# EXERCISE – 1

-- Exercise 1

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

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', '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 Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

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'));

-- Enable server output

SET SERVEROUTPUT ON;

-- Adding IsVIP column to Customers table

ALTER TABLE Customers ADD IsVIP CHAR(1);

-- SCENARIO - 1

-- Procedure to Apply Interest Discount to Customers Above 60 Years Old

CREATE OR REPLACE PROCEDURE ApplyInterestDiscount IS

    CURSOR cur IS

        SELECT CustomerID, DOB FROM Customers;

    cur\_CustomerID Customers.CustomerID%TYPE;

    cur\_DOB Customers.DOB%TYPE;

    currentDATE DATE := SYSDATE;

BEGIN

    FOR customer\_rec IN cur LOOP

        IF MONTHS\_BETWEEN(currentDATE, customer\_rec.DOB) / 12 > 60 THEN

            UPDATE Loans

            SET InterestRate = InterestRate \* 0.99

            WHERE CustomerID = customer\_rec.CustomerID;

            DBMS\_OUTPUT.PUT\_LINE('CustomerID: '||customer\_rec.CustomerID||' interest rate has been decreased by 1%.');

        END IF;

    END LOOP;

END ApplyInterestDiscount;

/

-- SCENARIO - 2

-- Procedure to Promote Customers to VIP Status Based on Balance

CREATE OR REPLACE PROCEDURE PromoteToVIP IS

    CURSOR cur IS

        SELECT CustomerID, Balance FROM Accounts;

    cur\_CustomerID Accounts.CustomerID%TYPE;

    cur\_Balance Accounts.Balance%TYPE;

BEGIN

    FOR account\_record IN cur LOOP

        IF account\_record.Balance > 5000 THEN

            UPDATE Customers

            SET IsVIP = 'Y'

            WHERE CustomerID = account\_record.CustomerID;

            DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || account\_record.CustomerID|| 'Promoted to VIP!!');

        ELSE

            UPDATE Customers

            SET IsVIP = 'N'

            WHERE CustomerID = account\_record.CustomerID;

            DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || account\_record.CustomerID|| 'Demoted from VIP!!');

        END IF;

    END LOOP;

END PromoteToVIP;

/

-- SCENARIO - 3

-- Procedure to Send Loan Reminders for Loans Due Within the Next 30 Days

CREATE OR REPLACE PROCEDURE SendLoanReminders IS

    CURSOR cur IS

        SELECT CustomerID, EndDate

        FROM Loans

        WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

    cur\_CustomerID Loans.CustomerID%TYPE;

    cur\_EndDate Loans.EndDate%TYPE;

BEGIN

    FOR loan\_rec IN cur LOOP

        DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || ' for CustomerID: ' || loan\_rec.CustomerID);

    END LOOP;

END SendLoanReminders;

/

-- Call the procedures

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('SCENARIO - 1: Apply discount to interest rate for the customers above age 60');

    ApplyInterestDiscount; -- Apply discount to interest rate for the customers above age 60

END;

/

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('SCENARIO - 2: Promote customers to VIP status based on balance');

    PromoteToVIP; -- Promote customers to VIP status based on balance

END;

/

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('SCENARIO - 3: Reminders for loans due within the next 30 days');

    SendLoanReminders; -- Send Loan Reminders for loans due within the next 30 days

END;

/

# EXERCISE – 2

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Insert data

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'));

-- Procedure to Apply Interest Discount to Customers Above 60 Years Old

CREATE OR REPLACE PROCEDURE ApplyInterestDiscount IS

    CURSOR cur IS

        SELECT CustomerID, DOB FROM Customers;

    cur\_CustomerID Customers.CustomerID%TYPE;

    cur\_DOB Customers.DOB%TYPE;

    currentDATE DATE := SYSDATE;

BEGIN

    FOR customer\_rec IN cur LOOP

        IF MONTHS\_BETWEEN(currentDATE, customer\_rec.DOB) / 12 > 60 THEN

            UPDATE Loans

            SET InterestRate = InterestRate \* 0.99

            WHERE CustomerID = customer\_rec.CustomerID;

        END IF;

    END LOOP;

END ApplyInterestDiscount;

/

-- Procedure to Promote Customers to VIP Status Based on Balance

CREATE OR REPLACE PROCEDURE PromoteToVIP IS

    CURSOR cur IS

        SELECT CustomerID, Balance FROM Accounts;

    cur\_CustomerID Accounts.CustomerID%TYPE;

    cur\_Balance Accounts.Balance%TYPE;

BEGIN

    FOR account\_rec IN cur LOOP

        IF account\_rec.Balance > 10000 THEN

            UPDATE Customers

            SET IsVIP = 'Y'

            WHERE CustomerID = account\_rec.CustomerID;

        ELSE

            UPDATE Customers

            SET IsVIP = 'N'

            WHERE CustomerID = account\_rec.CustomerID;

        END IF;

    END LOOP;

END PromoteToVIP;

/

-- Procedure to Send Loan Reminders for Loans Due Within the Next 30 Days

CREATE OR REPLACE PROCEDURE SendLoanReminders IS

    CURSOR cur IS

        SELECT CustomerID, EndDate

        FROM Loans

        WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

    cur\_CustomerID Loans.CustomerID%TYPE;

    cur\_EndDate Loans.EndDate%TYPE;

BEGIN

    FOR loan\_rec IN cur LOOP

        DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || ' for CustomerID: ' || loan\_rec.CustomerID);

    END LOOP;

END SendLoanReminders;

/

-- Procedure to Handle Exceptions During Fund Transfers Between Accounts

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

    fromAccountID IN INT,

    toAccountID IN INT,

    amount IN DECIMAL

) IS

BEGIN

    DECLARE

        insufficientFunds EXCEPTION;

        PRAGMA EXCEPTION\_INIT(insufficientFunds, -20001);

    BEGIN

        -- Start transaction

        SAVEPOINT start\_trans;

        -- Check if the from account has enough balance

        DECLARE

            from\_balance INT;

        BEGIN

            SELECT Balance INTO from\_balance

            FROM Accounts

            WHERE AccountID = fromAccountID;

            IF from\_balance < amount THEN

                DBMS\_OUTPUT.PUT\_LINE('Insufficient Funds');

                RAISE insufficientFunds;

            END IF;

        END;

        -- Deduct amount from fromAccount

        UPDATE Accounts

        SET Balance = Balance - amount

        WHERE AccountID = fromAccountID;

        -- Add amount to toAccount

        UPDATE Accounts

        SET Balance = Balance + amount

        WHERE AccountID = toAccountID;

        DBMS\_OUTPUT.PUT\_LINE('Funds transfered safely...');

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN insufficientFunds THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Insufficient funds for transfer from AccountID: ' || fromAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

        WHEN OTHERS THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('SQL Error during transfer from AccountID: ' || fromAccountID || ' to AccountID: ' || toAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

    END;

END SafeTransferFunds;

/

-- Procedure to Manage Errors When Updating Employee Salaries

CREATE OR REPLACE PROCEDURE UpdateSalary(

    empID IN INT,

    percentageIncrease IN DECIMAL

) IS

BEGIN

    DECLARE

        empNotFound EXCEPTION;

        PRAGMA EXCEPTION\_INIT(empNotFound, -20001);

    BEGIN

        -- Start transaction

        SAVEPOINT start\_trans;

        -- Update salary

        UPDATE Employees

        SET Salary = Salary + (Salary \* (percentageIncrease / 100))

        WHERE EmployeeID = empID;

        DBMS\_OUTPUT.PUT\_LINE('Employee salary increased by ' || percentageIncrease || ' percentage');

        -- Check if the update affected any row

        IF SQL%ROWCOUNT = 0 THEN

            RAISE empNotFound;

        END IF;

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN empNotFound THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Employee ID does not exist: ' || empID, SYSDATE);

            DBMS\_OUTPUT.PUT\_LINE('Employee ID does not exist!!');

            ROLLBACK TO start\_trans;

        WHEN OTHERS THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Error updating salary for EmployeeID: ' || empID, SYSDATE);

            ROLLBACK TO start\_trans;

    END;

END UpdateSalary;

/

-- Procedure to Ensure Data Integrity When Adding a New Customer

CREATE OR REPLACE PROCEDURE AddNewCustomer(

    CusID IN NUMBER,

    CusName IN VARCHAR2,

    CusDOB IN DATE,

    CusBalance IN NUMBER,

    CusLastModified IN DATE

)

IS

  Invalid\_Customer\_ID EXCEPTION;

  Customer\_Count NUMBER;

BEGIN

  SELECT count(\*) INTO Customer\_Count FROM Customers WHERE CustomerID = CusID;

  IF Customer\_Count > 0 THEN

    RAISE Invalid\_Customer\_ID;

  END IF;

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

  VALUES (CusID, CusName, CusDOB, CusBalance, CusLastModified);

  DBMS\_OUTPUT.PUT\_LINE('Customer registered Successfully');

EXCEPTION

  WHEN Invalid\_Customer\_ID THEN

    DBMS\_OUTPUT.PUT\_LINE('Invalid Customer ID');

END;

/

-- Call the procedures

BEGIN

    ApplyInterestDiscount;

END;

/

BEGIN

    PromoteToVIP;

END;

/

BEGIN

    SendLoanReminders;

END;

/

-- Example calls for SafeTransferFunds, UpdateSalary, and AddNewCustomer

BEGIN

    SafeTransferFunds(1, 2, 100.00);

END;

/

BEGIN

    UpdateSalary(1, 10.00);

END;

/

BEGIN

AddNewCustomer(CusID=>1,CusName=>'VARDHAN',CusDOB=>SYSDATE,CusBalance=>5000,CusLastModified=>SYSDATE);

END;

/

# EXERCISE – 2

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Insert data

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'));

-- Procedure to Apply Interest Discount to Customers Above 60 Years Old

CREATE OR REPLACE PROCEDURE ApplyInterestDiscount IS

    CURSOR cur IS

        SELECT CustomerID, DOB FROM Customers;

    cur\_CustomerID Customers.CustomerID%TYPE;

    cur\_DOB Customers.DOB%TYPE;

    currentDATE DATE := SYSDATE;

BEGIN

    FOR customer\_rec IN cur LOOP

        IF MONTHS\_BETWEEN(currentDATE, customer\_rec.DOB) / 12 > 60 THEN

            UPDATE Loans

            SET InterestRate = InterestRate \* 0.99

            WHERE CustomerID = customer\_rec.CustomerID;

        END IF;

    END LOOP;

END ApplyInterestDiscount;

/

-- Procedure to Promote Customers to VIP Status Based on Balance

CREATE OR REPLACE PROCEDURE PromoteToVIP IS

    CURSOR cur IS

        SELECT CustomerID, Balance FROM Accounts;

    cur\_CustomerID Accounts.CustomerID%TYPE;

    cur\_Balance Accounts.Balance%TYPE;

BEGIN

    FOR account\_rec IN cur LOOP

        IF account\_rec.Balance > 10000 THEN

            UPDATE Customers

            SET IsVIP = 'Y'

            WHERE CustomerID = account\_rec.CustomerID;

        ELSE

            UPDATE Customers

            SET IsVIP = 'N'

            WHERE CustomerID = account\_rec.CustomerID;

        END IF;

    END LOOP;

END PromoteToVIP;

/

-- Procedure to Send Loan Reminders for Loans Due Within the Next 30 Days

CREATE OR REPLACE PROCEDURE SendLoanReminders IS

    CURSOR cur IS

        SELECT CustomerID, EndDate

        FROM Loans

        WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

    cur\_CustomerID Loans.CustomerID%TYPE;

    cur\_EndDate Loans.EndDate%TYPE;

BEGIN

    FOR loan\_rec IN cur LOOP

        DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || ' for CustomerID: ' || loan\_rec.CustomerID);

    END LOOP;

END SendLoanReminders;

/

-- Procedure to Handle Exceptions During Fund Transfers Between Accounts

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

    fromAccountID IN INT,

    toAccountID IN INT,

    amount IN DECIMAL

) IS

BEGIN

    DECLARE

        insufficientFunds EXCEPTION;

        PRAGMA EXCEPTION\_INIT(insufficientFunds, -20001);

    BEGIN

        -- Start transaction

        SAVEPOINT start\_trans;

        -- Check if the from account has enough balance

        DECLARE

            from\_balance INT;

        BEGIN

            SELECT Balance INTO from\_balance

            FROM Accounts

            WHERE AccountID = fromAccountID;

            IF from\_balance < amount THEN

                DBMS\_OUTPUT.PUT\_LINE('Insufficient Funds');

                RAISE insufficientFunds;

            END IF;

        END;

        -- Deduct amount from fromAccount

        UPDATE Accounts

        SET Balance = Balance - amount

        WHERE AccountID = fromAccountID;

        -- Add amount to toAccount

        UPDATE Accounts

        SET Balance = Balance + amount

        WHERE AccountID = toAccountID;

        DBMS\_OUTPUT.PUT\_LINE('Funds transfered safely...');

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN insufficientFunds THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Insufficient funds for transfer from AccountID: ' || fromAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

        WHEN OTHERS THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('SQL Error during transfer from AccountID: ' || fromAccountID || ' to AccountID: ' || toAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

    END;

END SafeTransferFunds;

/

-- Procedure to Manage Errors When Updating Employee Salaries

CREATE OR REPLACE PROCEDURE UpdateSalary(

    empID IN INT,

    percentageIncrease IN DECIMAL

) IS

BEGIN

    DECLARE

        empNotFound EXCEPTION;

        PRAGMA EXCEPTION\_INIT(empNotFound, -20001);

    BEGIN

        -- Start transaction

        SAVEPOINT start\_trans;

        -- Update salary

        UPDATE Employees

        SET Salary = Salary + (Salary \* (percentageIncrease / 100))

        WHERE EmployeeID = empID;

        DBMS\_OUTPUT.PUT\_LINE('Employee salary increased by ' || percentageIncrease || ' percentage');

        -- Check if the update affected any row

        IF SQL%ROWCOUNT = 0 THEN

            RAISE empNotFound;

        END IF;

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN empNotFound THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Employee ID does not exist: ' || empID, SYSDATE);

            DBMS\_OUTPUT.PUT\_LINE('Employee ID does not exist!!');

            ROLLBACK TO start\_trans;

        WHEN OTHERS THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Error updating salary for EmployeeID: ' || empID, SYSDATE);

            ROLLBACK TO start\_trans;

    END;

END UpdateSalary;

/

-- Procedure to Ensure Data Integrity When Adding a New Customer

CREATE OR REPLACE PROCEDURE AddNewCustomer(

    CusID IN NUMBER,

    CusName IN VARCHAR2,

    CusDOB IN DATE,

    CusBalance IN NUMBER,

    CusLastModified IN DATE

)

IS

  Invalid\_Customer\_ID EXCEPTION;

  Customer\_Count NUMBER;

BEGIN

  SELECT count(\*) INTO Customer\_Count FROM Customers WHERE CustomerID = CusID;

  IF Customer\_Count > 0 THEN

    RAISE Invalid\_Customer\_ID;

  END IF;

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

  VALUES (CusID, CusName, CusDOB, CusBalance, CusLastModified);

  DBMS\_OUTPUT.PUT\_LINE('Customer registered Successfully');

EXCEPTION

  WHEN Invalid\_Customer\_ID THEN

    DBMS\_OUTPUT.PUT\_LINE('Invalid Customer ID');

END;

/

-- Call the procedures

BEGIN

    ApplyInterestDiscount;

END;

/

BEGIN

    PromoteToVIP;

END;

/

BEGIN

    SendLoanReminders;

END;

/

-- Example calls for SafeTransferFunds, UpdateSalary, and AddNewCustomer

BEGIN

    SafeTransferFunds(1, 2, 100.00);

END;

/

BEGIN

    UpdateSalary(1, 10.00);

END;

/

BEGIN

AddNewCustomer(CusID=>1,CusName=>'SHANMUKA',CusDOB=>SYSDATE,CusBalance=>5000,CusLastModified=>SYSDATE);

END;

/

# EXERCISE –3

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    DepartmentID INT,

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', '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 Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

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'));

-- SCENARIO - 1

-- Procedure to Process Monthly Interest for All Savings Accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

    -- Variable to store the number of rows updated

    v\_rows\_updated NUMBER;

BEGIN

    -- Update balance for all savings accounts by applying an interest rate of 1%

    UPDATE Accounts

    SET Balance = Balance \* 1.01

    WHERE AccountType = 'Savings';

    -- Get the number of rows updated

    v\_rows\_updated := SQL%ROWCOUNT;

    -- Display the number of rows updated

    DBMS\_OUTPUT.PUT\_LINE('Number of accounts updated: ' || v\_rows\_updated);

END ProcessMonthlyInterest;

/

-- SCENARIO -2

-- Procedure to Update Employee Bonus Based on Performance

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

    departmentID IN INT,

    bonusPercentage IN DECIMAL

) IS

BEGIN

    -- Update the salary of employees in the specified department by adding the bonus percentage

    UPDATE Employees

    SET Salary = Salary + (Salary \* (bonusPercentage / 100))

    WHERE DepartmentID = departmentID;

END UpdateEmployeeBonus;

/

-- SCENARIO - 3

-- Procedure to Transfer Funds Between Accounts

CREATE OR REPLACE PROCEDURE TransferFunds(

    fromAccountID IN INT,

    toAccountID IN INT,

    amount IN DECIMAL

) IS

BEGIN

    DECLARE

        insufficientFunds EXCEPTION;

        PRAGMA EXCEPTION\_INIT(insufficientFunds, -20001);

    BEGIN

        -- Start transaction

        SAVEPOINT start\_trans;

        -- Check if the from account has enough balance

        DECLARE

            from\_balance INT;

        BEGIN

            SELECT Balance INTO from\_balance

            FROM Accounts

            WHERE AccountID = fromAccountID;

            IF from\_balance < amount THEN

                RAISE insufficientFunds;

            END IF;

        END;

        -- Deduct amount from the source account

        UPDATE Accounts

        SET Balance = Balance - amount

        WHERE AccountID = fromAccountID;

        -- Add amount to the destination account

        UPDATE Accounts

        SET Balance = Balance + amount

        WHERE AccountID = toAccountID;

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN insufficientFunds THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Insufficient funds for transfer from AccountID: ' || fromAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

        WHEN OTHERS THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('SQL Error during transfer from AccountID: ' || fromAccountID || ' to AccountID: ' || toAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

    END;

END TransferFunds;

/

-- Example calls for the procedures

BEGIN

    ProcessMonthlyInterest;

END;

/

BEGIN

    UpdateEmployeeBonus(1, 10.00);

END;

/

BEGIN

    TransferFunds(1, 2, 100.00);

END;

/

# EXERCISE – 3

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    DepartmentID INT,

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', '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 Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

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'));

-- SCENARIO - 1

-- Procedure to Process Monthly Interest for All Savings Accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

    -- Variable to store the number of rows updated

    v\_rows\_updated NUMBER;

BEGIN

    -- Update balance for all savings accounts by applying an interest rate of 1%

    UPDATE Accounts

    SET Balance = Balance \* 1.01

    WHERE AccountType = 'Savings';

    -- Get the number of rows updated

    v\_rows\_updated := SQL%ROWCOUNT;

    -- Display the number of rows updated

    DBMS\_OUTPUT.PUT\_LINE('Number of accounts updated: ' || v\_rows\_updated);

END ProcessMonthlyInterest;

/

-- SCENARIO -2

-- Procedure to Update Employee Bonus Based on Performance

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

    departmentID IN INT,

    bonusPercentage IN DECIMAL

) IS

BEGIN

    -- Update the salary of employees in the specified department by adding the bonus percentage

    UPDATE Employees

    SET Salary = Salary + (Salary \* (bonusPercentage / 100))

    WHERE DepartmentID = departmentID;

END UpdateEmployeeBonus;

/

-- SCENARIO - 3

-- Procedure to Transfer Funds Between Accounts

CREATE OR REPLACE PROCEDURE TransferFunds(

    fromAccountID IN INT,

    toAccountID IN INT,

    amount IN DECIMAL

) IS

BEGIN

    DECLARE

        insufficientFunds EXCEPTION;

        PRAGMA EXCEPTION\_INIT(insufficientFunds, -20001);

    BEGIN

        -- Start transaction

        SAVEPOINT start\_trans;

        -- Check if the from account has enough balance

        DECLARE

            from\_balance INT;

        BEGIN

            SELECT Balance INTO from\_balance

            FROM Accounts

            WHERE AccountID = fromAccountID;

            IF from\_balance < amount THEN

                RAISE insufficientFunds;

            END IF;

        END;

        -- Deduct amount from the source account

        UPDATE Accounts

        SET Balance = Balance - amount

        WHERE AccountID = fromAccountID;

        -- Add amount to the destination account

        UPDATE Accounts

        SET Balance = Balance + amount

        WHERE AccountID = toAccountID;

        -- Commit transaction

        COMMIT;

    EXCEPTION

        WHEN insufficientFunds THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('Insufficient funds for transfer from AccountID: ' || fromAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

        WHEN OTHERS THEN

            INSERT INTO ErrorLogs (ErrorMessage, ErrorDate)

            VALUES ('SQL Error during transfer from AccountID: ' || fromAccountID || ' to AccountID: ' || toAccountID, SYSDATE);

            ROLLBACK TO start\_trans;

    END;

END TransferFunds;

/

-- Example calls for the procedures

BEGIN

    ProcessMonthlyInterest;

END;

/

BEGIN

    UpdateEmployeeBonus(1, 10.00);

END;

/

BEGIN

    TransferFunds(1, 2, 100.00);

END;

/

# EXERCISE – 4

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    DepartmentID INT,

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', '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 Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

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'));

-- Function to Calculate Age of Customers

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE)

RETURN NUMBER

IS

    age NUMBER;

BEGIN

    -- Calculate age based on the date of birth

    SELECT FLOOR((SYSDATE - dob) / 365)

    INTO age

    FROM dual;

    RETURN age;

END CalculateAge;

/

SHOW ERRORS FUNCTION CalculateAge;

-- Function to Compute Monthly Installment for a Loan

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

    loanAmount NUMBER,

    annualInterestRate NUMBER,

    loanDurationYears NUMBER

)

RETURN NUMBER

IS

    monthlyRate NUMBER;

    numPayments NUMBER;

    monthlyInstallment NUMBER;

BEGIN

    -- Calculate monthly interest rate

    monthlyRate := annualInterestRate / 100 / 12;

    -- Calculate total number of payments

    numPayments := loanDurationYears \* 12;

    -- Calculate the monthly installment using the formula

    monthlyInstallment := loanAmount \* (monthlyRate \* POWER(1 + monthlyRate, numPayments)) / (POWER(1 + monthlyRate, numPayments) - 1);

    RETURN monthlyInstallment;

END CalculateMonthlyInstallment;

/

SHOW ERRORS FUNCTION CalculateMonthlyInstallment;

-- Function to Check if Customer Has Sufficient Balance

CREATE OR REPLACE FUNCTION HasSufficientBalance(

    accountID NUMBER,

    amount NUMBER

)

RETURN BOOLEAN

IS

    balance NUMBER;

BEGIN

    -- Get the current balance of the account

    SELECT Balance INTO balance

    FROM Accounts

    WHERE AccountID = accountID AND ROWNUM = 1;

    -- Check if the balance is sufficient

    IF balance >= amount THEN

        RETURN TRUE;

    ELSE

        RETURN FALSE;

    END IF;

END HasSufficientBalance;

/

SHOW ERRORS FUNCTION HasSufficientBalance;

-- Anonymous block to test the functions

DECLARE

    v\_age NUMBER;

    v\_monthlyInstallment NUMBER;

    v\_hasSufficientBalance BOOLEAN;

BEGIN

    -- Test CalculateAge function

    v\_age := CalculateAge(TO\_DATE('1980-01-01', 'YYYY-MM-DD'));

    DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_age);

    -- Test CalculateMonthlyInstallment function

    v\_monthlyInstallment := CalculateMonthlyInstallment(10000, 5, 10);

    DBMS\_OUTPUT.PUT\_LINE('Monthly Installment: ' || v\_monthlyInstallment);

    -- Test HasSufficientBalance function

    v\_hasSufficientBalance := HasSufficientBalance(1, 500);

    IF v\_hasSufficientBalance THEN

        DBMS\_OUTPUT.PUT\_LINE('Sufficient Balance: TRUE');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Sufficient Balance: FALSE');

    END IF;

END;

/

# EXERCISE – 5

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    DepartmentID INT,

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Create AuditLog table for maintaining an audit log

CREATE TABLE AuditLog (

    AuditID INT PRIMARY KEY,

    TransactionID INT,

    TransactionDate DATE,

    AccountID INT,

    Amount INT,

    TransactionType VARCHAR2(10)

);

-- Create sequence for AuditLog primary key

CREATE SEQUENCE AuditLog\_Seq

START WITH 1

INCREMENT BY 1

NOCACHE;

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', '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 Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE + 25);

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'));

-- Triggers

-- Trigger to Automatically Update LastModified Date When a Customer's Record is Updated

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

    -- Update the LastModified column to the current date

    :NEW.LastModified := SYSDATE;

END;

/

SHOW ERRORS TRIGGER UpdateCustomerLastModified;

-- Trigger to Maintain an Audit Log for All Transactions

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

    -- Insert transaction details into the AuditLog table

    INSERT INTO AuditLog (AuditID, TransactionID, TransactionDate, AccountID, Amount, TransactionType)

    VALUES (AuditLog\_Seq.NEXTVAL, :NEW.TransactionID, :NEW.TransactionDate, :NEW.AccountID, :NEW.Amount, :NEW.TransactionType);

END;

/

SHOW ERRORS TRIGGER LogTransaction;

-- Trigger to Enforce Business Rules on Deposits and Withdrawals

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

    currentBalance DECIMAL(15,2);

BEGIN

    -- Check if the transaction type is withdrawal and ensure it does not exceed the balance

    IF :NEW.TransactionType = 'WITHDRAWAL' THEN

        -- Get the current balance of the account

        SELECT Balance INTO currentBalance

        FROM Accounts

        WHERE AccountID = :NEW.AccountID

        FOR UPDATE;

        -- Ensure the withdrawal does not exceed the balance

        IF :NEW.Amount > currentBalance THEN

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

        END IF;

    END IF;

    -- Check if the transaction type is deposit and ensure the amount is positive

    IF :NEW.TransactionType = 'DEPOSIT' THEN

        IF :NEW.Amount <= 0 THEN

            RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

        END IF;

    END IF;

END;

/

SHOW ERRORS TRIGGER CheckTransactionRules;

-- Anonymous block to test the triggers

BEGIN

    -- Update a customer record to test UpdateCustomerLastModified trigger

    UPDATE Customers SET Name = 'John Doe Updated' WHERE CustomerID = 1;

    -- Insert a transaction to test LogTransaction and CheckTransactionRules triggers

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

    VALUES (3, 1, SYSDATE, 500, 'DEPOSIT');

    -- Insert a withdrawal transaction to test CheckTransactionRules trigger

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

    VALUES (4, 1, SYSDATE, 200, 'WITHDRAWAL');

    -- Test insufficient funds scenario

    BEGIN

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

        VALUES (5, 1, SYSDATE, 2000, 'WITHDRAWAL');

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('before insufficient funds');

            DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

            DBMS\_OUTPUT.PUT\_LINE('after insufficient funds');

    END;

    -- Test negative deposit scenario

    BEGIN

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

        VALUES (6, 1, SYSDATE, -500, 'DEPOSIT');

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('before negative deposit');

            DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

            DBMS\_OUTPUT.PUT\_LINE('after negative deposit');

    END;

END;

/

# EXERCISE – 6

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    DepartmentID INT,

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', '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 Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

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'));

-- Scenario 1: Generate Monthly Statements for All Customers

CREATE OR REPLACE PROCEDURE GenerateMonthlyStatements AS

    -- Declare cursor and variables

    CURSOR c\_monthly\_statements IS

        SELECT a.CustomerID, t.TransactionDate, t.Amount

        FROM Transactions t

        JOIN Accounts a ON t.AccountID = a.AccountID

        WHERE EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

          AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

    v\_customer\_id INT;

    v\_transaction\_date DATE;

    v\_amount DECIMAL(10, 2);

BEGIN

    -- Loop through cursor

    FOR r\_monthly\_statement IN c\_monthly\_statements LOOP

        v\_customer\_id := r\_monthly\_statement.CustomerID;

        v\_transaction\_date := r\_monthly\_statement.TransactionDate;

        v\_amount := r\_monthly\_statement.Amount;

        -- Print or process the statement

        DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || v\_customer\_id ||

                             ', Date: ' || v\_transaction\_date ||

                             ', Amount: ' || v\_amount);

    END LOOP;

END;

/

-- SHOW ERRORS PROCEDURE GenerateMonthlyStatements;

-- Scenario 2: Apply Annual Fee to All Accounts

CREATE OR REPLACE PROCEDURE ApplyAnnualFee AS

    -- Declare cursor and variables

    CURSOR c\_accounts IS

        SELECT AccountID, Balance

        FROM Accounts;

    v\_account\_id INT;

    v\_balance DECIMAL(10, 2);

    v\_annual\_fee DECIMAL(10, 2) := 50.00;  -- Example annual fee amount

BEGIN

    -- Loop through cursor

    FOR r\_account IN c\_accounts LOOP

        v\_account\_id := r\_account.AccountID;

        v\_balance := r\_account.Balance;

        -- Deduct annual fee from account balance

        UPDATE Accounts

        SET Balance = v\_balance - v\_annual\_fee

        WHERE AccountID = v\_account\_id;

        DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || v\_account\_id || ' - Annual fee applied.');

    END LOOP;

END;

/

-- Scenario 3: Update Interest Rates for All Loans Based on a New Policy

CREATE OR REPLACE PROCEDURE UpdateLoanInterestRates AS

    -- Declare cursor and variables

    CURSOR c\_loans IS

        SELECT LoanID, InterestRate

        FROM Loans;

    v\_loan\_id INT;

    v\_interest\_rate DECIMAL(5, 2);

    v\_new\_interest\_rate DECIMAL(5, 2);

BEGIN

    -- Loop through cursor

    FOR r\_loan IN c\_loans LOOP

        v\_loan\_id := r\_loan.LoanID;

        v\_interest\_rate := r\_loan.InterestRate;

        -- Apply new interest rate based on policy

        v\_new\_interest\_rate := v\_interest\_rate \* 1.05;  -- Example: increase by 5%

        UPDATE Loans

        SET InterestRate = v\_new\_interest\_rate

        WHERE LoanID = v\_loan\_id;

        DBMS\_OUTPUT.PUT\_LINE('LoanID: ' || v\_loan\_id || ' - Interest rate updated to: ' || v\_new\_interest\_rate);

    END LOOP;

END;

/

BEGIN

    GenerateMonthlyStatements;

END;

/

BEGIN

    ApplyAnnualFee;

END;

/

BEGIN

    UpdateLoanInterestRates;

END;

/

# EXERCISE – 7

-- Enable server output

SET SERVEROUTPUT ON;

-- Create tables

CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance INT,

    LastModified DATE,

    IsVIP CHAR(1)

);

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    CustomerID INT,

    AccountType VARCHAR2(20),

    Balance INT,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID INT PRIMARY KEY,

    AccountID INT,

    TransactionDate DATE,

    Amount INT,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID INT PRIMARY KEY,

    CustomerID INT,

    LoanAmount INT,

    InterestRate INT,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary INT,

    Department VARCHAR2(50),

    DepartmentID INT,

    HireDate DATE

);

-- Create ErrorLogs table for logging errors

CREATE TABLE ErrorLogs (

    ErrorID INT PRIMARY KEY,

    ErrorMessage VARCHAR2(255),

    ErrorDate DATE

);

-- Scripts for Sample Data Insertion

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

VALUES (1, 'John Doe', TO\_DATE('1963-05-15', '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 Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 10000, 5, SYSDATE, SYSDATE+25);

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'));

-- SCENARIO - 1

-- Package for Customer Management

CREATE OR REPLACE PROCEDURE AddCustomer(

    p\_CustomerID IN INT,

    p\_Name IN VARCHAR2,

    p\_DOB IN DATE,

    p\_Balance IN DECIMAL

) AS

BEGIN

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

    VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance);

    INSERT INTO Accounts (CustomerID, Balance)

    VALUES (p\_CustomerID, p\_Balance);

END;

/

CREATE OR REPLACE PROCEDURE UpdateCustomerDetails(

    p\_CustomerID IN INT,

    p\_Name IN VARCHAR2,

    p\_DOB IN DATE

) AS

BEGIN

    UPDATE Customers

    SET Name = p\_Name, DOB = p\_DOB

    WHERE CustomerID = p\_CustomerID;

END;

/

CREATE OR REPLACE FUNCTION GetCustomerBalance(

    p\_CustomerID IN INT

)

RETURN DECIMAL IS

    customer\_balance DECIMAL(10, 2);

BEGIN

    SELECT Balance INTO customer\_balance

    FROM Accounts

    WHERE CustomerID = p\_CustomerID;

    RETURN customer\_balance;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 0; -- Return 0 if no balance found

END;

/

-- SCENARIO - 2

-- Package for Employee Management

CREATE OR REPLACE PROCEDURE HireEmployee(

    p\_EmployeeID IN INT,

    p\_Name IN VARCHAR2,

    p\_Position IN VARCHAR2,

    p\_Salary IN DECIMAL

) AS

BEGIN

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

    VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary);

END;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeDetails(

    p\_EmployeeID IN INT,

    p\_Name IN VARCHAR2,

    p\_Position IN VARCHAR2,

    p\_Salary IN DECIMAL

) AS

BEGIN

    UPDATE Employees

    SET Name = p\_Name, Position = p\_Position, Salary = p\_Salary

    WHERE EmployeeID = p\_EmployeeID;

END;

/

CREATE OR REPLACE FUNCTION CalculateAnnualSalary(

    p\_EmployeeID IN INT

)

RETURN DECIMAL IS

    annual\_salary DECIMAL(10, 2);

BEGIN

    SELECT Salary \* 12 INTO annual\_salary

    FROM Employees

    WHERE EmployeeID = p\_EmployeeID;

    RETURN annual\_salary;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 0; -- Return 0 if employee not found

END;

/

-- SCENARIO - 3

-- Package for Account Operations

CREATE OR REPLACE PROCEDURE OpenAccount(

    p\_CustomerID IN INT,

    p\_Balance IN DECIMAL

) AS

BEGIN

    INSERT INTO Accounts (CustomerID, Balance)

    VALUES (p\_CustomerID, p\_Balance);

END;

/

CREATE OR REPLACE PROCEDURE CloseAccount(

    p\_CustomerID IN INT

) AS

BEGIN

    DELETE FROM Accounts

    WHERE CustomerID = p\_CustomerID;

END;

/

CREATE OR REPLACE FUNCTION GetTotalBalance(

    p\_CustomerID IN INT

)

RETURN DECIMAL IS

    total\_balance DECIMAL(10, 2);

BEGIN

    SELECT SUM(Balance) INTO total\_balance

    FROM Accounts

    WHERE CustomerID = p\_CustomerID;

    RETURN total\_balance;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 0; -- Return 0 if no accounts found

END;

/