**MODULE – 1 (PL/SQL programming)**

**Exercise 1: Control Structures**

**DATABASE:**

|  |  |
| --- | --- |
| **CUSTOMERS:**  CREATE TABLE Customers (      CustomerID   NUMBER PRIMARY KEY,      CustomerName VARCHAR2(100),      Age          NUMBER,      Balance      NUMBER,      IsVIP        VARCHAR2(5)  ); | **LOANS:**  CREATE TABLE Loans (      LoanID       NUMBER PRIMARY KEY,      CustomerID   NUMBER,      InterestRate NUMBER,      DueDate      DATE,      FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  ); |

**INSERTING VALUES:**

|  |  |
| --- | --- |
| INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'FALSE');  INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'FALSE');  INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'FALSE');  INSERT INTO Customers VALUES (4, 'Diana', 30, 10500, 'FALSE') | INSERT INTO Loans VALUES (101, 1, 10.5, SYSDATE + 15);  INSERT INTO Loans VALUES (102, 2, 11.0, SYSDATE + 45);  INSERT INTO Loans VALUES (103, 3, 9.5, SYSDATE + 10);  INSERT INTO Loans VALUES (104, 4, 12.0, SYSDATE + 5); |

**OUTPUT (CUSTOMERS): A screenshot of a computer

AI-generated content may be incorrect.**

**OUTPUT(LOANS):**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**SOLUTION:**

BEGIN

   FOR customer\_rec IN (

      SELECT c.CustomerID, c.Age, l.LoanID, l.InterestRate

      FROM Customers c

      JOIN Loans l ON c.CustomerID = l.CustomerID

   )

   LOOP

      IF customer\_rec.Age > 60 THEN

         UPDATE Loans

         SET InterestRate = InterestRate - 1

         WHERE LoanID = customer\_rec.LoanID;

      END IF;

   END LOOP;

   COMMIT;

END;

/

**OUTPUT: A white background with black text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**SOLUTION:**

BEGIN

   FOR cust IN (SELECT CustomerID, Balance FROM Customers)

   LOOP

      IF cust.Balance > 10000 THEN

         UPDATE Customers

         SET IsVIP = 'TRUE'

         WHERE CustomerID = cust.CustomerID;

      END IF;

   END LOOP;

   COMMIT;

END;

/

**OUTPUT:**

**A white background with black text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**SOLUTION:**

BEGIN

   FOR loan\_rec IN (

      SELECT l.LoanID, c.CustomerName, l.DueDate

      FROM Loans l

      JOIN Customers c ON l.CustomerID = c.CustomerID

      WHERE l.DueDate <= SYSDATE + 30

   )

   LOOP

      DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.CustomerName ||

                           ', your loan (ID: ' || loan\_rec.LoanID ||

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

   END LOOP;

END;

/

**OUTPUT:**

A screenshot of a computer error

AI-generated content may be incorrect.

**Exercise 3: Stored Procedures**

**DATABASE:**

|  |  |
| --- | --- |
| **ACCOUNTS:**  CREATE TABLE Accounts (      AccountID   NUMBER PRIMARY KEY,      CustomerName VARCHAR2(100),      Balance      NUMBER,      AccountType  VARCHAR2(20)  ); | **EMPLOYEES:**  CREATE TABLE Employees (      EmpID      NUMBER PRIMARY KEY,      Name       VARCHAR2(100),      Department VARCHAR2(50),      Salary     NUMBER  ); |

**INSERTING VALUES:**

|  |  |
| --- | --- |
| INSERT INTO Accounts VALUES (101, 'Alice', 10000, 'Savings');  INSERT INTO Accounts VALUES (102, 'Bob', 5000, 'Savings');  INSERT INTO Accounts VALUES (103, 'Charlie', 7000, 'Current'); | INSERT INTO Employees VALUES (1, 'John', 'Sales', 50000);  INSERT INTO Employees VALUES (2, 'Jane', 'HR', 55000);  INSERT INTO Employees VALUES (3, 'Mark', 'Sales', 60000); |

**OUTPUT (ACCOUNTS):**

**A screenshot of a computer

AI-generated content may be incorrect.**

**OUTPUT (EMPLOYEES):**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

/

**OUTPUT:**

A white background with black text

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

   dept\_name IN VARCHAR2,

   bonus\_pct IN NUMBER

) IS

BEGIN

   UPDATE Employees

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

   WHERE Department = dept\_name;

   COMMIT;

END;

/

**OUTPUT:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**SOLUTION:**

CREATE OR REPLACE PROCEDURE TransferFunds (

   from\_acc IN NUMBER,

   to\_acc   IN NUMBER,

   amt      IN NUMBER

) IS

   from\_balance NUMBER;

BEGIN

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

   IF from\_balance < amt THEN

      RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

   END IF;

   UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = from\_acc;

   UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = to\_acc;

   COMMIT;

END;

/

**OUTPUT:**

**A white screen with black text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**