Week 2- PLSQL_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
);
```

Example Scripts for Sample Data Insertion

```
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'));

Exercise 1: Control Structures

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

o 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.

Scenario 2: A customer can be promoted to VIP status based on their balance.

o 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.

Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.

o 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:

```
-- Scenario 1: Applying a Discount to Loan Interest Rates for Customers Above 60 Years Old

DECLARE

   CURSOR c_customers IS

      SELECT customer_id, loan_interest_rate, age

      FROM customers

      JOIN loans ON customers.customer_id = loans.customer_id

      WHERE age > 60;

   v_new_interest_rate NUMBER;

BEGIN

   FOR rec IN c_customers LOOP

      v_new_interest_rate := rec.loan_interest_rate - 0.01; -- Applying 1% discount

      UPDATE loans

      SET loan_interest_rate = v_new_interest_rate

      WHERE customer_id = rec.customer_id;

   END LOOP;
```

```
    COMMIT; -- Commit changes to the database
    DBMS_OUTPUT.PUT_LINE('Discounts applied to eligible customers.');
END;
/
-- Scenario 2: Promoting Customers to VIP Status Based on Balance
DECLARE
    CURSOR c_customers IS
        SELECT customer_id, balance
        FROM customers
        WHERE balance > 10000;
BEGIN
    FOR rec IN c_customers LOOP
        UPDATE customers
        SET IsVIP = TRUE
        WHERE customer_id = rec.customer_id;
    END LOOP;

    COMMIT; -- Commit changes to the database
    DBMS_OUTPUT.PUT_LINE('VIP status updated for eligible customers.');
END;
/


-- Scenario 3: Sending Reminders for Loans Due Within the Next 30 Days
DECLARE
    CURSOR c_loans IS
        SELECT customer_id, loan_due_date
        FROM loans
        WHERE loan_due_date BETWEEN SYSDATE AND SYSDATE + 30;
BEGIN
    FOR rec IN c_loans LOOP
```

```
    -- Assuming you have a procedure or function to send reminders
    -- Example: send_reminder(rec.customer_id, rec.loan_due_date);
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan due on ' || rec.loan_due_date || ' for customer ID: '
|| rec.customer_id);
  END LOOP;
END;
/
```

Exercise 3: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

o 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.

Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

o 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.

Scenario 3: Customers should be able to transfer funds between their accounts.

o 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:

```
-- Scenario 1: ProcessMonthlyInterest Procedure
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  -- Update the balance of all savings accounts
  UPDATE accounts
  SET balance = balance * 1.01
  WHERE account_type = 'Savings';

  -- Commit the transaction
  COMMIT;
```

```
    DBMS_OUTPUT.PUT_LINE('Monthly interest applied to all savings accounts.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error processing monthly interest: ' || SQLERRM);
END ProcessMonthlyInterest;
/

-- Scenario 2: UpdateEmployeeBonus Procedure
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
    p_department_id IN NUMBER,
    p_bonus_percentage IN NUMBER
) IS
BEGIN
    -- Update the salary with the bonus percentage
    UPDATE employees
    SET salary = salary * (1 + p_bonus_percentage / 100)
    WHERE department_id = p_department_id;

    -- Check if any rows were updated
    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No employees found for the given department ID.');
    ELSE
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Bonus updated for employees in department ID: ' ||
p_department_id);
    END IF;

EXCEPTION
    WHEN OTHERS THEN
```

```
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error updating employee bonus: ' || SQLERRM);
END UpdateEmployeeBonus;
/

-- Scenario 3: TransferFunds Procedure
CREATE OR REPLACE PROCEDURE TransferFunds(
    p_from_account IN NUMBER,
    p_to_account IN NUMBER,
    p_amount IN NUMBER
) AS
    v_balance NUMBER;
BEGIN
    -- Start a transaction
    SAVEPOINT before_transfer;

    -- Check if the source account has enough balance
    SELECT balance INTO v_balance
    FROM accounts
    WHERE account_id = p_from_account;

    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account.');
    END IF;

    -- Deduct funds from the source account
    UPDATE accounts
    SET balance = balance - p_amount
    WHERE account_id = p_from_account;

    -- Add funds to the destination account
```

```
    UPDATE accounts
    SET balance = balance + p_amount
    WHERE account_id = p_to_account;

    -- Commit the transaction
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Funds transferred successfully.');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        ROLLBACK TO SAVEPOINT before_transfer;
        DBMS_OUTPUT.PUT_LINE('Error: One or both accounts do not exist.');
    WHEN OTHERS THEN
        ROLLBACK TO SAVEPOINT before_transfer;
        DBMS_OUTPUT.PUT_LINE('Error during fund transfer: ' || SQLERRM);
END TransferFunds;
/
```

Table Output:

	ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE	LASTMODIFIED
1	2	2	Checking	1500	6/29/2025, 2:51:21
2	3	3	Savings	8000	6/29/2025, 3:18:42
3	4	4	Savings	20000	6/29/2025, 3:18:42
4	5	5	Current	5000	6/29/2025, 3:18:42
5	1	1	Savings	1000	6/29/2025, 2:51:14

Download Execution time: 0.002 seconds

	CUSTOMERID	NAME	DOB	BALANCE	LASTMODIFIED
1	1	John Doe	5/15/1985, 12:00:0(	1000	6/29/2025, 2:47:20
2	3	Senior Citizen	1/1/1950, 12:00:00	8000	6/29/2025, 3:17:05
3	4	Wealthy Patron	8/20/1975, 12:00:0(	20000	6/29/2025, 3:18:42
4	5	Loan Due Soon	11/11/1980, 12:00:(	5000	6/29/2025, 3:18:42
5	2	Jane Smith	7/20/1990, 12:00:0(	1500	6/29/2025, 2:49:26

	EMPLOYEEID	NAME	POSITION	SALARY	DEPARTMENT	HIREDATE
1	3	Bob Brown	Developer	60000	IT	3/20/2017, 12:00:0
2	4	Dana Lee	Analyst	55000	Marketing	8/5/2018, 12:00:00
3	2	Bob Brown	Developer	60000	IT	3/20/2017, 12:00:0
4	1	Alice Johnson	Manager	70000	HR	6/15/2015, 12:00:0

Download Execution time: 0.003 seconds

	LOANID	CUSTOMERID	LOANAMOUNT	INTERESTRATE	STARTDATE	ENDDATE
1	2	2	10000	7	6/29/2025, 2:55:53	7/14/2025, 2:55:53
2	3	4	15000	5.5	6/29/2025, 3:18:42	6/29/2028, 3:18:42
3	4	5	12000	4.5	6/29/2025, 3:18:42	7/14/2025, 3:18:42
4	1	1	5000	5	6/29/2025, 2:51:49	6/29/2030, 2:51:49

Download Execution time: 0.002 seconds

	TRANSACTIONID	ACCOUNTID	TRANSACTIONDATE	AMOUNT	TRANSACTIONTYPE
1	1	1	6/29/2025, 2:51:29	200	Deposit
2	2	2	6/29/2025, 2:51:39	300	Withdrawal
3	3	3	6/29/2025, 3:18:42	1000	Deposit
4	4	4	6/29/2025, 3:18:42	1500	Withdrawal
5	5	5	6/29/2025, 3:18:42	2000	Deposit