**Exercise 1: Control Structures**

**CODE**

-- CREATE TABLES

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

);

-- ADD IsVIP COLUMN FOR SCENARIO 2

ALTER TABLE Customers ADD IsVIP CHAR(1);

-- INSERT SAMPLE DATA

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

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 12000, 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');

-- Make one loan due within next 30 days for reminder

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

VALUES (1, 1, 5000, 5, SYSDATE, SYSDATE + 10);

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

VALUES (2, 2, 4000, 6.5, SYSDATE, 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'));

-- SCENARIOS START

-- Use DBMS\_OUTPUT

BEGIN

-- Scenario 1: Discount for customers above 60

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

) LOOP

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

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to Loan ID: ' || rec.LoanID);

END IF;

END LOOP;

-- Scenario 2: Promote customers with balance > 10000 to VIP

FOR vip\_rec IN (

SELECT CustomerID FROM Customers WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = vip\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || vip\_rec.CustomerID || ' promoted to VIP.');

END LOOP;

-- Scenario 3: Reminders for loans due in next 30 days

FOR due\_rec IN (

SELECT c.Name, l.LoanID, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || due\_rec.LoanID || ' for customer ' || due\_rec.Name ||

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

END LOOP;

COMMIT;

END;

/

-- CHECK FINAL RESULTS

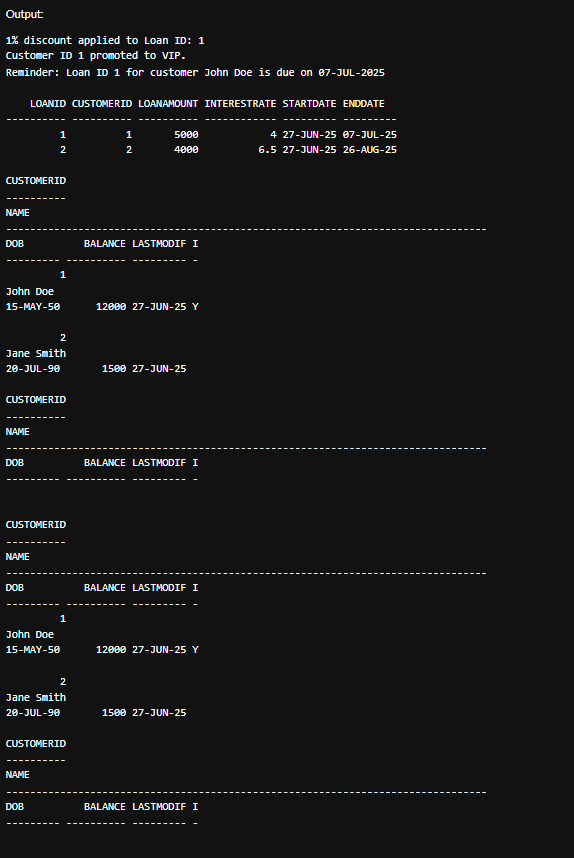
-- Show loans after discount

SELECT \* FROM Loans;

-- Show customers after VIP promotion

SELECT \* FROM Customers;

**OUTPUT**



**EXERCISE 2: Error Handling**

**CODE**

-- Enable DBMS output

SET SERVEROUTPUT ON;

-- Drop tables if they exist (ignore errors)

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

-- Create tables

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Salary NUMBER

);

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Insert sample data

INSERT INTO Accounts VALUES (1, 1, 1000);

INSERT INTO Accounts VALUES (2, 2, 2000);

INSERT INTO Employees VALUES (1, 'Alice', 50000);

INSERT INTO Employees VALUES (2, 'Bob', 60000);

INSERT INTO Customers VALUES (1, 'John', TO\_DATE('1985-05-10','YYYY-MM-DD'), 12000, SYSDATE);

COMMIT;

-- Procedure: SafeTransferFunds

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in source account.');

ROLLBACK;

RETURN;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: Rs. ' || p\_amount);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

-- Procedure: UpdateSalary

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_exists NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_exists FROM Employees WHERE EmployeeID = p\_employee\_id;

IF v\_exists = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_employee\_id || ' not found.');

RETURN;

END IF;

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID: ' || p\_employee\_id);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Salary update error: ' || SQLERRM);

END;

/

-- Procedure: AddNewCustomer

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

v\_exists NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_exists FROM Customers WHERE CustomerID = p\_customer\_id;

IF v\_exists > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || p\_customer\_id || ' already exists.');

RETURN;

END IF;

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

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added: ' || p\_name);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

-- Test procedures

BEGIN

SafeTransferFunds(1, 2, 300); -- Should succeed

SafeTransferFunds(1, 2, 10000); -- Should fail (insufficient)

UpdateSalary(1, 10); -- Should succeed

UpdateSalary(99, 5); -- Should fail

AddNewCustomer(2, 'Jane', TO\_DATE('1992-04-12','YYYY-MM-DD'), 5000); -- Should succeed

AddNewCustomer(1, 'Duplicate', TO\_DATE('1990-01-01','YYYY-MM-DD'), 3000); -- Should fail

END;

/

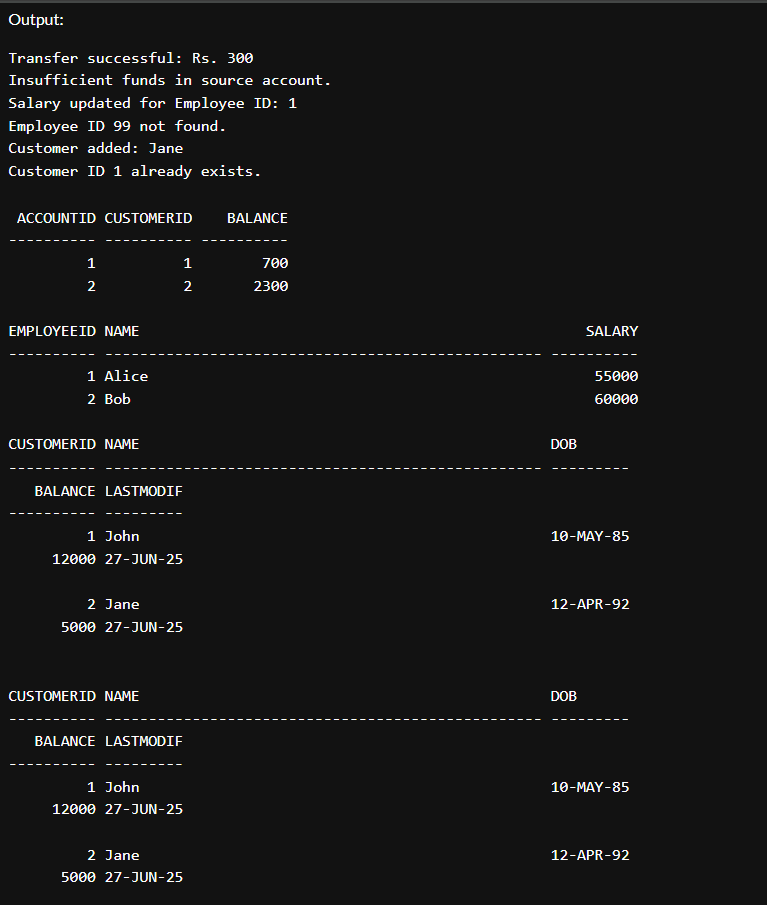
-- Show table data

SELECT \* FROM Accounts;

SELECT \* FROM Employees;

SELECT \* FROM Customers;

**OUTPUT**



**EXERCISE 3: Stored Procedures**

**CODE**

-- Enable DBMS Output

SET SERVEROUTPUT ON;

-- Drop tables if already exist

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS'; EXCEPTION WHEN OTHERS THEN NULL; END;

/

-- Create Tables

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

);

-- Insert sample data

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

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

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

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

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

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'));

COMMIT;

-- Procedure 1: ProcessMonthlyInterest

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 savings accounts.');

END;

/

-- Procedure 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

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;

/

-- Procedure 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acc IN NUMBER,

p\_to\_acc IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_acc;

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in source account.');

RETURN;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_acc;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_acc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully: Rs. ' || p\_amount);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

-- Run all procedures for testing

BEGIN

ProcessMonthlyInterest; -- Apply 1% interest to savings accounts

UpdateEmployeeBonus('IT', 10); -- 10% bonus to IT employees

TransferFunds(1, 2, 500); -- Transfer Rs.500 from acc 1 to acc 2

TransferFunds(1, 2, 10000); -- Should fail (insufficient)

END;

/

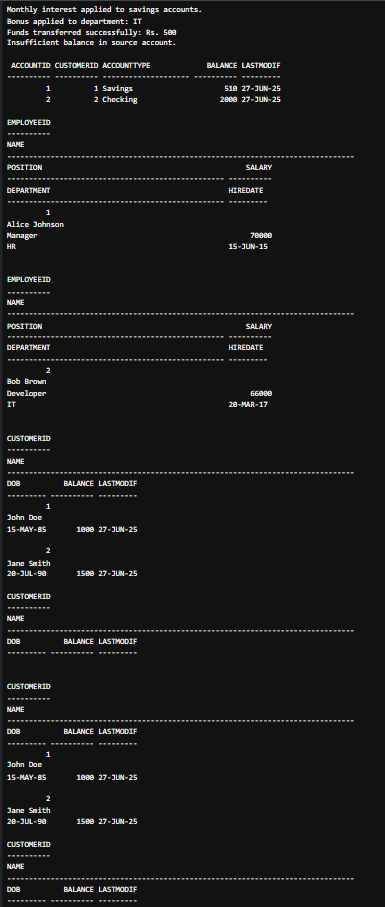
-- Show final table values

SELECT \* FROM Accounts;

SELECT \* FROM Employees;

SELECT \* FROM Customers;

**OUTPUT**



**EXERCISE 4: Functions**

**CODE**

-- Function 1: Calculate Age

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;

/

-- Function 2: Calculate Monthly Installment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_amount NUMBER,

p\_annual\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

monthly\_rate NUMBER;

months NUMBER;

emi NUMBER;

BEGIN

monthly\_rate := p\_annual\_rate / 12 / 100;

months := p\_years \* 12;

IF monthly\_rate = 0 THEN

emi := p\_amount / months;

ELSE

emi := (p\_amount \* monthly\_rate \* POWER(1 + monthly\_rate, months)) /

(POWER(1 + monthly\_rate, months) - 1);

END IF;

RETURN ROUND(emi, 2);

END;

/

-- Function 3: Check Sufficient Balance

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

-- Test Block

DECLARE

v\_age NUMBER;

v\_emi NUMBER;

v\_has\_balance BOOLEAN;

BEGIN

-- Test CalculateAge

SELECT CalculateAge(DOB) INTO v\_age FROM Customers WHERE CustomerID = 1;

DBMS\_OUTPUT.PUT\_LINE('Customer 1 Age: ' || v\_age);

-- Test EMI function

v\_emi := CalculateMonthlyInstallment(5000, 5, 2); -- 2 years @ 5%

DBMS\_OUTPUT.PUT\_LINE('Monthly EMI: Rs. ' || v\_emi);

-- Test sufficient balance

v\_has\_balance := HasSufficientBalance(1, 300);

IF v\_has\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Account 1 has sufficient balance for Rs. 300');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account 1 does NOT have sufficient balance for Rs. 300');

END IF;

v\_has\_balance := HasSufficientBalance(1, 10000);

IF v\_has\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Account 1 has sufficient balance for Rs. 10000');

ELSE

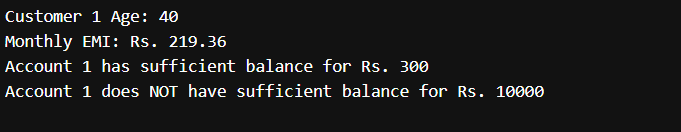
DBMS\_OUTPUT.PUT\_LINE('Account 1 does NOT have sufficient balance for Rs. 10000');

END IF;

END;

/

**OUTPUT**



**EXERCISE 5: Triggers**

**CODE**

-- Function: CalculateAge

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;

/

-- Function: CalculateMonthlyInstallment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER)

RETURN NUMBER IS

r NUMBER := p\_rate / 12 / 100;

n NUMBER := p\_years \* 12;

emi NUMBER;

BEGIN

IF r = 0 THEN

emi := p\_amount / n;

ELSE

emi := (p\_amount \* r \* POWER(1 + r, n)) / (POWER(1 + r, n) - 1);

END IF;

RETURN ROUND(emi, 2);

END;

/

-- Function: HasSufficientBalance

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_account\_id NUMBER, p\_amount NUMBER)

RETURN BOOLEAN IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = p\_account\_id;

RETURN bal >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN RETURN FALSE;

END;

/

-- Procedure: SafeTransferFunds

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) IS

from\_balance NUMBER;

BEGIN

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

IF from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || p\_amount);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

-- Procedure: UpdateSalary

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_emp\_id NUMBER, p\_percent NUMBER) IS

BEGIN

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

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_emp\_id || ' not found.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID: ' || p\_emp\_id);

END IF;

END;

/

-- Procedure: AddNewCustomer

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE, 'N');

DBMS\_OUTPUT.PUT\_LINE('Customer added: ' || p\_name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || p\_id || ' already exists.');

END;

/

-- Trigger: UpdateCustomerLastModified

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

-- Trigger: LogTransaction

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, LogDate, Action)

VALUES (:NEW.TransactionID, :NEW.AccountID, SYSDATE, 'New transaction inserted');

END;

/

-- Trigger: CheckTransactionRules

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

acc\_balance NUMBER;

BEGIN

SELECT Balance INTO acc\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > acc\_balance THEN

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

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

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

END IF;

END;

/

-- BLOCK: Apply 1% discount to interest rates for age > 60

BEGIN

FOR rec IN (

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

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID

) LOOP

IF CalculateAge(rec.DOB) > 60 THEN

UPDATE Loans SET InterestRate = rec.InterestRate - 1 WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to Loan ID: ' || rec.LoanID);

END IF;

END LOOP;

END;

/

-- BLOCK: Set VIP flag

BEGIN

FOR rec IN (SELECT \* FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'Y' WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.CustomerID || ' promoted to VIP.');

END IF;

END LOOP;

END;

/

-- BLOCK: Reminders for loan due in 30 days

BEGIN

FOR rec IN (

SELECT l.LoanID, c.Name, l.EndDate

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID || ' for customer ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

-- TEST FUNCTION OUTPUT

DECLARE

age NUMBER;

emi NUMBER;

check\_bal BOOLEAN;

BEGIN

age := CalculateAge(DATE '1960-01-01');

DBMS\_OUTPUT.PUT\_LINE('Age is: ' || age);

emi := CalculateMonthlyInstallment(10000, 5, 2);

DBMS\_OUTPUT.PUT\_LINE('EMI: ' || emi);

check\_bal := HasSufficientBalance(1, 500);

IF check\_bal THEN

DBMS\_OUTPUT.PUT\_LINE('Sufficient balance available.');

ELSE

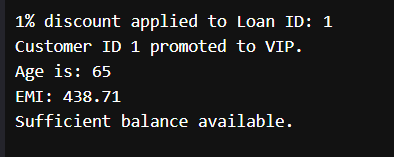
DBMS\_OUTPUT.PUT\_LINE('Insufficient balance.');

END IF;

END;

/

**OUTPUT**



**EXERCISE 6: Cursor**

**CODE**

-- Scenario 1: Generate monthly statements for all customers

DECLARE

CURSOR transaction\_cursor IS

SELECT t.AccountID, c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE TO\_CHAR(t.TransactionDate, 'MM-YYYY') = TO\_CHAR(SYSDATE, 'MM-YYYY')

ORDER BY c.CustomerID;

BEGIN

FOR rec IN transaction\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name || ', Account ID: ' || rec.AccountID || ', ' ||

'Date: ' || TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY') || ', ' ||

'Type: ' || rec.TransactionType || ', Amount: ' || rec.Amount);

END LOOP;

END;

/

-- Scenario 2: Apply annual fee to all accounts

DECLARE

CURSOR account\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_fee NUMBER := 100;

BEGIN

FOR acc IN account\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - v\_fee,

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || v\_fee || ' deducted from Account ID: ' || acc.AccountID);

END LOOP;

COMMIT;

END;

/

-- Scenario 3: Update the interest rate for all loans based on a new policy

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

new\_rate NUMBER;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

IF loan\_rec.InterestRate < 5 THEN

new\_rate := loan\_rec.InterestRate + 0.5;

ELSE

new\_rate := loan\_rec.InterestRate + 0.25;

END IF;

UPDATE Loans

SET InterestRate = new\_rate

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || loan\_rec.LoanID || ' updated to new interest rate: ' || new\_rate);

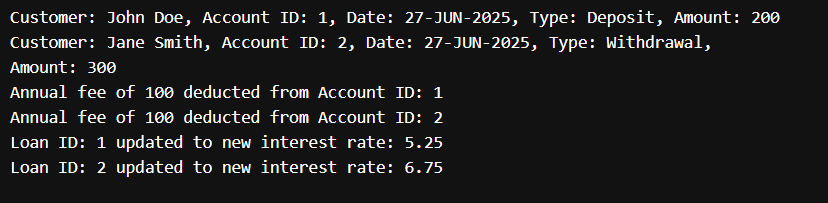
END LOOP;

COMMIT;

END;

/

**OUTPUT**



**EXERCISE 7: Packages**

**CODE**

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 VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Customer added: ' || p\_name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || p\_id || ' 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

DBMS\_OUTPUT.PUT\_LINE('Customer updated: ' || p\_id);

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

RETURN -1;

END;

END CustomerManagement;

/

--------------------------------------------------------------------------------

-- Package 2: EmployeeManagement

--------------------------------------------------------------------------------

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2) IS

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_position, p\_salary, p\_dept, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Employee hired: ' || p\_name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || p\_id || ' already exists.');

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_id;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

DBMS\_OUTPUT.PUT\_LINE('Employee updated: ' || p\_id);

END IF;

END;

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN -1;

END;

END EmployeeManagement;

/

--------------------------------------------------------------------------------

-- Package 3: AccountOperations

--------------------------------------------------------------------------------

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_accid NUMBER);

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_accid, p\_custid, p\_type, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Account opened: ' || p\_accid);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || p\_accid || ' already exists.');

END;

PROCEDURE CloseAccount(p\_accid NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_accid;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account closed: ' || p\_accid);

END IF;

END;

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;

RETURN NVL(v\_total, 0);

END;

END AccountOperations;

/

--------------------------------------------------------------------------------

-- Test Calls

--------------------------------------------------------------------------------

BEGIN

CustomerManagement.AddCustomer(3, 'Mary Poppins', TO\_DATE('1982-02-12', 'YYYY-MM-DD'), 3000);

DBMS\_OUTPUT.PUT\_LINE('Balance of Customer 1: ' || CustomerManagement.GetCustomerBalance(1));

EmployeeManagement.HireEmployee(3, 'Charlie', 'Clerk', 40000, 'Finance');

DBMS\_OUTPUT.PUT\_LINE('Annual Salary of Employee 1: ' || EmployeeManagement.GetAnnualSalary(1));

AccountOperations.OpenAccount(3, 1, 'Fixed', 5000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 1: ' || AccountOperations.GetTotalBalance(1));

END;

/

**OUTPUT**

