**Exercise 1: Control Structures**

**Schema & Test data :**

-- Customers Table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Loans Table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Sample Data

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

VALUES (1, 'John Doe', TO\_DATE('1950-01-01', 'YYYY-MM-DD'), 5000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 3000, SYSDATE);

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

VALUES (1, 1, 8000, 12, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

VALUES (2, 2, 6000, 10, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

COMMIT;

**PL/SQL Logic :**

BEGIN

FOR cust IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

) LOOP

IF FLOOR(MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

END IF;

END LOOP;

COMMIT;

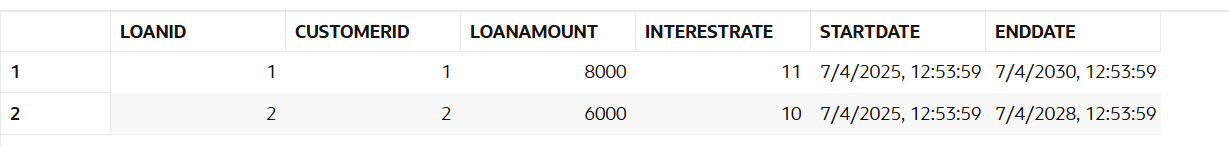
END;

/

**Query :**

SELECT \* FROM Loans;

**Output :**



**Exercise 2: Error Handling**

**Schema & Test data :**

**PL/SQL Logic :**

-- Drop table if already exists

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Create Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Insert sample data

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'));

COMMIT;

**Query :**

-- Procedure to update salary with error handling

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_Percent IN NUMBER

)

IS

v\_OldSalary NUMBER;

BEGIN

-- Get current salary

SELECT Salary INTO v\_OldSalary FROM Employees WHERE EmployeeID = p\_EmployeeID;

-- Update salary

UPDATE Employees

SET Salary = Salary + (Salary \* p\_Percent / 100)

WHERE EmployeeID = p\_EmployeeID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully. New salary = ' ||

(v\_OldSalary + (v\_OldSalary \* p\_Percent / 100)));

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' does not exist.');

WHEN OTHERS THEN

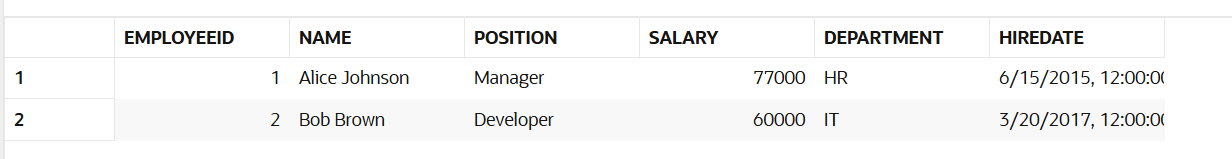
ROLLBACK;

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

END;

/

**Output :**



**Exercise 3: Stored Procedures**

**Schema & Test data :**

-- Create Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

-- Insert sample data

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

VALUES (101, 1, 'Savings', 10000, SYSDATE);

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

VALUES (102, 2, 'Savings', 8000, SYSDATE);

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

VALUES (103, 3, 'Checking', 9000, SYSDATE);

COMMIT;

**PL/SQL Logic :**

-- Procedure to apply interest to savings accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

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

END;

/

**Query :**

-- Execute the procedure

BEGIN

ProcessMonthlyInterest;

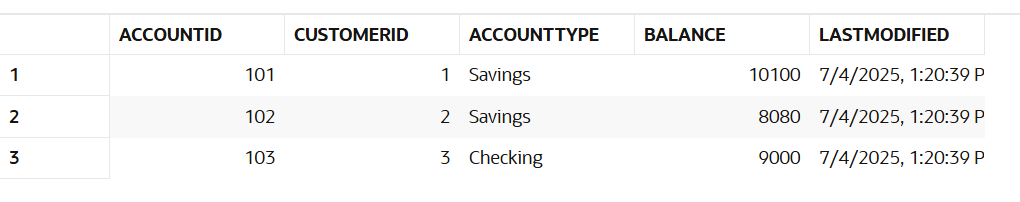
END;

/

-- Check updated balances

SELECT \* FROM Accounts;

**Output :**



**Exercise 4: Functions**

**Schema & Test data :**

-- Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Insert sample 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('1960-01-01', 'YYYY-MM-DD'), 2000, SYSDATE);

COMMIT;

**PL/SQL Logic :**

-- Function to calculate age from DOB

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB DATE)

RETURN NUMBER

IS

v\_Age NUMBER;

BEGIN

v\_Age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

/

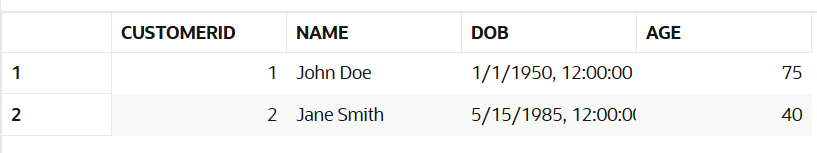
**Query :**

-- Use the function in SELECT

SELECT CustomerID, Name, DOB, CalculateAge(DOB) AS Age

FROM Customers;

**Output :**

****

**Exercise 5: Triggers**

**Schema & Test data :**

-- Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Insert sample data

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

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

COMMIT;

**PL/SQL Logic :**

-- Trigger to auto-update LastModified column

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Query :**

-- Update Balance to trigger LastModified change

UPDATE Customers

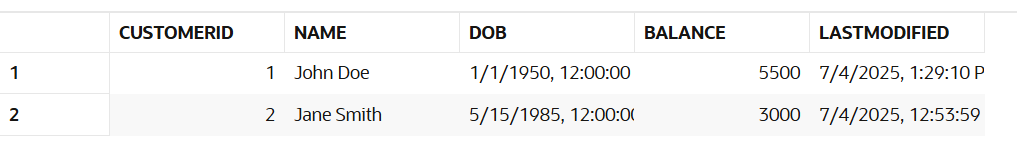
SET Balance = Balance + 500

WHERE CustomerID = 1;

-- Check updated row

SELECT \* FROM Customers;

**Output :**

****

**Exercise 6: Cursors**

**Schema & Test data :**

-- Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

-- Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

-- Sample data

INSERT INTO Accounts VALUES (1, 101, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Savings', 8000, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE - 1, 500, 'Deposit');

INSERT INTO Transactions VALUES (2, 1, SYSDATE - 5, 200, 'Withdrawal');

INSERT INTO Transactions VALUES (3, 2, SYSDATE - 10, 1000, 'Deposit');

INSERT INTO Transactions VALUES (4, 2, SYSDATE - 40, 300, 'Deposit'); -- Old transaction

COMMIT;

**PL/SQL Logic :**

DECLARE

CURSOR txn\_cursor IS

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

FROM Accounts a

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionDate >= ADD\_MONTHS(TRUNC(SYSDATE, 'MM'), 0);

v\_rec txn\_cursor%ROWTYPE;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Monthly Statements ---');

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_rec;

EXIT WHEN txn\_cursor%NOTFOUND;

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

', Customer ID: ' || v\_rec.CustomerID ||

', Date: ' || TO\_CHAR(v\_rec.TransactionDate, 'DD-MON') ||

', Amount: ' || v\_rec.Amount ||

', Type: ' || v\_rec.TransactionType);

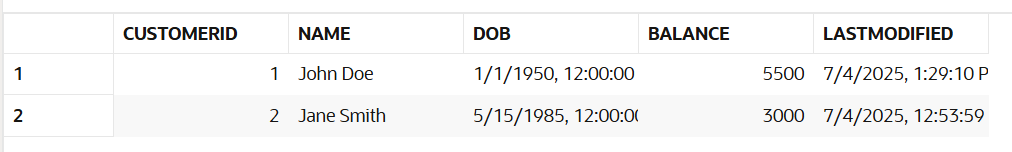
END LOOP;

CLOSE txn\_cursor;

END;

/

**Output :**

****

**Exercise 7: Packages**

**Schema & Test data :**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Insert sample customer

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

COMMIT;

**PL/SQL Logic :**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

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

VALUES (p\_ID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer already exists.');

END;

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name, Balance = p\_Balance, LastModified = SYSDATE

WHERE CustomerID = p\_ID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Customer not found.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer updated.');

END IF;

END;

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Customers WHERE CustomerID = p\_ID;

RETURN v\_Balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Customer not found.');

RETURN -1;

END;

END CustomerManagement;

/

**Query :**

-- Add new customer

BEGIN

CustomerManagement.AddCustomer(2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 2000);

END;

/

-- Update customer details

BEGIN

CustomerManagement.UpdateCustomer(2, 'Jane A. Smith', 2500);

END;

/

-- Get balance using function

DECLARE

v\_bal NUMBER;

BEGIN

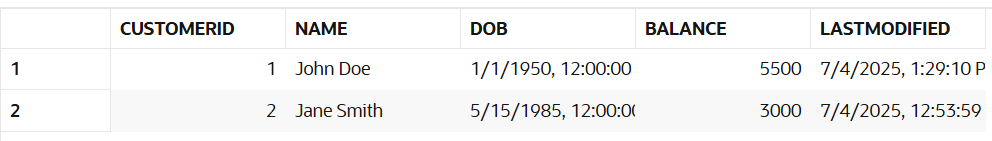
v\_bal := CustomerManagement.GetCustomerBalance(2);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || v\_bal);

END;

/

**Output :**

****