**Superset ID: 6416829**

**Exercise 1: Control Structures**

**QUERY:**

-- Create Tables

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP CHAR(1),

LoanInterest NUMBER

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

Amount NUMBER

);

-- Insert Sample Data

INSERT INTO Customers VALUES (1, 'John Doe', 65, 12000, 'N', 8.5);

INSERT INTO Customers VALUES (2, 'Alice Smith', 59, 9500, 'N', 7.5);

INSERT INTO Customers VALUES (3, 'Bob Brown', 72, 11000, 'N', 9.0);

INSERT INTO Customers VALUES (4, 'Emma Davis', 45, 10500, 'N', 8.2);

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10, 5000); -- Due soon

INSERT INTO Loans VALUES (102, 2, SYSDATE + 40, 3000); -- Not due soon

INSERT INTO Loans VALUES (103, 3, SYSDATE + 5, 7000); -- Due soon

COMMIT;

-- Scenario 1: Apply 1% Discount for Customers > 60

BEGIN

FOR cust IN (SELECT CustomerID, LoanInterest, Age FROM Customers) LOOP

IF cust.Age > 60 THEN

UPDATE Customers

SET LoanInterest = LoanInterest - 1

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 2: Set IsVIP = 'Y' for Balance > $10,000

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 3: Print Reminders for Loans Due in Next 30 Days

BEGIN

FOR loan IN (

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan.LoanID ||

' for Customer ID ' || loan.CustomerID ||

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

END LOOP;

END;

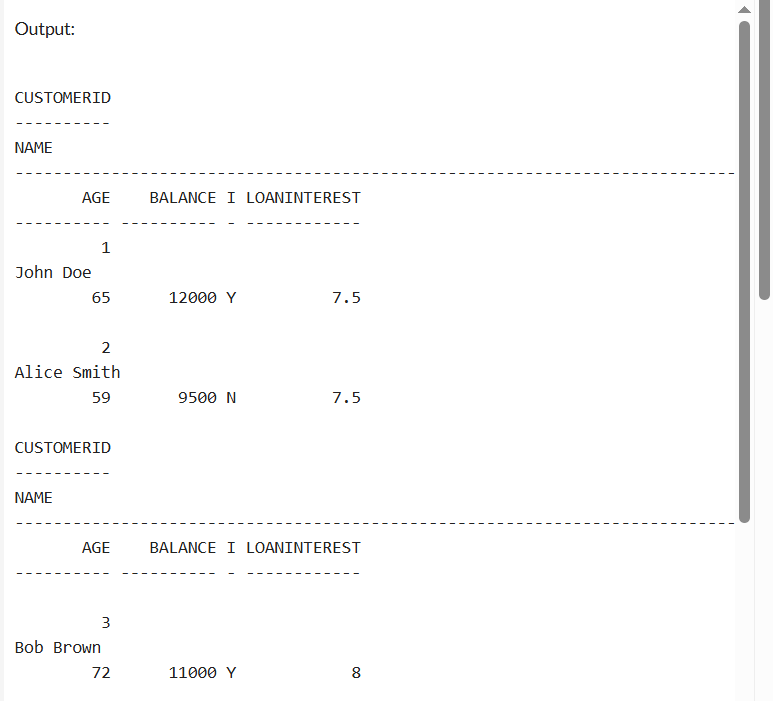
/

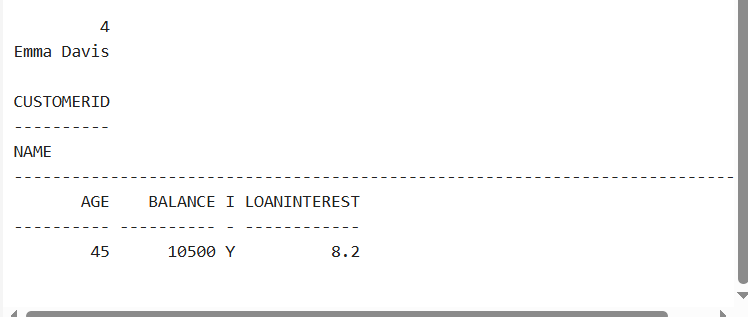
-- View Final Customer Table

SELECT \* FROM Customers;

-- View Final Loan Table

SELECT \* FROM Loans;





**Exercise 2: Error Handling**

**QUERY:**

-- Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Create Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

-- Create Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Customers

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

-- Accounts

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

INSERT INTO Accounts VALUES (2, 1, 'Checking', 500, SYSDATE);

-- Employees

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

COMMIT;

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

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 successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

/

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from IN NUMBER,

p\_to IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account');

END IF;

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_emp\_id IN NUMBER,

p\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found');

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

EXCEPTION

WHEN OTHERS THEN

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

END;

/

-- Add new customer

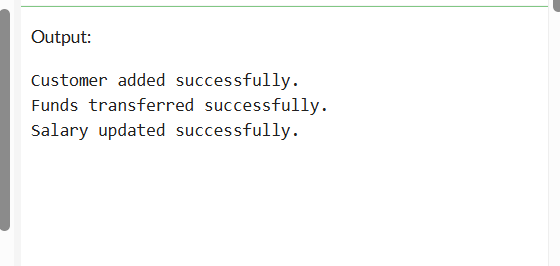
EXEC AddNewCustomer(2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500);

-- Transfer funds

EXEC SafeTransferFunds(1, 2, 200);

-- Update salary

EXEC UpdateSalary(1, 10);



**Exercise 3: Stored Procedures**

**QUERY:**

-- Create Customers Table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Create Accounts Table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

-- Create Employees Table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Insert Accounts

INSERT INTO Accounts VALUES (1, 1, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts VALUES (2, 1, 'Checking', 3000, SYSDATE);

-- Insert Employees

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', SYSDATE);

INSERT INTO Employees VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', SYSDATE);

COMMIT;

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 processed successfully.');

EXCEPTION

WHEN OTHERS THEN

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

END;

/

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;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No employees found in department ' || p\_department);

ELSE

COMMIT;

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

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

/

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

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

END IF;

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

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

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

BEGIN

UpdateEmployeeBonus('HR', 5);

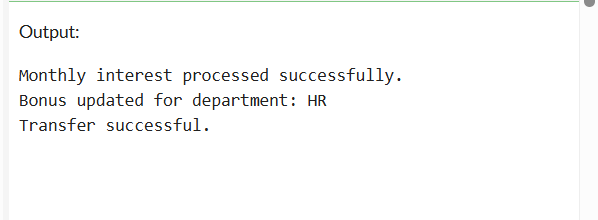
END;

/

BEGIN

TransferFunds(1, 2, 100);

END;/



**Exercise 4: Functions**

**QUERY:**

-- Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 10000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Alice Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 5000, SYSDATE);

COMMIT;

-- Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts VALUES (2, 1, 'Checking', 3000, SYSDATE);

COMMIT;

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;

/

SELECT Name, DOB, CalculateAge(DOB) AS Age FROM Customers;

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_annual\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_months NUMBER;

v\_emi NUMBER;

BEGIN

v\_monthly\_rate := p\_annual\_rate / (12 \* 100);

v\_months := p\_years \* 12;

v\_emi := (p\_loan\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_months)) /

(POWER(1 + v\_monthly\_rate, v\_months) - 1);

RETURN ROUND(v\_emi, 2);

END;

/

SELECT CalculateMonthlyInstallment(50000, 7.5, 5) AS MonthlyEMI FROM dual;

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN NUMBER IS -- 1 = TRUE, 0 = FALSE

v\_balance NUMBER;

BEGIN

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

IF v\_balance >= p\_amount THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

/

DECLARE

result NUMBER;

BEGIN

result := HasSufficientBalance(1, 1000);

IF result = 1 THEN

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

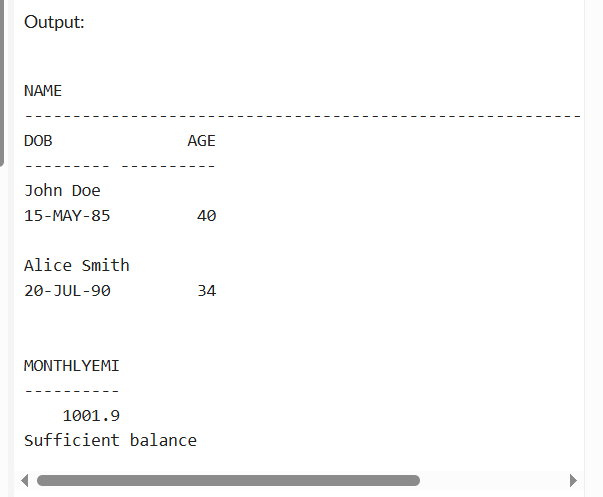
ELSE

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

END IF;

END;

/



**Exercise 5: Triggers**

**QUERY:**

-- Customers Table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- 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) -- 'Deposit' or 'Withdrawal'

);

-- AuditLog Table

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10),

Timestamp DATE

);

-- Insert Customers

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15','YYYY-MM-DD'), 10000, SYSDATE);

-- Insert Accounts

INSERT INTO Accounts VALUES (1, 1, 'Savings', 5000, SYSDATE);

COMMIT;

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10),

Timestamp DATE

);

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10),

Timestamp DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AccountID, Amount, TransactionType, Timestamp)

VALUES (:NEW.AccountID, :NEW.Amount, :NEW.TransactionType, SYSDATE);

END;

/

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Get current balance

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

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > v\_balance THEN

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

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

END;

/

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10) -- 'Deposit' or 'Withdrawal'

);

-- Insert a deposit (valid)

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 1000, 'Deposit');

-- Insert a withdrawal exceeding balance (should fail)

INSERT INTO Transactions VALUES (2, 1, SYSDATE, 10000, 'Withdrawal');

-- Check audit log entries

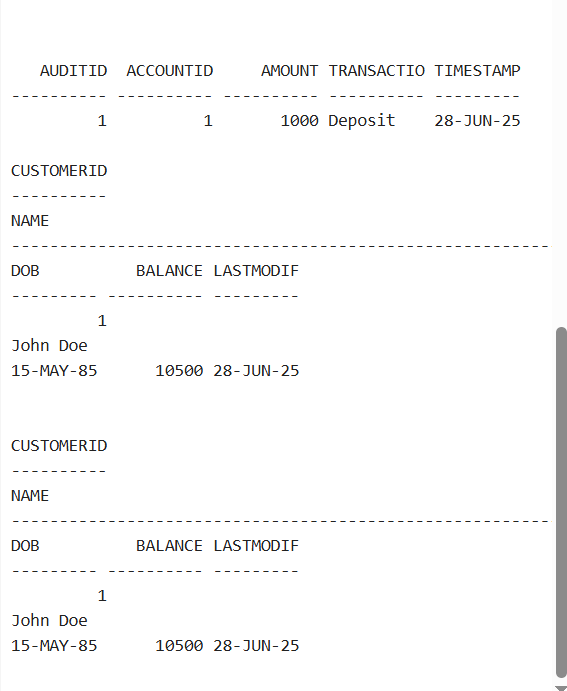
SELECT \* FROM AuditLog;

-- Update a customer to test trig

UPDATE Customers SET Balance = Balance + 500 WHERE CustomerID = 1;

SELECT \* FROM Customers;

**OUTPUT:**



**Exercise 6: Cursors**

QUERY:

-- 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, 1, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 7000, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 1000, 'Deposit');

INSERT INTO Transactions VALUES (2, 1, SYSDATE, 500, 'Withdrawal');

INSERT INTO Transactions VALUES (3, 2, SYSDATE, 2000, 'Deposit');

COMMIT;

DECLARE

CURSOR monthly\_txns IS

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

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

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

v\_cust\_id NUMBER;

v\_acc\_id NUMBER;

v\_date DATE;

v\_amt NUMBER;

v\_type VARCHAR2(10);

BEGIN

OPEN monthly\_txns;

LOOP

FETCH monthly\_txns INTO v\_cust\_id, v\_acc\_id, v\_date, v\_amt, v\_type;

EXIT WHEN monthly\_txns%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || v\_cust\_id ||

' | Account ' || v\_acc\_id ||

' | Date ' || TO\_CHAR(v\_date, 'DD-MON') ||

' | ' || v\_type || ': ' || v\_amt);

END LOOP;

CLOSE monthly\_txns;

END;

/

DECLARE

CURSOR all\_accounts IS

SELECT AccountID, Balance FROM Accounts;

v\_acc\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_fee NUMBER := 500;

BEGIN

OPEN all\_accounts;

LOOP

FETCH all\_accounts INTO v\_acc\_id, v\_balance;

EXIT WHEN all\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_fee,

LastModified = SYSDATE

WHERE AccountID = v\_acc\_id;

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || v\_fee || ' applied to Account ' || v\_acc\_id);

END LOOP;

CLOSE all\_accounts;

COMMIT;

END;

/

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE

);

DECLARE

CURSOR all\_loans IS

SELECT LoanID, InterestRate FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_rate Loans.InterestRate%TYPE;

v\_increment NUMBER := 1.5;

BEGIN

OPEN all\_loans;

LOOP

FETCH all\_loans INTO v\_loan\_id, v\_rate;

EXIT WHEN all\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_rate + v\_increment

WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated Loan ' || v\_loan\_id || ' to new rate: ' || (v\_rate + v\_increment));

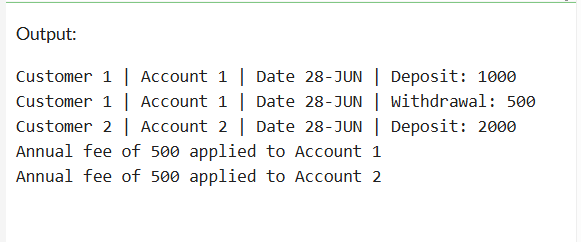
END LOOP;

CLOSE all\_loans;

COMMIT;

END;

/

****

**Exercise 7: Packages**

**QUERY:**

-- Customers Table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Employees Table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Accounts Table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15','YYYY-MM-DD'), 10000, SYSDATE);

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 5000, SYSDATE);

COMMIT;

-- Spec

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(

p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER

);

PROCEDURE UpdateCustomerBalance(

p\_id NUMBER, p\_new\_balance NUMBER

);

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

-- Body

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;

END;

PROCEDURE UpdateCustomerBalance(p\_id NUMBER, p\_new\_balance NUMBER) IS

BEGIN

UPDATE Customers

SET Balance = p\_new\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_id;

COMMIT;

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;

END;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(

p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER

);

PROCEDURE UpdateCustomerBalance(

p\_id NUMBER, p\_new\_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;

END;

PROCEDURE UpdateCustomerBalance(p\_id NUMBER, p\_new\_balance NUMBER) IS

BEGIN

UPDATE Customers

SET Balance = p\_new\_balance,

LastModified = SYSDATE

WHERE CustomerID = p\_id;

COMMIT;

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;

END;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(

p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2,

p\_salary NUMBER, p\_department VARCHAR2

);

PROCEDURE UpdateEmployeeDetails(

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\_department VARCHAR2

) IS

BEGIN

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

VALUES(p\_id, p\_name, p\_position, p\_salary, p\_department, SYSDATE);

COMMIT;

END;

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

BEGIN

UPDATE Employees

SET Salary = p\_salary

WHERE EmployeeID = p\_id;

COMMIT;

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;

END;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(

p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER

);

PROCEDURE CloseAccount(p\_account\_id NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(

p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER

) IS

BEGIN

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

VALUES(p\_account\_id, p\_customer\_id, p\_type, p\_balance, SYSDATE);

COMMIT;

END;

PROCEDURE CloseAccount(p\_account\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_account\_id;

COMMIT;

END;

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO v\_total FROM Accounts WHERE CustomerID = p\_customer\_id;

RETURN v\_total;

END;

END AccountOperations;

/

-- Call from CustomerManagement

BEGIN

CustomerManagement.AddCustomer(2, 'Alice Smith', TO\_DATE('1990-07-20','YYYY-MM-DD'), 5000);

DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(2));

END;

/

-- Call from EmployeeManagement

BEGIN

EmployeeManagement.HireEmployee(3, 'Bob Brown', 'Developer', 60000, 'IT');

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.GetAnnualSalary(3));

END;

/

-- Call from AccountOperations

BEGIN

AccountOperations.OpenAccount(3, 2, 'Checking', 2000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalBalance(2));

END;

/

BEGIN

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

END;

/