**TABLE CREATION:**

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

);

**DATA INSERTION:**

Customers

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

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

INSERT INTO Customers VALUES (3, 'Elder Tom', TO\_DATE('1950-03-10', 'YYYY-MM-DD'), 2000, SYSDATE);

INSERT INTO Customers VALUES (4, 'Richie Rich', TO\_DATE('1988-01-01', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Customers VALUES (5, 'Paul Wilson', TO\_DATE('1992-11-23', 'YYYY-MM-DD'), 500, SYSDATE,'TRUE');

INSERT INTO Customers VALUES (6, 'Rita Kumar', TO\_DATE('1980-08-08', 'YYYY-MM-DD'), 8000, SYSDATE,'FALSE');

INSERT INTO CUSTOMERS VALUES (7, 'Walter Grant', TO\_DATE('1940-12-12', 'YYYY-MM-DD'), 12000, SYSDATE, 'TRUE');

Accounts

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Accounts VALUES (3, 3, 'Savings', 2000, SYSDATE);

INSERT INTO Accounts VALUES (4, 4, 'Savings', 15000, SYSDATE);

INSERT INTO Accounts VALUES (5, 5, 'Checking', 500, SYSDATE);

INSERT INTO Accounts VALUES (6, 6, 'Savings', 8000, SYSDATE);

INSERT INTO Accounts VALUES (7, 7, 'Savings', 12000, SYSDATE);

Transactions

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

Loans

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

INSERT INTO Loans VALUES (2, 3, 3000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 6));

INSERT INTO Loans VALUES (3, 4, 10000, 4.5, SYSDATE, ADD\_MONTHS(SYSDATE, 12));

INSERT INTO Loans VALUES (4, 5, 1500, 7, SYSDATE, SYSDATE + 20);

INSERT INTO Loans VALUES (5, 6, 8000, 5.5, SYSDATE, ADD\_MONTHS(SYSDATE, 24));

INSERT INTO Loans VALUES (6, 7, 10000, 6, SYSDATE, SYSDATE + 10);

Employees

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

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

INSERT INTO Employees VALUES (3, 'Charles King', 'Tester', 50000, 'IT', TO\_DATE('2018-01-15', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (4, 'Debra Queen', 'Analyst', 65000, 'Finance', TO\_DATE('2016-09-09', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (5, 'Edward Prince', 'DevOps', 62000, 'IT', TO\_DATE('2019-12-12', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (6, 'Fiona Duchess', 'Support', 58000, 'HR', TO\_DATE('2020-03-01', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (7, 'George Baron', 'Engineer', 61000, 'IT', TO\_DATE('2021-05-05', '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.

**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 rec IN (

    SELECT l.LoanID, c.DOB

    FROM Customers c

    JOIN Loans l ON c.CustomerID = l.CustomerID

  ) LOOP

    IF TRUNC(MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12) > 60 THEN

      UPDATE Loans

      SET InterestRate = InterestRate - 1

      WHERE LoanID = rec.LoanID;

    END IF;

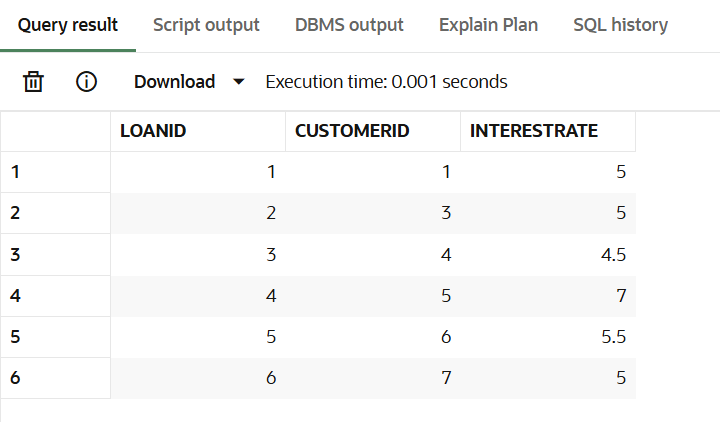
  END LOOP;

  COMMIT;

END;

SELECT LoanID, CustomerID, InterestRate FROM Loans;

**Output: Loans Table**

****

**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:**

ALTER TABLE Customers

ADD IsVIP VARCHAR2(5) DEFAULT 'FALSE' NOT NULL;

BEGIN

  FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

    IF rec.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = rec.CustomerID;

    END IF;

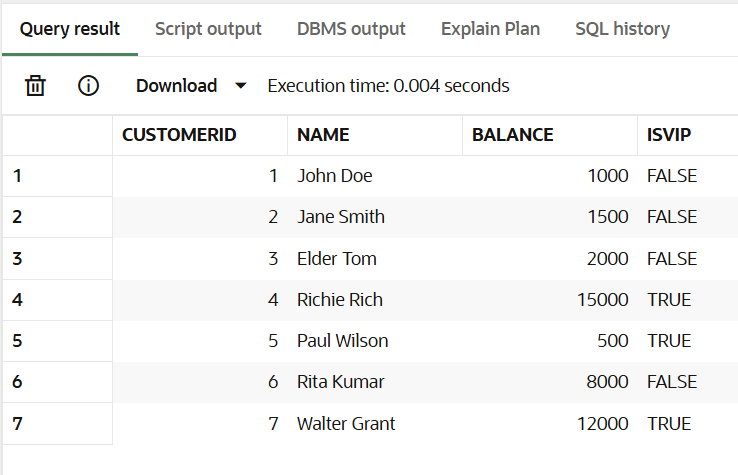
  END LOOP;

  COMMIT;

END;

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

**Output: Customers Table**

****

**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 rec IN (

    SELECT c.Name, l.EndDate

    FROM Loans l

    JOIN Customers c ON l.CustomerID = c.CustomerID

    WHERE l.EndDate <= SYSDATE + 30

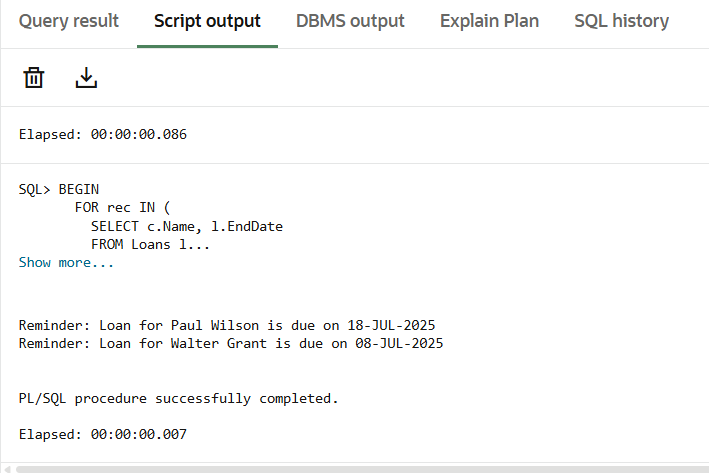
  ) LOOP

    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for ' || rec.Name || ' is due on ' || TO\_CHAR(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.

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

  -- Update all savings accounts with 1% interest

  UPDATE Accounts

  SET Balance = Balance + (Balance \* 0.01)

  WHERE AccountType = 'Savings';

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to savings accounts.');

END;

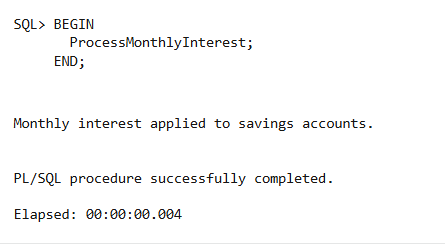
BEGIN

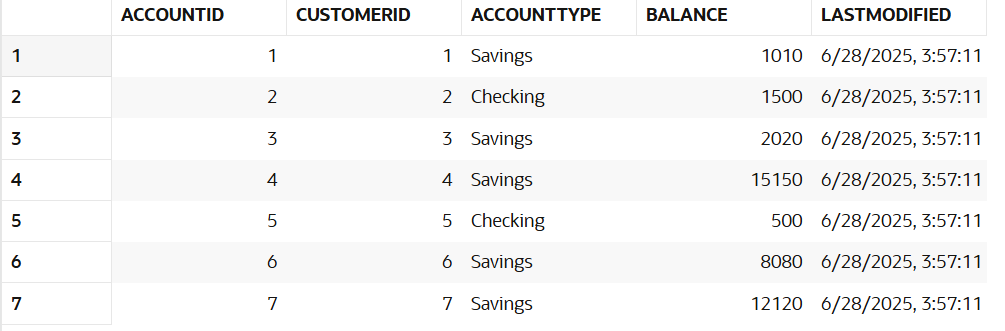
  ProcessMonthlyInterest;

END;

SELECT \* FROM Accounts;

**Output: Accounts Table**

****

****

**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\_department IN VARCHAR2,

  p\_bonus\_percent IN NUMBER

)

IS

BEGIN

  -- Apply bonus to all employees in the given department

  UPDATE Employees

  SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

  WHERE Department = p\_department;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || p\_department);

END;

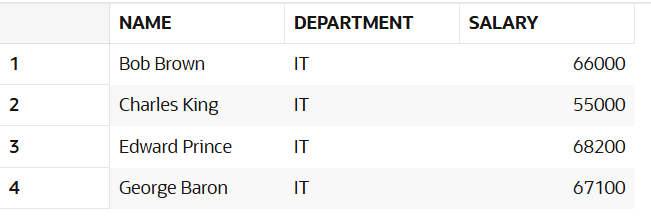
BEGIN

  UpdateEmployeeBonus('IT', 10);

END;

SELECT Name, Department, Salary FROM Employees WHERE Department = 'IT';

**Output: Employees Table**

****

**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\_fromAccountID IN NUMBER,

  p\_toAccountID   IN NUMBER,

  p\_amount        IN NUMBER

)

IS

  v\_balance NUMBER;

BEGIN

  -- Get balance of source account

  SELECT Balance INTO v\_balance

  FROM Accounts

  WHERE AccountID = p\_fromAccountID;

  IF v\_balance < p\_amount THEN

    DBMS\_OUTPUT.PUT\_LINE('Insufficient balance. Transfer cancelled.');

  ELSE

    -- Deduct from source

    UPDATE Accounts

    SET Balance = Balance - p\_amount

    WHERE AccountID = p\_fromAccountID;

    -- Add to destination

    UPDATE Accounts

    SET Balance = Balance + p\_amount

    WHERE AccountID = p\_toAccountID;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

  END IF;

EXCEPTION

  WHEN OTHERS THEN

    ROLLBACK;

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

END;

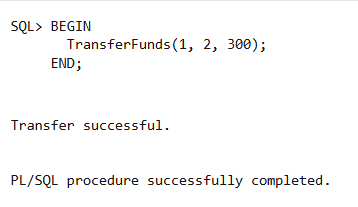
BEGIN

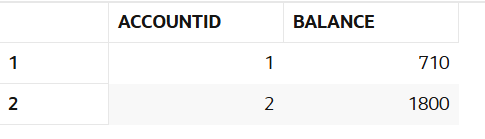
  TransferFunds(1, 2, 300);

END;

SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1, 2);

**Output: Accounts Table**

****

****