\_new\_salary

WHERE EmployeeID = emp\_rec.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Employee: ' || emp\_rec.Name ||

' | Old Salary: ' || v\_old\_salary ||

' | New Salary: ' || v\_new\_salary ||

' | Department: ' || p\_department);

END LOOP;

COMMIT;

END;

**Department and bonus passed as parameter:**

SET SERVEROUTPUT ON;

EXEC UpdateEmployeeBonus('IT', 10);

### OUTPUT:

### 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.

**Stored Procedure:**

CREATE PROCEDURE TransferFunds (

p\_source\_account IN NUMBER,

p\_destination\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_source\_balance NUMBER;

v\_dest\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account;

IF v\_source\_balance < p\_amount THEN

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

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_source\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_destination\_account;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ₹' || p\_amount ||

' transferred from Account ' || p\_source\_account ||

' to Account ' || p\_destination\_account);

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: One or both account IDs not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

ROLLBACK;

END;

SET SERVEROUTPUT ON;

EXEC TransferFunds(1, 2, 500);

### OUTPUT:

